UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Posgrado en Ciencia e Ingeniería de la Computación

ESTUDIO Y REALIZACIÓN DE MECANISMOS DE SEGURIDAD INFORMÁTICA PARA TARJETAS INTELIGENTES "SMART CARD", UTILIZANDO UN MODO ROBUSTO DE IMPLEMENTACIÓN

T E S I S

QUE PARA OBTENER EL TÍTULO DE

Maestro en Ingeniería de la Computación

P R E S E N T A:

HUMBERTO FERRER CAMACHO

DIRECTOR DE TESIS:

DR. FRANCISCO GARCÍA UGALDE

México D.F., 1 de octubre de 2008

[1] Dedicada a

Mi hija Dayra A. Ferrer Navarro
por ser mi razón de continuar.

Agradecimientos

A mi esposa Lidia Deyanira Navarro Rojas y a Daniela Y., por apoyar y comprender que no pude dedicarles el tiempo que se merecían para poder lograr esta meta.

A mis papás, Humberto Ferrer Espadín y Teresa Silvia Camacho García. Por enseñarme que lo más importante es nunca desistir, que sin su apoyo y enseñanzas no habría logrado esta meta.

A mi tutor Francisco García Ugalde que gracias a su guía, enseñanzas y su paciencia fue un pilar importante de este logro.

A mis profesores, quienes con sus enseñanzas y la disciplina aprendida en sus clases, me dieron las herramientas para lograr este trabajo.

A mi jefe Alejandro Obregón y a Reyna Ramos, por haberme dado el tiempo para dedicarle a esta meta y apoyarme aún cuando eso implicara mayor trabajo para ellos.

A todos mis compañeros de la maestría en especial a Carlitos, Marco, Martha, Paco, Pedro y Yasmine, ya que mucho del trabajo realizado fue en equipo. Sobre todo les agradezco por brindarme su amistad y apoyo incondicional.

Y todos aquellas personas que me apoyaron para poder hacer este sueño realidad.

Índice general

Índice de Figuras

Índice de tablas

Resumen

Con frecuencia las *smart cards* o tarjetas inteligentes son utilizadas en aplicaciones de seguridad, debido a esta razón los métodos de autentificación y la manera en que almacenan la información de forma segura son partes fundamentales de sus sistemas operativos. En el presente trabajo se describe cómo se mejoraron estos puntos, utilizando el sistema operativo *sosse* (*Simple Operating System for Smartcard Education*) en las *smart cards*.

El proceso se llevó a cabo utilizando el lenguaje ensamblador de dichas tarjetas, en caso de este trabajo se utilizó una ATMega163 la cual utilizan una arquitectura RISC (*Reduced Instruction Set Computer*) de 8 bits, debido a que es un lenguaje de bajo nivel brinda la ventaja de poder optimizar las implementaciones al máximo.

El sistema operativo *sosse* posee un algoritmo criptográfico llamado TEA (*Tiny Encryption Algorithm*), este algoritmo es muy débil en términos de seguridad, por esta razón en este trabajo se añade al sistema operativo el algoritmo criptográfico AES (*Advanced Encryption Standard*), para lograr esto se desarrollaron varias versiones y se selecciona la mejor de estas para ser incluida, aunque esta tarea ya se ha realizado en trabajos anteriores como [5] o [6], estos manejan llaves de dimensión reducidas de 128 bits, en el presente trabajo se logra trabajar con llaves de 256 bits, obteniendo un equilibrio entre la dimensión de código, los tiempos de ejecución y el uso de la memoria, que son los criterios de optimización tomados.

El AES se basa en cuatro transformaciones que son *SubBytes()*, *ShiftRows()*, *MixColumns()* y *AddRoundKey()*. Para estas se contempla desde puntos de vista teórico y se ratifican en el campo práctico las mejores optimizaciones para una implementación en la arquitectura RISC de 8 bits.

Capítulo 1

Introducción

1.1. Hipótesis

Se desea lograr un aumento de seguridad al utilizar las *smart card* sin perder de vista las limitaciones de las mismas.

1.2. Antecedentes

Una de las mayores ventajas de las *smart cards* o "tarjetas inteligentes" es el alto nivel de seguridad que estas brindan. Una *smart card* completa consiste de: Una CPU [1], una memoria ROM [2], una memoria *EEPROM*[3], una memoria *RAM*[4]; son muy convenientes debido a sus dimensiones reducidas, aunque también por su tamaño reducido tienen limitaciones tanto en memoria como en capacidad de cómputo, en los últimos años esta capacidad se ha incrementado bastante pero sigue siendo una limitante en la capacidad criptográfica de estas. Uno de los sistemas operativos (COS)[5] que soportan estas tarjetas es el *sosse*[6], este incluye un algoritmo de cifrado simétrico, el algoritmo TEA[7], es bastante limitado y no es un estándar oficial. Así que se han dado propuestas para implantar un algoritmo estándar, el llamado Advanced Encryption Standard (AES). Por diseño, este algoritmo soporta un manejo de llaves de 128, 192 y 256 bits.

Debido a que los algoritmos comerciales simétricos (como DES[8], Triple DES[9], AES) son algoritmos de cifrado por bloques, o sea operan con bloques de bits de longitud fija, a los cuales se les aplica una transformación. Al estándar AES se le establecieron varios modos de operación de las unidades de cifrado por bloques, algunos de estos modos van encadenando

[1] Unidad central de procesamiento, *Central Processing Unit*

[2] Memoria de solo lectura, *Read Only Memory*

[3] Memorias programables reescribirles eléctricamente, *Electrically erasable programmable memory*

[4] Memoria de acceso aleatorio o memoria de acceso directo, *Random Access Memory*

[5] Sistema operativo para las tarjetas inteligentes, *Card Operating System*

[6] Sistema operativo para las tarjetas inteligenes con licencia de código abierto,*Simple Operating System for Smartcard Education*

[7] Pequeño algoritmo de cifrado, *Tiny Encryption Algorithm*

[8] Es un algoritmo de cifrado, es decir, un método para cifrar información, escogido como FIPS en los Estados Unidos en 1976, *Data Encryption Standard*

[9] Triple DES se llama al algoritmo que hace triple cifrado del DES

las salidas de un bloque con el siguiente. Estos modos son conocidos como: *Electronic codebook* (ECB), *Cipher-block chaining* (CBC), *Cipher feedback* (CFB), *output feedback* (OFB) y *Counter* (CTR).

1.3. ¿Qué es el algoritmo AES?

El Advanced Encryption Standard (AES), también conocido como Rijndael, es un esquema de cifrado por bloque, este estándar fue desarrollado por los criptólogos Belgas, Joan Daemen y Vincent Rijmen. Este algoritmo utiliza tamaños de llave de 128, 192 ó 256 bits. Al contrario de su predecesor DES, Rijndael es una red de permutación y sustituciones, no una red de Feistel[10]. AES es más rápido tanto en *software* como en *hardware*, es relativamente fácil de implementar y requiere menor cantidad de memoria.

El AES fue elegido en un concurso internacional ya que la longitud mínima de la llave es de 128 bits, los ataques a fuerza bruta con tecnología actual y proyectada eran considerados no prácticos. Desde su definición en 2001 no se han detectado vulnerabilidades en criptoanálisis. El AES tiene una alta eficacia de cómputo, lo que permite su utilización en aplicaciones que requieran alta velocidad. Además la implementación de este es más fácil que la de su predecesor, lo que implica una mayor flexibilidad tanto en las implementaciones en *software*, como en *hardware*.

En el algoritmo AES la mayoría de los cálculos se hacen en un campo finito determinado, lo que proporciona la ventaja de poder manejar las palabras como bytes.

El AES opera con un arreglo de 4×4 bytes, llamado *state*. Para el cifrado, cada ronda de la aplicación del algoritmo AES (excepto la última) consiste en cuatro pasos llamados:
SubBytes() , *ShiftRows()*, *MixColumns()* , *AddRoundKey()*

La ronda final omite la transformación *MixColumns()*.

1.4. Objetivos

Este trabajo de tesis se ha planteado implementar una versión del algoritmo de cifrado AES, en una tarjeta inteligente "*smart card*". Esta implementación debe manejar una llave de 256 bits y debe de ser eficiente tanto en tamaño, como en tiempo de procesamiento, debido a las limitaciones de estas tarjetas. En principio se manejara el modo de operación ECB, debido a que se ajusta al modelo de autenticación "reto respuesta"[11].

[10] Es un método de cifrado en bloque con una estructura particular, presentan la ventaja de ser reversibles por lo que las operaciones de cifrado y descifrado son idénticas, requiriendo únicamente invertir el orden de las subclaves utilizadas, el más popular de estos es el DES.

[11] También llamado modelo de autentificación externa, ya que el *host* genera un reto, este se manda a la *smart card*, la smart card aplica el algoritmo de cifrado a la cadena, manda la respuesta al *host*, este cifra aplica la cadena original y lo compara con la cadena recibida, si las cadenas son iguales se acepta la autentificación, de lo contrario se rechaza.

1.5. Metas

Se pretende incrementar la seguridad que ofrecen las tarjetas inteligentes "*smart cards*" mejorando las implementaciones existentes del algoritmo AES que se han documentado, al incrementar el tamaño de la llave, optimizando los escasos recursos que se tienen en dichas tarjetas, tanto en tamaño de código, tiempo de ejecución y utilizando muy poca memoria *RAM*.

1.6. Justificación

Una *smart card* es aproximadamente del tamaño de una tarjeta de crédito (o más pequeña como son las tarjetas *SIM*[12] de teléfonos celulares) y es capaz de proveer servicios de seguridad.

Algunos de sus usos de estas son en tarjetas de crédito, *SIM* para telefonía móvil, identificación de alta seguridad, tarjetas de control de acceso, almacenamiento de historiales médicos de un paciente, etc.

Los protocolos criptográficos protegen el intercambio de dinero entre la tarjeta inteligente y la máquina receptora de una manera segura y confiable.

Cuando las tarjetas posen algoritmos criptográficos, las posibilidades de identificación y autentificar se multiplican ya que se pueden almacenar datos de forma segura como son los certificados digitales, historiales médicos en archivos protegidos dentro de la tarjeta, de modo que estos elementos privados nunca salgan de la tarjeta y las operaciones de autentificación se realicen a través de la *smart card*.

Algunas de las aplicaciones de las tarjetas inteligentes son:

- Identificación digital, permite la validación de la identidad del portador.
- Control de acceso, para restringir, o permitir el acceso en una determinada área.
- Tarjetas de identidad médica, identificar al paciente y almacenar los principales datos de su historial clínico.
- Firma Digital, almacenar el certificado digital de forma segura.

Otros ejemplos en que se pueden ocupar [7], es integrar un lector en una lap[13]/pc[14], modificar el *bios*[15] para que lea como parte inicial del arranque. La *smart card* valida el arranque, carga la autentificación de los hosts, etc. Esto con la finalidad de aumentar la seguridad de los equipos cuando esta es un factor crítico.

[12] Módulo de Identificación del Suscriptor es una tarjeta inteligente desmontable usada en teléfonos móviles que almacena de forma segura la clave de servicio del suscriptor usada para identificarse ante la red, *Subscriber Identify Module*

[13] Laptop o computadora personal portátil

[14] Computadora personal

[15] El sistema Básico de entrada/salida es un código de interfaz que localiza y carga el sistema operativo en la RAM, *Basic Input-Output System*.

La programación de estas aplicaciones depende del sistema operativo COS, se le puede dar una mayor flexibilidad a las aplicaciones con el *sosse*, el cual es un COS pero tiene la ventaja de ser *software* libre lo que nos da la posibilidad de tener acceso a los fuentes y poder modificarlo, con la finalidad de aumentar la robustes de la seguridad que este brinda. El *sosse* utiliza como algoritmo de cifrado al TEA, que se ha demostrado es un algoritmo vulnerable. Por esta razón al incorporar el AES con manejo de llaves de 256 bit se logrará mejorar la seguridad que el *sosse* brinda. Así las aplicaciones mencionadas tendrán un mayor grado de fiabilidad.

Capítulo 2

Marco teórico

2.1. Introducción

En el presente capítulo se describe la teoría básica necesaria para poder comprender el trabajo realizado. En primera instancia se da una breve introducción a la teoría de campos finitos, posteriormente se da una descripción detallada del algoritmo AES basada en el FIPS-197[1] y finalmente se explica que son las *smart cards*.

2.2. Teoría de campos finitos

La teoría de campos finitos es una rama del álgebra que relaciona las aritméticas de los campos con sus extensiones de Galois cuyo grupo es abeliano. A continuación se dan las herramientas básicas de la teoría de campos, necesarias para poder entender el razonamiento matemático del AES.

2.2.1. Grupos

Definición:
Un conjunto $\mathbb{G} \neq \emptyset$ es un grupo, si en $\mathbb{G}$ esta definida una operación $\cdot\ni\cdot$ $\forall$ a,b,c $\in \mathbb{G}$ se cumple:

1. $a \cdot b \in \mathbb{G}$ — Cerradura
2. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ — Asociatividad
3. $\exists e \in \mathbb{G} \ \cdot\ni\cdot\ a \cdot e = e \cdot a = a \ \forall a \in \mathbb{G}$ — Existencia del elemento neutro
4. $\forall a \in \mathbb{G}\ , \exists a^{-1} \in \mathbb{G} \ \cdot\ni\cdot\ a \cdot a^{-1} = a^{-1} \cdot a = e$ — Existencia del inverso

Ejemplos de grupos
$\mathbb{Z}_4 = \{0, 1, 2, 3\}$, definimos $(\cdot)$ como $\mathbb{Z}_4, a \cdot b := (a + b)\ mod\ 4$ — los enteros módulo 4 con la suma.

Viendo la tabla (2.1) se puede ver que las cuatro propiedades se cumplen.

[1] Se refiere al estándar donde se define el algoritmo de cifrado AES FIPS por las siglas *Federal Information Processing Standards*

·	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Tabla 2.1: *Definición de · en* $\mathbb{Z}_4$

Las propiedades 1 y 2 se ven claramente.
El elemento neutro es el 0 y
El inverso del 0 es el 0, del 1 es el 3, de 2 es el 2 y del 3 es el 1.

Teorema
Si G es un grupo, entonces.

I) El elemento identidad, o neutro es único

II) $\forall\ y,x,a \in G \cdot\ni\cdot\ a\cdot x = a\cdot y \rightarrow x = y$

III) $\forall\ a \in G$, $\exists$ un único inverso en G.

IV) $\forall\ a \in G$, $(a^{-1})^{-1} = a$

Para ver la demostración, se puede consular [3].

Definición:

Sea $\mathbb{H}$ con $\emptyset \neq \mathbb{H} \leq \mathbb{G}$ es un subgrupo de $\mathbb{G}$ si $\mathbb{H}$ es un grupo en si bajo la misma operación de $\mathbb{G}$.

Ejemplo.
Sea $\mathbb{G}$ el plano cartesiano y $\mathbb{H}$ una recta que pasa por el origen (ver figura 2.1).

$\mathbb{G}$ un grupo
$\mathbb{H}$ un subgrupo de $\mathbb{G}$
Eso ya que cumple $\mathbb{H}$ las propiedades de grupo y $\mathbb{H} \subseteq \mathbb{G}$ bajo la operación $(\cdot)$.

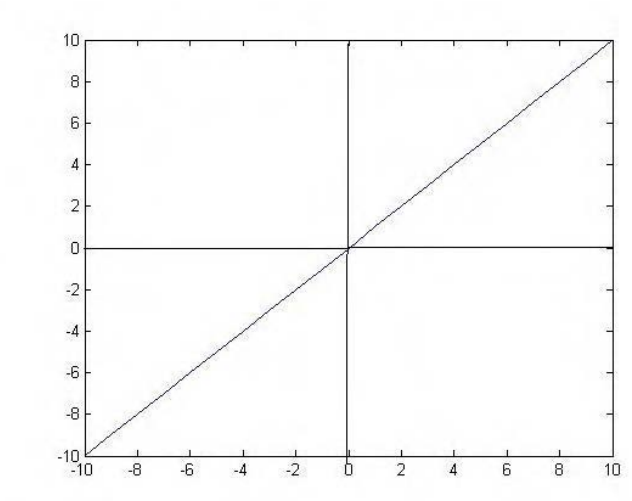

Figura 2.1: *Ejemplo de un grupo y un subgrupo*

Definición:
Sea G un grupo, diremos que G es cíclico si $\exists\ a \in G \cdot\ni\cdot \forall\ b \in G, \exists\ r \in \boldsymbol{N} \setminus \{0\} \cdot\ni\cdot a^r = b$, esto es:

$<a> := \{a^r |\ r \in \boldsymbol{N} \setminus \{0\}\ \} = G$
Es decir para todo elemento b existe un elemento a que elevado a una potencia entera mayor a cero llega a ser b.

Teorema:
Sea G un grupo finito y sea $a \in G$ entonces $<a>$ es un subgrupo de G.
El generado de a es un subconjunto de G.

Demostración.

Es necesario ver si cumple las propiedades 1 y 4.
La 1 se cumple directamente. □

La 4, se logra si va multiplicando un elemento por si mismo, en un cálculo se alcanzará al elemento identidad y habremos encontrado el inverso, esto por que G es finito. □

2.2.2. Anillos

Definición:

Un anillo R es un conjunto con dos operaciones que comúnmente se denotan por $(+,\cdot)$ que satisfacen las siguientes propiedades:

1) R es un grupo conmutativo bajo $(+)$
2) $\cdot$ es cerrado en R
3) $\cdot$ es asociativo en R
4) Las operaciones de $+$ y $\cdot$ cumplen con la ley distributiva
   $a \cdot (b + c) = a \cdot b + a \cdot c$
   $(b + c) \cdot a = b \cdot a + c \cdot a$

Ejemplos.
Enteros, reales, complejos
$\mathbb{Z}\ ,+,\cdot \rightarrow$ es un anillo

Teorema.
Sea R un anillo entonces $\forall\ a,b \in R$ se tiene.

1) $a \cdot 0 = 0 \cdot a = 0$
2) $a \cdot (-b) = (-a) \cdot b = -(a \cdot b)$

Demostración.

1) $a\cdot 0 = a\cdot (0+0) = a\cdot 0+a\cdot 0 \rightarrow a\cdot 0 = a\cdot 0+a\cdot 0 : 0 = a\cdot 0$ □

2) $0 = a\cdot 0 = a\cdot (b-b) = a\cdot b+a\cdot (-b) \rightarrow 0 = a\cdot b+a\cdot (-b)$

Sumando en ambos lados de la ecuación el inverso aditivo
$-(a\cdot b)+0 = a\cdot b+a\cdot (-b)-(a\cdot b) \rightarrow a\cdot(-b) = -(a\cdot b)$ □

2.2.3. Campo

Definición.

Un campo F es un conjunto donde existen definidas dos operaciones $(+,\cdot)$ en donde se cumple.

1) El conjunto F es un grupo conmutativo bajo $+$

2) F es cerrado $(\cdot)$ y el conjunto de todos los elementos distintos de cero (el neutro bajo $+$) es un grupo conmutativo bajo $(\cdot)$

3) $\forall\ a,b \in F,\ (a+b)\cdot c = a\cdot c+b\cdot c$

Ejemplos.

El conjunto de los números reales, los números racionales, los números complejos, el campo finito de los números enteros módulo dos, o también conocido como campo de Galois $\mathbb{Z}_2 = GF(2) = \{0,1\}$ con

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

El campo finito $\mathbb{Z}_3 = GF(3) = \{0,1,2\}$ con

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

En general los números enteros módulo n, donde n es primo o una potencia de un primo. Es decir $\mathbb{Z}_n = GF(n) \Leftrightarrow n = p^m$ para p primo > 1 y m entero ≥ 1

Un ejemplo de un campo finito de Galois es $GF(16) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, A, B, C, D, E, F\}$

Representación decimal	Representación hexadecimal	Representación binaria	Representación polinomial
0	0	0000	0
1	1	0001	1
2	2	0010	α
3	3	0011	α+1
4	4	0100	α^2
5	5	0101	α^2 +1
6	6	0110	α^2+α
7	7	0111	α^2+α+1
8	8	1000	α^3
9	9	1001	$\alpha^3 + 1$
10	A	1010	α^3+α
11	B	1011	α^3+α+1
12	C	1100	α^3+α^2
13	D	1101	α^3+α^2+1
14	E	1110	α^3+α^2+α
15	F	1111	α^3+α^2+α+1

De acuerdo al trabajo de esta tesis se debe considerar las siguientes operaciones:

Si se utiliza la multiplicación y la suma sobre enteros se corre el riesgo de obtener ceros indeseados, por ejemplo $8 * 2 mod 16 = 0$. Por esta razón se usan las sumas y multiplicaciones sobre polinomios, se debe hacer un módulo con un polinomio irreducible para que no se pase del grado. Debe ser irreducible ya que no queremos que se pueda factorizar en 2 más pequeños.

Ejemplos de operaciones, tomando en cuenta al polinomio $\pi(\alpha) = \alpha^4 + \alpha + 1$, se tiene

Suma
$6 + A = (\alpha^2 + \alpha) + (\alpha^3 + \alpha) = \alpha^3 + \alpha^2 = C$
Multiplicación
$6 \cdot A = (\alpha^2 + \alpha) \cdot (\alpha^3 + \alpha) = [\alpha^5 + \alpha^4 + \alpha^3 + \alpha^2] \bmod \pi(\alpha) = \alpha^3 + 1 = 9$

$$\begin{array}{r} \alpha+1 \\ \alpha^4+\alpha+1 \overline{)\alpha^5+\alpha^4+\alpha^3+\alpha^2} \\ \alpha^3+1 \end{array}$$

Note que de cualquier elemento en este campo, su negativo o inverso es el mismo ya que la suma y la resta son lo mismo.

División.

Para encontrar la división A/B, lo que se debe de hacer es encontrar B^{-1}=1/B entonces utilizar la regla

$$\frac{A}{B} = A \cdot B^{-1}$$

Teorema.

Si $\pi(\alpha)$ es un polinomio de grado 4 irreducible sobre $GF(2)$ entonces todo polinomio $\beta(\alpha) \neq 0$ de grado menor a 4 tiene un inverso único $\beta^{-1}(\alpha)$ $\cdot\ni\cdot$ $\beta(\alpha)\beta^{-1}(\alpha) \equiv 1 \bmod \pi(\alpha)$

Demostración.

Los polinomios irreducibles de grado 4 son 3.

Sea $P = \{1, \alpha, \alpha+1, \alpha^3+\alpha^2+\alpha+1\}$ y sea $T = \{\beta(\alpha)\cdot A_i(\alpha) | A_i(\alpha) \in P\} \subseteq P$

Pd. $T=P$

Sup. $A_1(\alpha)\ \beta(\alpha) = A_2(\alpha)\ \beta(\alpha)\ mod\ \pi(\alpha) \rightarrow \pi(\alpha) \mid (\ A_1(\alpha)$ - $A_2(\alpha)\ )\ \beta(\alpha) \rightarrow$
$\pi(\alpha) \mid (\ A_1(\alpha)$ - $A_2(\alpha)\ )$ ó $\pi(\alpha) \mid \beta(\alpha) \rightarrow \pi(\alpha) \mid (\ A_1(\alpha)$ - $A_2(\alpha)\ ) \rightarrow$
$A_1(\alpha) = A_2(\alpha)$ *por lo tanto* $\exists\ A(\alpha) \in P \cdot\ni\cdot\ A(\alpha)\ \beta(\alpha) = 1 \therefore A\ (\alpha) = \beta^{-1}\ (\alpha)$ □

Para construir un $GF(3)^2$ se debe tener un polinomio irreducible de grado 2 sobre $GF(3)$, este es un polinomio ternario. -Así de forma general para generar $GF(p^m)$ se tiene que buscar un polinomio irreducible de grado m sobre $GF(p)$-.

Ejemplos. $GF(3^2) = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

Son polinomios ternarios a lo sumo de grado 1 y el polinomio irreducible para este es $\pi(\alpha)=\alpha^2+\alpha+2$

0	00	0
1	01	1
2	02	2
3	10	α
4	11	α+1
5	12	α+2
6	20	2α
7	21	2α+1
8	22	2α+2

Construcción de un $GF(2^4)$

$GF(2^4) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, A, B, C, D, E, F\}$

$\pi(\alpha) = \alpha^4 + \alpha + 1$ Si tomamos a α como raíz tenemos $\alpha^4 + \alpha + 1 = 0 \longrightarrow \alpha^4 = \alpha + 1$ Sustituyendo el valor de α^4 obtenemos la tabla.

0	0	0000	0
1	1	0001	1
α	α	0010	2
α^2	α^2	0100	4
α^3	α^3	1000	8
α^4	α+1	0011	3
α^5	α^2+α	0110	6
α^6	α^3+α^2	1100	C
α^7	α^3+α+1	1011	B
α^8	α^2+1	0101	5
α^9	α^3+α	1010	A
α^{10}	$\alpha^2 + \alpha + 1$	0111	7
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	1110	E
α^{12}	α^3+α^2+α+1	1111	F
α^{13}	$\alpha^3 + \alpha^2 + \alpha$	1110	D
α^{14}	$\alpha^3 + 1$	1001	9
α^{15}	1	0001	1

Las operaciones se realizan sobre los exponentes tomando módulo 2^4.
Ejemplo de algunas operaciones.

$6^{-1} = (\alpha^5)^{-1} = \alpha^{-5} = \alpha^{15-5} = \alpha^{10} = 7$

$A \cdot 7 = \alpha^9 \cdot \alpha^{10} = \alpha^{19} = \alpha^{15-19} = \alpha^4 = 3$

$\sqrt{6} = (\alpha^5)^{1/2} = (\alpha^{15+5})^{1/2} = (\alpha^{20})^{1/2} = \alpha^{10} = 7$

Al considerar un byte como un conjunto de 8 bits, es práctico utilizar GF(2^8). Los elementos de este campo se muestran en el Apéndice A.

2.3. AES (FIPS-197)

Después de tres años de competencia internacional, el AES fue anunciado por la *National Institute of Standards and Technology* (NIST) en *Federal Information Processing Standards* (FIPS) *Publication* 197, como el *Advanced Encryption Standard* (AES) en el año 2001, considerada una técnica de cifrado aprobada para ser usada por el Gobierno de los Estados Unidos de América, empresas privadas y particulares. Al ser implementado como un componente clave del protocolo de seguridad en general, el AES permite un alto grado de seguridad criptográfica además de ser eficiente y rápido. Esta técnica de cifrado utiliza llaves de 128, 192 y 256 bits.

La diferencia entre el AES y el Rijndael es el rango de valores soportados de la longitud de bloque y la longitud de la llave. El Rijndael varia el tamaño de los bloques, tanto en la longitud de bloque como de la llave, esto quiere decir que son especificados los tamaños de manera independiente siempre y cuando sean múltiplos de 32 bits. Esto le da al Rijndael una mayor robustez.

El AES solo permite bloques de 128 bits y soporta longitudes de llave de 128, 192 y 256 bits. Los bloques extra y las longitudes de llaves extras tratadas por el Rijndael no son evaluadas en la sección de proceso del AES, por lo tanto no están tomadas en la cuenta en especificación del estándar FIPS-197.

2.3.1. Referencias sobre los parámetros del algoritmo, símbolos y funciones

En esta parte se da un breve descripción sobre los términos que se van en la descripción del algoritmo AES, más adelante se explican con mayor detalle.

AddRoundKey — Cada byte del estado es combinado con la clave de la ronda, usando la operación de XOR.

InvMixColumns — Transformación inversa del *MixColumns*.

InvShiftRow — Transformación inversa del *ShiftRow*.

InvSubBites — Transformación inversa del *SubBites*.

K — Llave de cifrado.

MixColumns — operación de mezclado que opera en las columnas del estado, combinando los cuatro bytes en cada columna usando una transformación lineal.

N_b — Número de columnas del estado. Para este estándar es $N_b = 4$.

N_k — Número de palabras de 32-bits de la llave cifrada. Para este estándar son los valores de $N_k = 4$, 6 y 8.

N_r — Número de rondas de acuerdo a *Nb* y *Nk*. Para este estándar son los valores de $N_r = 10$, 12 y 14.

RotWord — Función usada en la rutina de expansión de llave la cual recibe cuatro valores y realiza una permutación cíclica.

ShiftRows — Transformación de cifrado, en este paso se realiza una transposición donde cada fila del estado es rotado de manera cíclica un número determinado de veces.

SubBytes — Transformación de cifrado, en este paso se realiza una sustitución no lineal donde cada byte es reemplazado con otro de acuerdo a una tabla *lookup* (S-box).

SubWord — Función usada en la rutina de expansión de llave, toma cuatro bytes como una palabra, aplica la S-box y regresa una palabra.

XOR — Operación de o-exclusiva.

$\oplus$ — Operación de o-exclusiva.

$\cdot$ — Multiplicación en campo finito.

2.3.2. Notación y convenciones

La entrada y la salida para el algoritmo de AES es una secuencia de 128 bits. Estas secuencias son referidas como los bloques y el número de bits de estas es referido como su longitud. La llave de cifrado para el AES, es una secuencia de 128, 192 o 256 bits.

El byte: La unidad de procesamiento del algoritmo de AES es el byte, es una secuencia de bites tratados como una sola entrada. La entrada, la salida y la llave de cifrado se toman como un arreglo de bytes, las entradas pueden denotarse como un "a" y las salidas como a_n ó a[n]. Donde n esta en los siguientes rangos.

Longitud de la llave = 128 bits, $0 \leq n < 16$; longitud del bloque = 128 bits, $0 \leq n < 16$;
Longitud de la llave = 192 bits, $0 \leq n < 24$;
Longitud de la llave = 256 bits, $0 \leq n < 32$.

Todos los valores en el AES se tratan como una concatenación de bits individuales (0 ó 1) {b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0}. Estos bytes pueden ser interpretados como elementos de un campo finito usando una representación polinomial:

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i$

Ejemplo

{01100011} se identifica en el campo como $x^6 + x^5 + x + 1$

Es también conveniente denotar valores en formato hexadecimal por lo tanto el elemento {01100011} se puede representar como {63}, donde el carácter que denota al grupo de cuatro-bits que contiene los bits más significativos se localiza a la izquierda.

Algunas operaciones en campo finitos implican un bit adicional (b_8) a la izquierda. Donde el byte adicional, aparecerá como {01} inmediatamente antes del byte de 8 bits; por ejemplo, una secuencia de 9-bits será presentada como {01}{1b}.

Un arreglo de bytes se puede representar de la siguiente manera:

$a_0\, a_1\, a_2\, a_3\, a_4\, a_5\, a_6\, a_7\, a_8\, a_9\, a_{10}\, a_{11}\, a_{12}\, a_{13}\, a_{14}\, a_{15}$

Este arreglo de bytes también puede verse como bits ordenados de la siguiente manera:

$input_0\, input_1\, input_2 \ldots input_{126}\, input_{127}$

donde

$a_0 = input_0 input_1 \ldots input_7$
$\ldots$
$a_{15} = input_{120} input_{121} \ldots input_{127}$

El estado: Internamente, las operaciones del algoritmo AES utilizan un arreglo de bytes de dos dimensiones llamado estado. El estado consiste en cuatro filas de bytes, cada una contiene *Nb* bytes, donde *Nb* es la longitud del bloque dividido entre 32. Este arreglo se denota con *s* y cada byte individual contiene dos índices *r* y *c*, el número de fila y de columna respectivamente. Ejemplo s_{rc} o *s[r,c]*.

bytes de entrada

in_0	in_4	in_8	in_12
in_1	in_5	in_9	in_13
in_2	in_6	in_10	in_14
in_3	in_7	in_11	in_15

$\rightarrow$

estado

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

$\rightarrow$

bytes de salida

out_0	out_4	out_8	out_12
out_1	out_5	out_9	out_13
out_2	out_6	out_10	out_14
out_3	out_7	out_11	out_15

Figura 2.2: *Del lado izquierdo se encuentra el estado inicial (texto en claro), en el centro el estado mientras se procesa el AES y de lado derecho el estado final o salida (texto cifrado).*

Por lo tanto, al principio del cifrado o del descifrado, el arreglo de la entrada, se copia al arreglo del estado según el esquema.

El estado también puede tomarse como un arreglo de columnas: Los cuatro bytes en cada columna del estado proveen un índice si los vemos como palabras de 32 bits, donde el número *r* proporciona un índice para los cuatro bytes dentro de cada palabra. Por lo tanto, el estado se puede interpretar como un arreglo unidimensional de 32 bytes:

$w_0 = s_{0,0}\ s_{1,0}\ s_{2,0}\ s_{3,0}$ $\qquad$ $w_2 = s_{0,2}\ s_{1,2}\ s_{2,2}\ s_{3,2}$
$w_1 = s_{0,1}\ s_{1,1}\ s_{2,1}\ s_{3,1}$ $\qquad$ $w_3 = s_{0,3}\ s_{1,3}\ s_{2,3}\ s_{3,3}$

2.3.3. Especificación del algoritmo

Para el algoritmo de AES, la longitud del bloque de entrada, la del bloque de la salida y la del estado son de 128 bits. Esta es representada por $Nb = 4$, que se refiere al número de palabras de 32-bits (número de columnas) en el estado.

La longitud de la llave de la cifrado, *K*, es de 128, 192, o 256 bytes. La longitud de la llave es representada por $Nk = 4$, 6, o 8, que se refiere al número de las palabras de 32-bits (número de columnas) en la llave.

El número de rondas que se realizan durante la ejecución del algoritmo depende del tamaño de la llave. El número de rondas es representado por *Nr*, donde *Nr*=10 cuando *Nk*=4, $Nr = 12$ cuando $Nk = 6$, y $Nr = 14$ cuando $Nk = 8$. (Ver tabla 2.9.)

	Longitud de la llave (Nk palabras)	*Tamaño de bloque (Nb palabras)*	*Número de rondas (Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabla 2.9: *Combinaciones validas entre las llaves y las rondas.*

Para cifrar o descifrar, el algoritmo AES utiliza una función de ronda que se compone de cuatro diversas transformaciones orientadas a bytes, las cuales son:

1) Sustituir los bytes usando una tabla de substitución (S-box),

2) Intercambiar las filas del estado,

3) Mezclar los datos cercanos de las columna del estado, y

4) Agregar la llave de ronda al estado. Estas transformaciones (y sus inversos) se describen más adelante.

2.3.4. Cifrado

Al comenzar a cifrar, la entrada se copia en el estado. Después de aplicar *Round Key* (añadir la llave), el estado es transformado aplicando las funciones de rondas 10, 12, ó 14 veces (dependiendo de la longitud de la llave). El estado final es copiado en la salida.

La función es parametrizada utilizando el esquema de la llave que consiste en un arreglo unidimensional de palabras de cuatro bytes derivado de usar la rutina *Key Expansion*.

El cifrado se describe en el pseudo código mostrado en la tabla 2.10. Utilizando las transformaciones individuales – *SubBytes(), ShiftRows(), MixColumns(),* y *AddRoundKey()* – que procesan el estado y se describen más adelante, el arreglo *w[]* contiene el esquema de la llave.

Todas las *Nr* rondas son idénticas con excepción de la ronda final, que no incluye la transformación *MixColumns()*.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]) begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
end
```

Tabla 2.10: *Pseudo código para el cifrado del AES*

2.3.5. Transformación *SubBytes()*

La transformación *SubBytes()* es una operación de substitución no lineal de bytes, independiente de cada otro byte del estado, utiliza una tabla (S-box). Esta S-box, es invertible y esta construida por la composición de dos transformaciones:

1) Se toma el inverso multiplicativo en el campo finito $GF(2^8)$; el elemento $\{00\}$ es mapeado así mismo.

2) Se aplica la siguiente transformación afín (sobre GF(2)):

   $b'_i = b_i \oplus b_{(i+4) mod\ 8} \oplus b_{(i+5) mod\ 8} \oplus b_{(i+6) mod\ 8} \oplus c$i

   Para $0 \leq i < 8$ donde b_i es el i-esimo bit y c_i es el i-esimo bit del byte c con un valor $\{63\}$ o $\{01100011\}$.

   Una transformación afín de elementos de la matriz en la S-box puede ser representada como se muestra en la figura 2.3.

En la figura 2.4 se muestra el efecto de la transformación de *SubBytes()* en el estado.

La S-box usada en la transformación *SubBytes()* se presenta en forma hexadecimal en la tabla 2.11. Por ejemplo, si $s_{1,1} = \{53\}$, entonces el valor de la substitución estaría determinada por la intersección de la fila con el índice '5' y la columna con el '3'. Esto da lugar a que $s'_{1,1}$ tenga el valor de $\{ed\}$.

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Figura 2.3: *Representación matricial de la transformación SubBytes()*

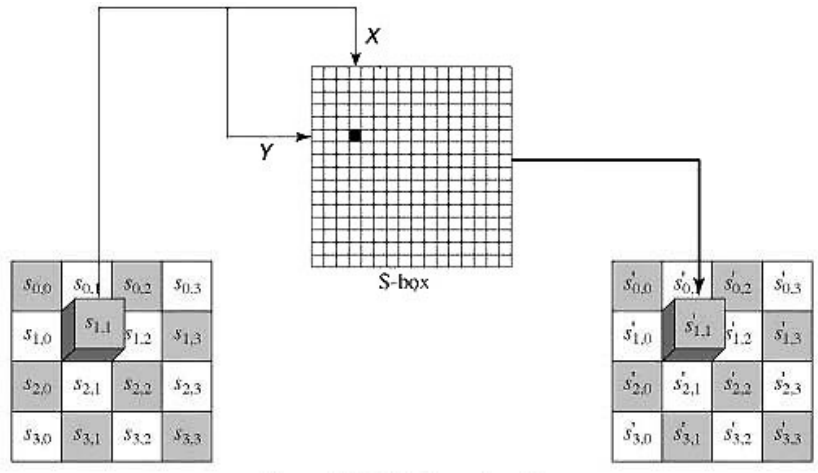

Figura 2.4: *Transformación SubBytes() aplicada a cada elemento del estado*

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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 2.11: *Tabla S-box Esta tabla esta definida en FIPS-197[2]*

2.3.6. Transformación *ShiftRows()*

En la transformación *ShiftRows()*, los bytes en las ultimas tres filas del estado son rotados cíclicamente diferente número de bytes. La primera columna r = 0, no es rotada.

La rotación de *ShiftRows()* se realiza de las siguiente manera:

$S'_{r,c} = S_{r,(c+shift(r,Nb))mod\ Nb}$ *para* $0 < r < 4$ *y* $0 \leq Nb$,

donde la función *shift(r, Nb)* depende del número de la fila, *r*, y con *Nb*=4, tal y como sigue:

shift(1,4) =1; *shift*(2,4) = 2 ; *shift*(3,4) = 3

Esto tiene el efecto de mover los bytes en las posiciones más "bajas"tantas posiciones con el número de la fila, mientras que los bytes del tope de la tabla no son movidos.

En la figura 2.5 se ilustra la transformación de *ShiftRows()*.

2.3.7. Transformación *MixColumns()*

La transformación de *MixColumns()* opera sobre el estado, columna por columna, tratando cada columna como un polinomio de grado cuatro sobre $GF(2^8)$ modulo $x^4 - 1$ con un *a(x)* como:

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

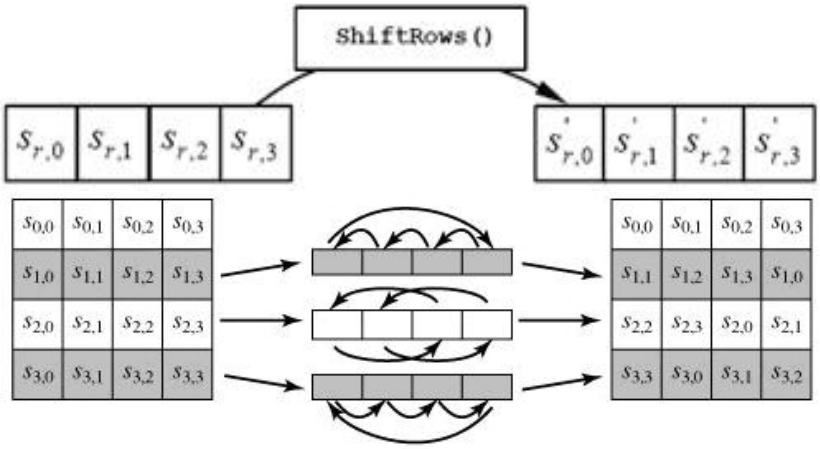

Figura 2.5: *ShiftRows().Corrimiento cíclico de las últimas tres filas del estado.*

Esto puede ser re-escrito como producto de matrices de la siguiente manera

$s'(x) = a(x) \otimes s(x)$:

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix} \qquad \text{para } 0 \leq c < Nb$$

Como resultado de esta multiplicación, los cuatro bytes de las columnas son substituidos por los siguientes

$S'_{0,c} = (\{02\} \cdot S_{0,c}) \oplus (\{03\} \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$

$S'_{1,c} = S_{0,c} \oplus (\{02\} \cdot S_{1,c}) \oplus (\{03\} \cdot S_{2,c}) \oplus S_{3,c}$

$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \cdot S_{2,c}) \oplus (\{03\} \cdot S_{3,c})$

$S'_{3,c} = (\{03\} \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \cdot S_{3,c})$

En la figura 2.6 se ilustra la transformación *MixColumns()*.

2.3.8. Transformación *AddRoundKey()*

En la transformación *AddRoundKey()*, la llave de ronda es añadida al estado por la aplicación de XOR. Cada llave de ronda consiste de *Nb* palabras que se obtienen del esquema de llave. A estas llaves de ronda también se les conoce como sub-llaves. Estas *Nb* palabras son añadidas en la columna del estado tal y como se muestra a continuación:

$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [w_{round*Nb+c}]$ para $0 \leq c < Nb$,

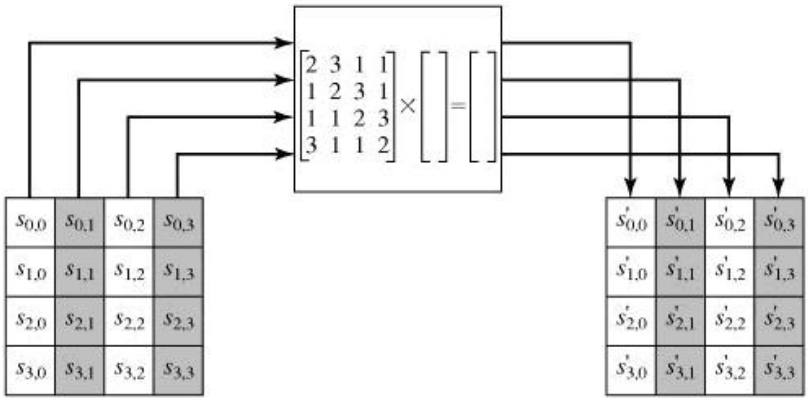

Figura 2.6: *Se ilustra la aplicación de la transformación MixColumns()*

donde $[w_i]$ es el esquema de la llave, y *round* es un valor en el rango $0 \leq round \leq Nr$. En el cifrado, al comienzo se añade la ronda inicial *round = 0*, antes de la aplicación de la función de ronda. La aplicación de la transformación *AddRoundKey()* en las *Nr* rondas ocurre cuando $1 \leq round \leq Nr$. En la figura 2.7 se ilustra esta transformación.

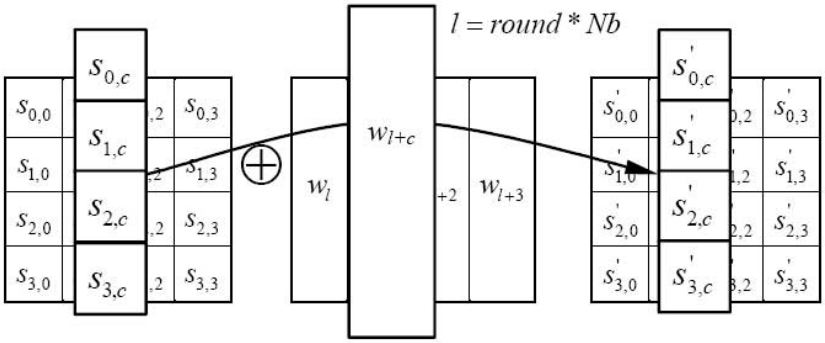

Figura 2.7: *En la transformación AddRoundKey() hace un XOR en cada columna del estado con el esquema de llave*

2.3.9. Expansión de la llave

El algoritmo de AES toma la llave de cifrado, *K*, realiza una rutina expansión de llave para generar un esquema de llave. La extensión de la llave genera *Nb * (Nr + 1)* palabras: el algoritmo requiere un conjunto inicial de *Nb* palabras, y para cada una de las *Nr* rondas requiere las *Nb* palabras de la llave. El esquema resultante consiste en un arreglo lineal de palabras de 4-bytes, denotado $[w_i]$, con i en el rango $0 \leq i < Nb(Nr+1)$.

La expansión de la llave genera al esquema de llaves y se realiza de acuerdo al pseudo código mostrado en la tabla 2.12.

SubWord() es una función que toma una palabra de entrada de cuatro-bytes y aplica la S-box a cada uno de los cuatro bytes para producir una palabra de salida. La función *RotWord()* toma la palabra $[a_0, a_1, a_2, a_3]$ como entrada, realiza una permutación cíclica y devuelve $[a_1,$

a_2, a_3, a_0]. *Rcon[i]*, contiene los valores [x^{i-1}, {*00*}, {*00*}, {*00*}], con x^{i-1} potencia de x (x es denotada como {*02*}) en el campo *GF(*2^8*)*.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk) begin
    word temp

    i = 0

    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while

    i = Nk

    while (i < Nb * (Nr+1)]
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Tabla 2.12: *Pseudo código de expanción de llave*

Del código de la tabla 2.12 se puede ver que en las primeras *Nk* palabras de la llave expandida, está contenida la llave de cifrado. Cada palabra, *w[i]*, es el XOR de la palabra anterior, *w[i-1]* y la *Nk* de la palabra anterior, *w[i-Nk]*. Esta transformación consiste en un corrimiento cíclico de los bytes de una palabra (*RotWord()*), seguido por la aplicación de la *lookup table* a los cuatro bytes de la palabra (*SubWord()*).

Es importante notar que la rutina de expansión de una llave de longitud de 256-bits (*Nk*=8) es ligeramente diferente que las de 128-bits y 192-bits. Si *Nk*=8 y $i-4$ es múltiplo de *Nk*, entonces en *SubWord()* se aplica a $w[i-1]$ antes del XOR.

2.3.10. Descifrado

Las transformaciones usadas al cifrar se pueden invertir y después ejecutar en orden inverso para producir el descifrado del algoritmo AES. Las transformaciones individuales usadas en el descifrado - *InvShiftRows()*, *InvSubBytes()*, *InvMixColumns()*, y *AddRoundKey()* - procesan el estado. Se describen mas a delante.

El descifrado se describe en el pseudo código de la tabla 2.13, el arreglo *w[]* contiene el esquema de la llave, que fue descrito anteriormente.

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]) begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    for round = Nr-1 step -1 downto 1
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)
    end for

    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[0, Nb-1])

    out = state
end
```

Tabla 2.13: *Pseudo código para el descifrado.*

2.3.11. Transformación *InvShiftRows()*

La transformación *InvShiftRows()* es la transformación inversa de *ShiftRows()*. Los bytes en las últimas tres filas del estado son corridos cíclicamente diferente número de bytes. La primera fila, r = 0, no es recorrida. Las tres filas inferiores son corridas por *Nb - shift(r,Nb)* bytes, donde *shift(r, Nb)* depende del número de la fila.

La transformación **InvShiftRows()**se realiza de la siguiente manera:

$$s'_{r,(c+shift(r,Nb))mod\ Nb} = s_{r,c} \text{ para } 0 < r < 4 \ y \ 0 \leq c < Nb$$

En la figura 2.8 se muestra el proceso de la operación *InvShiftRows()*.

2.3.12. Transformación *InvSubBytes()*

InvSubBytes() es la transformación inversa de la substitución de bytes, la cual usa una tabla inversa de la *S-box*, esta se aplica a cada byte del estado. Estos inversos son calculados tomando los inversos multiplicativos en $GF(2^8)$.

En la tabla 2.14 se muestra la tabla inversa de S-box, esta se utiliza en la transformación *InvSubBytes()*.

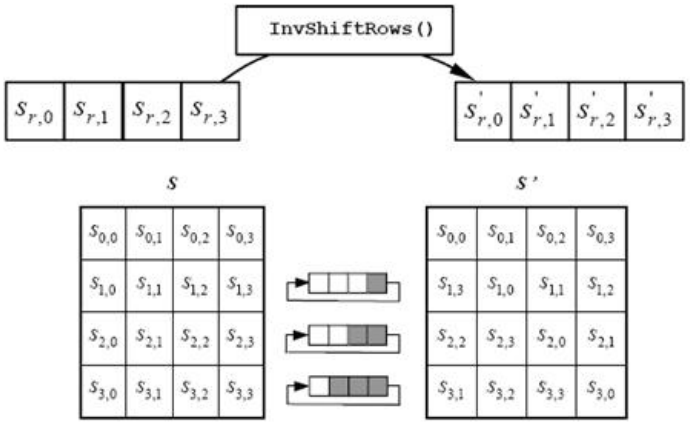

Figura 2.8: *Aplicación de la transformación InvShiftRows() al estado*

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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 2.14: *Tabla inversa de S-box esta tabla esta dada por el FIPS-197 [2]*

2.3.13. Transformación *InvMixColumns()*

InvMixColumns() es la transformación inversa de *MixColumns()*. *InvMixColumns()* opera el estado columna-por-columna, tratando cada columna como un polinomio. Las columnas son consideradas como polinomios sobre $GF(2^8)$ y multiplicadas modulo $x^4 + 1$ con el polinomio fijo $a^{-1}(x)$, donde

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

Esto puede ser escrito como producto de matrices. Sea

$s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix} \qquad \text{para } 0 \leq c < Nb$$

Como resultado de esta multiplicación, los cuatro bytes en una columna son representados como sigue:

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

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

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

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

2.3.14. Transformación inversa de *AddRoundKey()*

El inverso de *AddRoundKey()*, es ella misma, puesto que implica solamente el uso de la operación de XOR.

2.3.15. Descifrado equivalente

En el descifrado directo presentado en la sección 2.3.10, la secuencia de las transformaciones se diferencia respecto a la del cifrado, mientras que la forma del esquema de llaves permanece igual tanto para el cifrado y como para el descifrado. Sin embargo, varias características del algoritmo AES permiten un descifrado equivalente que tenga la misma secuencia de transformaciones que el cifrado (con las transformaciones substituidas por sus inversos). Esto se logra cambiando el esquema de llaves.

Las dos propiedades que permiten el descifrado equivalente son las siguientes:

1) Las transformaciones de *SubBytes()* y *ShiftRows()* conmutan; es decir, una transformación de *SubBytes()* seguida inmediatamente por una transformación *ShiftRows()* es equivalente a una aplicación de la transformación *ShiftRows()* seguida de la transformación *SubBytes()*. Esto también es verdad para sus inversos, *InvSubBytes()* e *InvShiftRows()*.

2) Las operaciones que mezclan columnas - *MixColumns()* e *InvMixColumns()* - son lineales con respecto a la entrada de la columna, esto significa

   InvMixColumns(state XOR Round Key) = InvMixColumns(state) XOR InvMixColumns(Round Key).

Estas propiedades permiten que el orden de las transformaciones *InvSubBytes()* y *InvShiftRows()* se puedan invertir. El orden de las transformaciones *AddRoundKey()* y *InvMixColumns()* también se pueden invertir, siempre y cuando las columnas (palabras) del esquema de la llave de cifrado se modifiquen usando la transformación *InvMixColumns()*.

El descifrado equivalente es definido modificando el orden de algunas transformaciones dentro del ciclo principal *for* del descifrado directo, se invierten las transformaciones *InvSubBytes()* y de *InvShiftRows()*, también se invierte el orden de las transformaciones *AddRoundKey()* y *InvMixColumns()* estos cambios son realizados sobre el pseudo código mostrado en la tabla 2.13. Hay que destacar que estoy cambios no afectan a las primeras *Nb* palabras del esquema de la llave.

Dado estos cambios, los resultados del descifrado equivalente algunas veces son más eficientes que el descifrado descrito en la sección 2.3.10 y la tabla 2.13. El pseudo código para el descifrado equivalente aparece en la tabla 2.15. (La palabra del arreglo **dw**[] contiene modificado el esquema de la llave. La modificación en la rutina de expansión de la llave también aparece en la tabla 2.15).

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

    for round = Nr-1 step -1 downto 1
        InvSubBytes(state)
        InvShiftRows(state)
        InvMixColumns(state)
        AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
    end for

    InvSubBytes(state)
    InvShiftRows(state)
    AddRoundKey(state, dw[0, Nb-1])

    out = state
end
```

Para el descifrado equivalente, el siguiente pseudo código es añandido al final de la rutina de expanción de llave(Sección 2.3.9):

```
    for i = 0 step 1 to (Nr+1)*Nb-1
        dw[i] = w[i]
    end for

    for round = 1 step 1 to Nr-1
        // note change of type
        InvMixColumns(dw[round*Nb, (round+1)*Nb-1])
    end for
```

Tabla 2.15: *Pseudo código para el descifrado equivalente.*

2.4. Método de cifrado del AES

Para otros algoritmos de cifrado por bloques como el DES, o el TDES existen recomendaciones FIPS Pub. 81 (ECB, CBC, CFB y OFB), para el AES también hay una recomendación de modos para el cifrado algunos de estos son: *Electronic Codebook* (ECB), *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB) y *Counter* (CTR), aparte de estas recomendaciones se tienen otro tipo de recomendaciones basadas en códigos de autentificación de mensajes (MAC) como CMAC[2] o CBC-MAC[3], pero estos modos trabajan con todo el mensaje que se tiene que mandar y generan un código que sirve para validar si corresponde al mensaje mandado, o no. Este procedimiento es muy extenso para los pocos recursos de una *smart card* y por lo tanto no será mencionado.

Los métodos de cifrado CBC, CFB y OFB requieren de un vector de inicialización al que se le denota como IV este se utiliza en el paso inicial del cifrado del mensaje. Es importante resaltar que el vector no debe ser secreto usualmente se coloca la misma llave como el vector IV.

Se denotará al algoritmo de cifrado (AES) como $CIPH_k$ y al descifrado como $CIPH_k^{-1}$, a los bloques de texto en claro como $P_1, P_2, ..., P_n$ y el texto cifrado como $C_1, C_2, ..., C_n$. La funcione MSB_i regresan los "i" bits más significativos, la función LSB_i regresa los i bits menos significativos y el símbolo pipe "|" representa la concatenación de bits.

Los modos ECB, CBC y CFB requieren que la longitud del texto a cifrar sea múltiplo del tamaño del bloque, así que para lograr esto se requiere añadir cierta cantidad de bits, o padding.

2.4.1. Modo ECB *Electronic Codebook*

Por las características de este modo de cifrado, es viable realizar implementaciones en paralelo, ya que se pueden cifrar los bloques de manera independiente como se muestra en la figura 2.9. Tiene la desventaja que al conseguir la suficiente cantidad de texto cifrado y su contraparte en texto claro se pueden realizar ataques de diccionario. A continuación se proporciona la definición de ECB:

ECB Cifrado:	$C_j = \text{CIPH}_k(P_j)$	para j = 1 ... n
ECB Descifrado:	$P_j = \text{CIPH}_k^{-1}(C_j)$	para j = 1 ... n

Al cifrar con ECB se aplica la función de cifrado de forma directa e independiente a cada bloque de texto en claro, para obtener el texto cifrado.

Al descifrar con ECB se aplica la función en descifrado en forma directa e independiente a cada bloque de texto cifrado, para obtener el texto en claro.

[2]Cipher-based Message Authentication Code

[3]Es el modo de cifrado CBC que incluye autentificación de mensaje

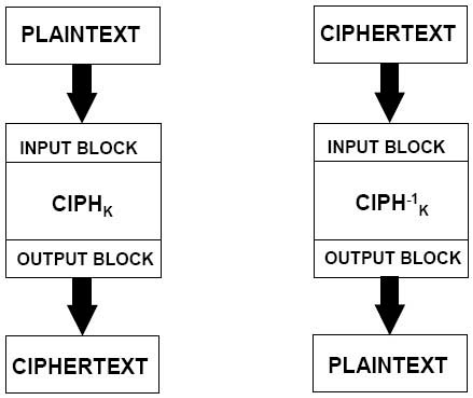

Figura 2.9: *Método ECB, de lado izquierdo el cifrado de lado derecho el descifrado*

2.4.2. Modo CBC *Cipher Block Chaining*

En este modo se encadena el texto cifrado de la ronda anterior, con el texto en claro del bloque actual, esto se realiza mediante el uso del operador XOR. Como ya se había mencionado, este modo requiere de un vector de inicialización IV, para combinarlo con el primer bloque de texto en claro, este método se define de la siguiente manera:

$$\begin{array}{lll} \text{CBC Cifrado:} & C_1 = \text{CIPH}_K(P_1 \bigoplus IV); & \\ & C_j = \text{CIPH}K_(P_j \bigoplus C_{j-1}) & \text{para j = 2 ... n} \\ \text{CBC Descifrado:} & P_1 = \text{CIPH}_K^{-1}(C_1) \bigoplus IV; & \\ & P_j = \text{CIPH}_K^{-1}(C_j) \bigoplus C_{j-1} & \text{para j = 2 ... n} \end{array}$$

Al cifrar utizano el modo CBC se realiza un XOR con el primer bloque de texto plano y con IV, este resultado se le aplica CIPH_K la salida de ese bloque se le hace un XOR con el siguiente bloque de texto en claro.

En el proceso de descifrado utilizando el modo CBC se aplica el CIPH_K^{-1} al primer bloque C_1 de bloque de entrada, al resultado se le aplica XOR con IV, al siguiente bloque C_2 se le aplica CIPH_K^{-1} y se realiza XOR con C_1.

Este proceso de cifrado y descifrado con este modo se muestra en la figura 2.10.

2.4.3. Modo CFB (Cipher Feedback)

Este modo encadena el texto cifrado de la ronda anterior con la ronda actual mediante un XOR que se hace con la salida el CIPH_K.

Este modo de operación requiere de un parámetro s, que debe estar entre 1 y b ($1 \leq s \leq b$), el texto es segmentado en ($P_j^{\#}$) de bits tamaño s y el texto cifrado es segmentado en ($C_j^{\#}$). Usualmente este valor s es incorporado en el nombre, por ejemplo: el modo 1-bit CFB, modo 8-bit CFB, el modo 64-bit CFB, o el modo 128-bit CFB.

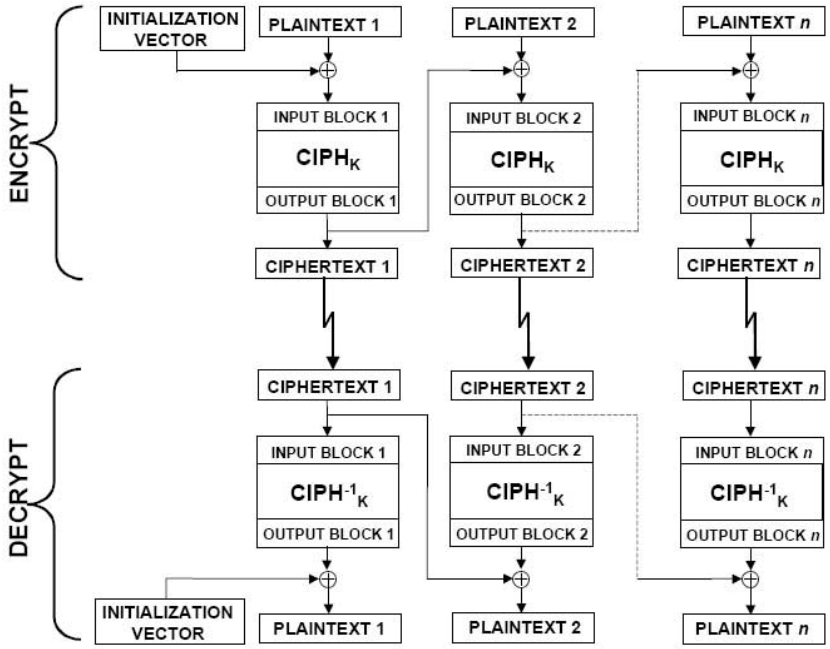

Figura 2.10: *Método CBC*

El modo CFB se define de la siguiente manera:

CFB Cifrado:	$I_1 = IV$;	
	$I_j = \text{LSB}_{b-s}(I_{(j-1)}) \text{ — } C^{\#}_{j-1}$	para j= 2...n
	$O_j = \text{CIPH}_K(I_1)$	para j = 1,2 ... n
	$C^{\#}_j = \text{P}^{\#}_j \bigoplus \text{MSB}_s(O_j)$	para j = 2 ... n
CFB Descifrado:	$I_1 = IV$;	
	$I_j = \text{LSB}_{b-s}(I_{(j-1)}) \text{ — } C^{\#}_{j-1}$	para j= 2...n
	$O_j = \text{CIPH}_K(I_1)$	para j = 1,2 ... n
	$P^{\#}_j = \text{C}^{\#}_j \bigoplus \text{MSB}_s(O_j)$	para j = 2 ... n

El vector IV se cifra con CIPH_K y a esta salida se le realiza un XOR con los s bits más significativos de la salida y el primer bloque $\text{P}^{\#}_1$, esto nos da el primer bloque cifrado $\text{C}^{\#}_1$. Los $s - b$ bist menos significativos de IV son concatenados con $\text{C}^{\#}_1$ para obtener el segundo bloque de entrada, este proceso se repite para todos los bloques de texto en claro —$\text{P}^{\#}_n$ bloques—.

El proceso de descifrado es muy parecido solo que en lugar de usar los bloques de texto en claro se utilizan los de texto cifrado y para los bloques de entrada al CIPH_K se utiliza el bloque anterior $C^{\#}_{j-1}$.

El proceso de cifrado y descifrado se muestra en la figura 2.11. Este modo es útil cuando se requiere mandar información que se esta generando en tiempo real y se requiere mandar la información de manera constante, y no se puede esperar a que se llene un *buffer* grade.

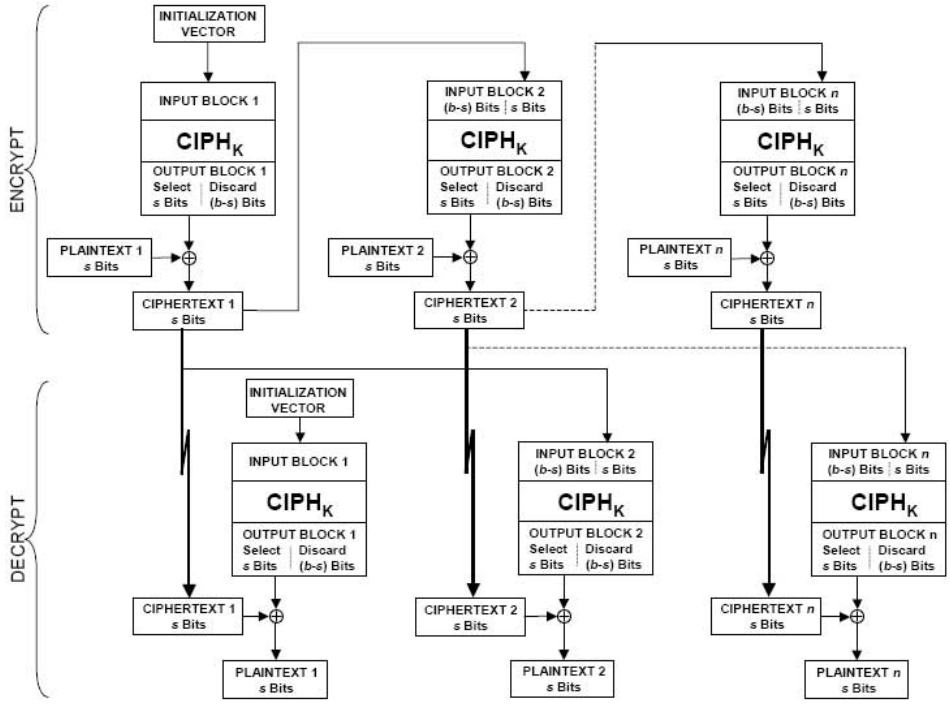

Figura 2.11: *Método CFB*

2.4.4. Modo OFB *Output Feedback*

Este modo encadena el texto cifrado de la ronda anterior con la ronda actual mediante un XOR que se hace con la salida del CIPH_K, a diferencia del CFB este modo toma el bloque completo y este se pasa a la siguiente ronda.

El modo OFB se define de la siguiente manera:

CFB Cifrado:	$I_1 = IV;$	
	$I_j = O_{j-1}$	para j = 2 ... n
	$O_j = \mathrm{CIPH}_K(I_j)$	para j = 1,2 ... n
	$C_j = P_j \bigoplus O_j$	para j = 1,2 ... n
	$C_n^* = \mathrm{P}_n^* \bigoplus \mathrm{MSB}_u(O_n)$	
CFB Descifrado:	$I_1 = IV;$	
	$I_j = O_{j-1}$	para j = 2 ... n
	$O_j = \mathrm{CIPH}_K(I_j)$	para j = 1,2 ... n
	$P_j = C_j \bigoplus O_j$	para j = 1,2 ... n
	$P_n^* = \mathrm{C}_n^* \bigoplus \mathrm{MSB}_u(O_n)$	

En este modo el vector IV es transformado y a la salida se le aplica un XOR con P_j, así se obtiene el primer bloque cifrado, este es la entrada para la siguiente ronda.

El proceso de cifrado y descifrado se muestra en la figura 2.12.

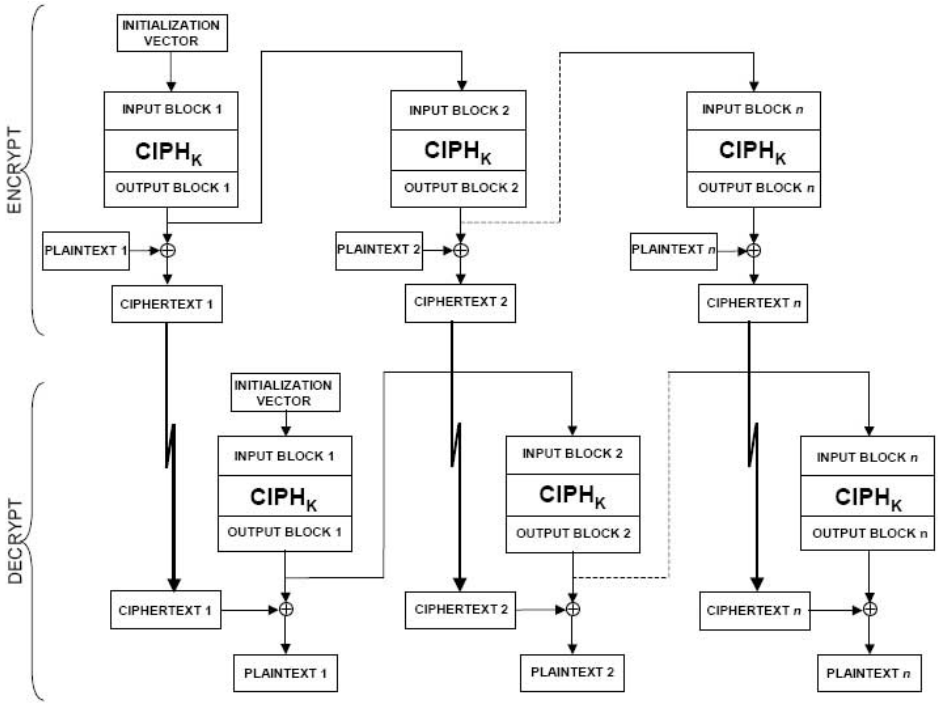

Figura 2.12: *Método OFB*

2.4.5. Modo CTR *Counter*

Se llama modo contador (Counter) ó CTR debido a que utiliza una secuencia de bloques de salida a los cuales se les aplica un XOR con los bloques de texto en claro. Estos bloques se generan aplicando al contador la función CIPH_K, se les nombrará como $T_1, T_2, ..., T_n$. El modo CRT se define de la siguiente manera:

CFB Cifrado:	$O_j = \mathrm{CIPH}_K(T_j)$	para j = 1,2 ... n
	$C_j = P_j \bigoplus O_j$	para j = 1,2 ... n-1
	$C_n^* = \mathrm{P}_n^* \bigoplus \mathrm{MSB}_u(O_n)$	
CFB Descifrado:	$O_j = \mathrm{CIPH}_K(T_j)$	para j = 1,2 ... n
	$P_j = \mathrm{C}_j \bigoplus O_j$	para j = 1,2 ... n-1
	$P_n^* = \mathrm{C}_n^* \bigoplus \mathrm{MSB}_u(O_n)$	

Para cifrar, se generan los contadores a estos se les aplica la transformación CIPH_K y finalmente se les aplica una XOR; para el último bloque que no necesariamente es múltiplo del tamaño del bloque, se toman los bits más significativos del bloque anterior y se realiza el XOR, los demás bits son descartados.

En el descifrado se calculan los contadores y se realiza el mismo procedimiento que al cifrar, solo que se toma como entrada los bloques C_i.

Al trabajar con este modo es viable realizar una implementación en paralelo, ya que se puede hacer de manera independiente al mensaje, la generación de los contadores y la aplicación de la función CIPH_K. Más aun la generación de los T_i se puede hacer a priori,

además la aplicación de XOR se realiza de forma independiente entre los bloques.

2.5. Tarjetas inteligentes o *smart cards*

Uno de los mecanismos más comunes de seguridad es cifrar la información para luego ser transmitida por un canal, esto asegura la confidencialidad, otro mecanismo es ingresar este tipo de algoritmos junto con la llave en un dispositivo portátil con el fin de autentificar al usuario mediante este dispositivo y almacenar algún tipo de información importante (códigos secretos, información financiera, etc.).

2.5.1. ¿Que es una *SMART CARD*?

La idea en las *smart cards* es insertar un chip en una tarjeta plástica, ya que el chip integra todas las cualidades de computo en una pastilla, evita que haya una ruta de penetración a los accesos ilegales.

La gran ventaja de las *smart cards* respecto de las tarjetas magnéticas es que permite ejecutar algoritmos criptográficos en su circuitería interna. Estas brindan máxima seguridad en todos los sistemas en las que participan. En este sentido son muy convenientes debido a su tamaño, aunque por su reducido tamaño también tienen limitaciones tanto en memoria y en capacidad de cómputo, en los últimos años esta capacidad se ha incrementado bastante pero sigue siendo una limitante en la capacidad criptográfica de estas.

La mayor demanda en las *smart cards* de clave pública se da en las compañías operadoras de teléfono, bancos y corporaciones de seguro.

El soporte físico de una *smart card* convencional es un rectángulo de plástico impreso con información concerniente a la aplicación.

De acuerdo con "*International Organization for Standardization (ISO) Standard 7816*", consiste en cuatro partes:

ISO7816-1: Define las características físicas de la tarjeta.

ISO7816-2: Define las dimensiones y la posición de los contactos.

ISO7816-3: Define los protocolos de transmisión.

ISO7816-4: Es el estándar de los comandos.

El CPU de las tarjetas son microcontroladores de 8 bits, los más comunes son Motorola 68HC05, Intel 80C51. El microprocesador de la tarjeta ejecuta un programa escrito en una memoria *ROM* el cual no puede ser modificado de ninguna manera. Esto garantiza que el fabricante controla el código en forma estricta.

Para almacenar datos individuales de cada usuario especifico, la primera generación de *smart cards* de memorias no volátiles usaban *EPROM* (memorias programables

eléctricamente). Estas requerían una fuente extra de "alto voltaje"típicamente de 15 a 20V. En cambio las mas actuales contienen *EEPROM* (*electrically erasable programmable memory*), las cuales pueden ser escritas y borradas cientos de veces. También es posible a veces importar programas dentro de la *EEPROM* de la tarjeta de acuerdo a las necesidades.

El tamaño de la *EEPROM* es un punto crítico a la hora del diseño de aplicaciones de clave pública, donde las claves son relativamente largas. Consecuentemente los programadores de las *smart cards* frecuentemente adoptan técnicas típicas de optimización como por ejemplo reconstruir la clave pública a partir de una contraseña, reconstruir una clave secreta a partir de una semilla más corta (números secretos más cortos), evitando esquemas de claves largas, implementar algoritmos de compresión para los datos redundantes. Diversos fabricantes han desarrollado sistemas operativos completos para este propósito.

Finalmente la tarjeta contiene un puerto de comunicación serial (trabajando en forma asincrónica) para intercambiar datos e información de control entre la tarjeta y el mundo externo. Una tasa de transferencia de bits común es de 9600 bits por segundo, pero hay interfaces mucho más veloces que son comúnmente usadas (desde 19200 hasta 115200 bits por segundo) estando totalmente de acuerdo con la norma ISO 7816-3 [9].

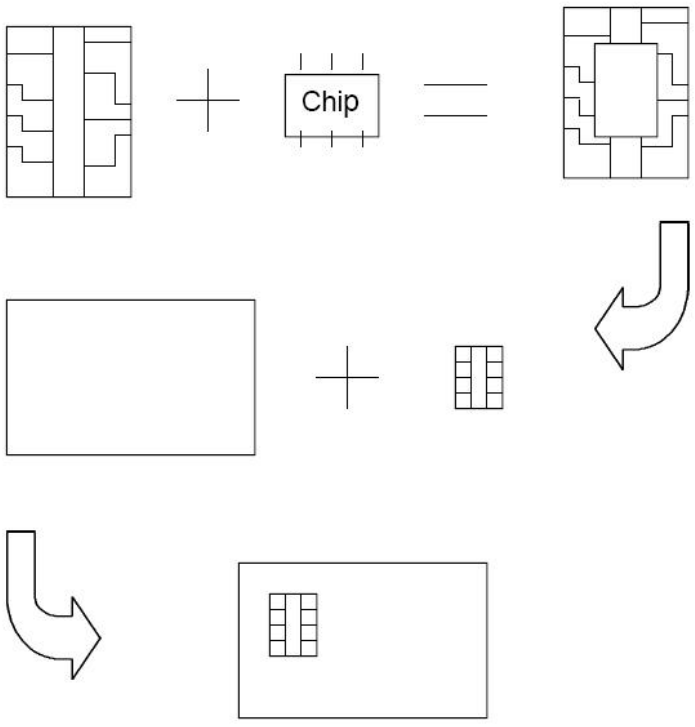

Figura 2.13: *Ensamblado de una* Smart Card

En la figura 2.13 se observa el procedimiento de fabricación de las *smart cards*. Los pasos son:

1) Armado del chip + micromodulo,

2) chip + micromódulo + tarjeta de plastico.

Si esto no es hecho (juntar todo en un solo chip) podría existir una posible ruta de penetración para un acceso ilegal a la tarjeta. El standard especifica las características de la tarjeta para resistir una serie de fatigas mecánicas. El tamaño del chip está consecuentemente limitado, y presenta restricciones, especialmente en la memoria y en la capacidad criptográfica.

Los chips de las *smarts cards* son muy confiables y la mayoría de los fabricantes garantizan las propiedades eléctricas de sus chips por 10 años, o más. Los standards especifican como debe ser protegida una tarjeta contra las agresiones mecánicas, eléctricas o químicas.

Los sensores de seguridad permiten al microcontrolador prevenir intentos de monitoreo por reseteo de la *RAM/EEPROM*. Detectores de reloj reaccionan ante una muy elevada, o una muy baja frecuencia de reloj. Por obvias razones de seguridad, la información acerca de los detectores de seguridad, habitualmente es bastante difícil de obtener de los fabricantes. En general, las *smarts cards* pueden ayudar en situaciones que requieran objetos portátiles seguros. La calidad de la tarjeta con respecto a la falsificación de la misma, combinado con la criptografía de clave pública, generalmente provee una solución adecuada para muchos de los problemas de seguridad existentes hoy en día.

La primera regla de seguridad es juntar todos los elementos en un solo chip como se muestra en la figura 2.13, el circuito internamente se muestra en la figura 2.14, donde se aprecian la *RAM*, la parte de entrada y salida *I/O*, el *CPU*, la *ROM* y la *EPROM*.

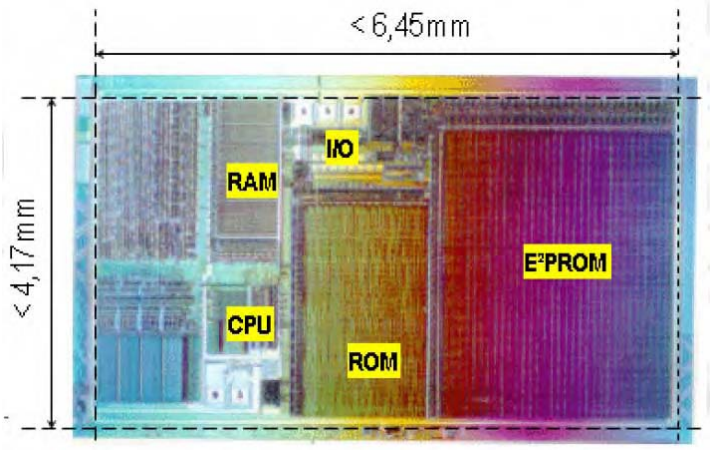

Figura 2.14: *Disposición interna de los componentes de la* Smart Card

2.5.2. ISO7816-1

Define las características físicas, algunas de estas son:

Luz ultra violeta Deben tener protección contra un ambiente con fuerte radiación de luz UV.

Rayos X La exposición de cualquier parte de la tarjeta que sea de 0.1 G y energía-media de 70 a 140 Kv (acumulativa en un año) no debe causar fallos en la tarjeta.

Superficie de contacto La diferencia en el nivel de la superficie de contacto no debe exceder 0.1mm.

Resistencia mecánica La tarjeta debe resistir daños en sus superficie causados por su uso normal.

Resistencia eléctrica La resistencia entre los conectores no debe exceder los 0.5 Ohm, con una corriente entre los 50uA a 300mA.

2.5.3. ISO7816-2

El tamaño mínimo de contacto debe ser de 1.7 mm por 2 mm, el micro módulo contiene una cadena de 8 contactos, la configuración de conexión se ilustra en la figura 2.15, donde:

Vdd Corriente de entrada 5 V.

RST Reset

CLK Reloj

RFU Para uso interno de la tarjeta

GND Tierra

Vpp Pin para programación

I/O Entrada y salida en comunicación serial

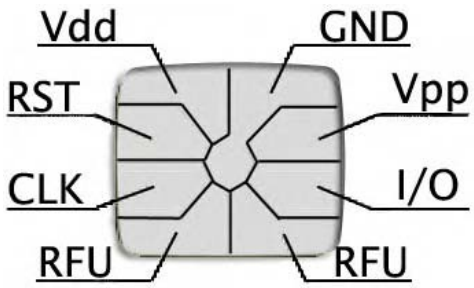

Figura 2.15: *Conexión de la SmartCard*

2.5.4. ISO7816-3

Las comunicaciones con las *smarts cards* están de acuerdo al standard ISO/IEC 7816-3 [9]. Este definen los siguientes tipos de comunicaciones:

- T=0 es el protocolo de transmisión de asíncrono *half duplex* orientado a byte.
- T=1 es el protocolo de transmisión de asíncrono *half duplex* orientado al bloque.

- T=2 y T=3 son reservados para futuras operación en *full duplex*.
- T=4 es reservado para mejoras en el protocolo de transmisión asíncrono *half duplex*.
- T=5 al T=13 son reservados para uso futuro.
- T=14 es reservado para el protocol del estándar ISO.
- T=15 son reservados para futuras extensiones.

2.5.5. Comunicaciones T=0

La interfaz del controlador siempre se inicia utilizando el protocolo T=0. La interacción con esta interfaz y el contacto de la tarjeta se tiene que realizar mediante una sucesión de instrucciones y sus respuestas. Para este protocolo, los datos solo fluyen en una dirección. En otras palabras, cualquiera de las instrucciones de mensajes contiene el dato para la interfaz, o la instrucción de respuesta. La dirección del flujo de los datos es implícita en la definición de la instrucción y de ahí ambos controladores de la interfaz y la placa necesitan tener necesariamente conocimiento a-priori.

Cuando se requiere transferir datos en ambas direcciones para una instrucción en particular entonces se toma una instrucción de respuesta después de la instrucción primaria para recuperar de la respuesta.

Las instrucciones de mensaje consisten de 5 caracteres de encabezado que la interfaz del controlador manda a la placa de contacto. Esta contesta produciendo un byte, después los datos son enviados a la placa, o desde la placa, depende de la instrucción en particular. Este byte del procedimiento permite que el controlador del dispositivo controle el voltaje de programación de V_{pp} *EPROM*. En el caso de la memoria *EEPROM* este byte del *procedimiento* es redundante. El encabezado de la instrucción consiste en los 5 bytes siguientes.

- CLA - la clase de instrucción (el FF es reservado para el PTS).
- Ins - el código de la instrucción (e.g read memory).
- P1 - argumento de la instrucción (e.g dirección de memoria).
- P2 - argumento adicional del código de la INS.
- P3 - la longitud del bloque de datos

Cuando P3 es igual a cero los datos de la tarjeta serán 256 bytes.

La condición normal para el byte del procedimiento del ACK, repite el byte de la instrucción (INS). Otras opciones permiten que los dispositivos de la interfaz controlen el voltaje de programación de V_{pp} según lo requerido. La tarjeta puede enviar opcionalmente un byte NULO del procedimiento (60 hex) para dar mayor tiempo al proceso de la instrucción. En esta situación el controlador de la interfaz debe esperar otro byte del procedimiento. El estándar de la ISO también permite que la tarjeta envié el primer byte de estado (SW1)

como el byte del procedimiento.

El protocolo T=0 también incluye un mecanismo de detección y corrección de errores. En el cual solo localiza los errores y retransmite el carácter corrupto.

2.5.6. Comunicaciones T=1

La comunicación T=1 es un protocolo de transmisión de bloques asíncrono *half duplex*. En los términos del modelo de OSI este protocolo funciona en la capa 2, la capa de trasmisión de datos. La capa física (la capa 1) funciona de la misma manera que para el protocolo T=0 a excepción de la detección y corrección de errores. Esencialmente este protocolo pone un bloque de caracteres que permite el,

- control de flujo
- cadenas de bloques
- corrección de errores

La ventaja más obvia del protocolo T=1 es la capacidad de manejar flujo de datos en ambas direcciones, también quita restricciones sobre T=0 de la relación auxiliar principal, donde el dispositivo de la interfaz (IFD) inicia siempre una instrucción a la cual la placa responde. Para este protocolo de bloque, una instrucción se puede iniciar por el IFD, o la placa.

Otra ventaja del protocolo T=1 es la capacidad de encadenar los bloques de datos de tal forma que bloques arbitrariamente grandes de datos se puedan transferir como resultado de una sola instrucción.

Este protocolo también tiene un sistema más sofisticado de control de errores. Esto permite el uso de un código de detección de error del bloque (EDC) y de la capacidad de retransmitir los bloques que están conforme a una cierta condición de error. En comparación del protocolo T=0 tiene un esquema primitivo de la detección y de corrección de error del carácter.

En el protocolo general T=1 ofrece las ventajas donde el uso está manejando grandes bloques de datos, particularmente cuando se requiere pasar datos en ambas direcciones como parte de una instrucción en particular. La eficacia del protocolo es realmente evidente para transmisiones de datos más grandes, puesto que la capa física subyacente todavía está funcionando en modo de caracter en cuanto al protocolo T=0.

No puede haber duda que el control de errores está mejorado perceptiblemente sobre el protocolo T=0 pero a una velocidad más baja de 9600 bit/second la probabilidad de los errores de la comunicación se reduce mucho. Sin embargo está claro que hay una tendencia hacia el uso del protocolo T=1 y parece muy probable que éste se convierta en el protocolo predominante del futuro. Este protocolo es especificado en el ISO 7816 - 3/AMD.

El marco del bloque

El marco consiste en tres campos.

- campo del prólogo
- campo de información (opcional)
- campo del epílogo

Prologue Field			Information Field	Epilogue Field
Node Address NAD	Protocol Control Byte PCB	Length LEN	Optional INF	Error Detection LRC or CRC EDC
1 Byte	1 Byte	1 Byte	0-254 Bytes	1/2 Bytes

El campo del prólogo consiste en tres bytes.

- NAD dirección del nodo.
- PCB byte de control del protocolo
- LEN longitud de los datos

El byte de NAD utiliza los bits 3-1 para identificar la dirección, los bits 7-5 de la fuente para identificar la dirección de destino. Los bits 4 y 8 se utilizan para el control de V_{pp}. El byte de la dirección del nodo permite el uso de múltiples canales lógicos, donde se requiera que ambas direcciones se deban fijar, de otra manera valdrá cero.

El byte del PWB permite la identificación de tres tipos de trama de bloque.

- Un bloque con la información (I - bloque).
- Un bloque *receive ready* (R - bloque).
- Un bloque de supervisión (S - bloque)

El bloque de información es la trama que es usada para transmitir las instrucciones y los datos entre la placa y el IFD. El bloque *receive ready* se utiliza como reconocimiento cuando el protocolo está enviando datos como secuencia de bloques encadenados. El bloque supervisor se utiliza para establecer parámetros del control y para efectuar una resincronización, o para abortar como resultado una cierta condición de error. El bloque de información también actúa como byte del reconocimiento en el modo de no encadenamiento.

El byte de LEN indica el número de bytes en el campo de información de la trama. Su gama permitida de valores es a partir del 00 - FF hexadecimal. Esto permite un campo de información máximo de 254 bytes.

El campo del epílogo contiene el código de la detección de error del bloque que puede ser un LRC (control por redundancia longitudinal), o un CRC (control por redundancia cíclico). El LRC es 1 byte mientras que el CRC ocupa 2 bytes. Esta opción es definida por los caracteres específicos de la interfaz.

2.5.7. Estructura de archivo.

Hay tres categorías de archivos.

- Archivo maestro (MF).
- Archivo dedicado (DF).
- El archivo elemental (EF)

El archivo maestro es un archivo obligatorio en el estándar y representa la raíz de la estructura del archivo. Contiene la información de control del archivo y la memoria destinada. Dependiendo de la implementación puede tener archivos dedicados y/o archivos elementales como hijos, ver figura 2.16.

Un archivo dedicado tiene características similares al archivo maestro y puede también tener otros archivos dedicados y/o archivos elementales como hijos.

Un archivo elemental son las hojas del árbol de jerarquía del MF, contiene datos así como información de control del archivo. Los diferentes tipos de archivo elementales son los siguientes.

- Archivo de trabajo.
- Archivo público.
- Archivo de control de aplicación.
- Archivos secretos.

Los archivos de trabajo están para almacenar datos del uso mientras que los archivos públicos permiten que los datos sean accedidos incondicionalmente. Los archivos de control de la aplicación permiten el acceso de lectura mientras que los archivos secretos no pueden ser accedidos fuera de la tarjeta.

Cada archivo es referido por un identificador de dos bytes que dan la ruta ó *path* de cualquier archivo desde la raíz. Este concepto de *path* es el mismo principio al utilizado en PCs por MSDOS. Los archivos dedicados se pueden también referir por el nombre del archivo.

La estructura de datos para un archivo elemental permite cuatro opciones que se ilustran en la figura 2.17.

- Lineales de tamaño fijo.
- Lineales de tamaño variable.
- Cíclicos
- Transparentes.

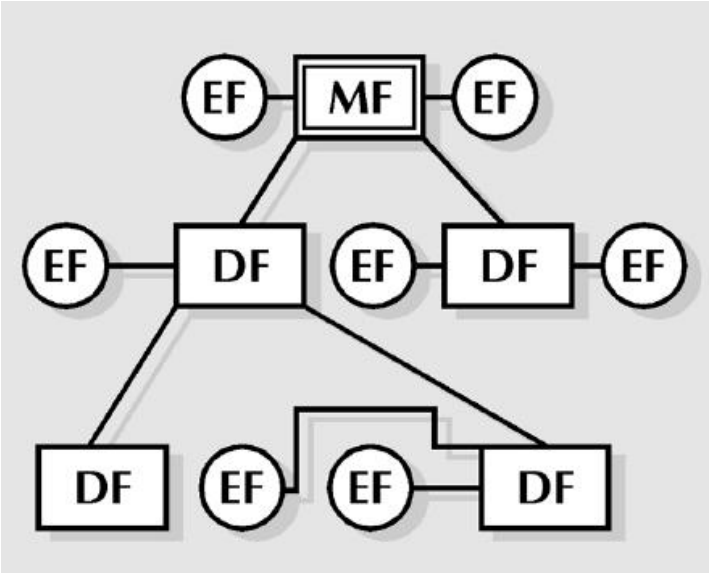

Figura 2.16: *Esquema del sistema de archivos bajo la norma ISO 7816-4*

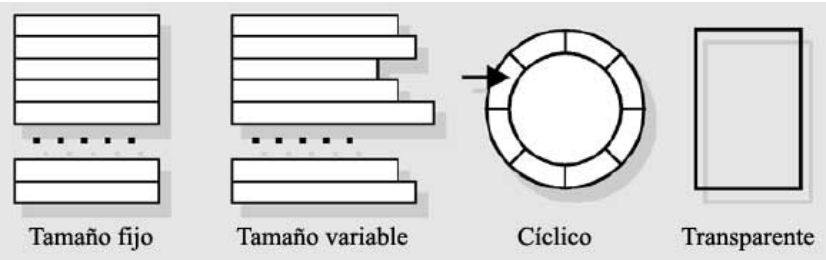

Figura 2.17: *Se muestran las cuatro estructuras de archivo.*

Capítulo 3

Proceso del implantación del AES

En la primera parte del presente capítulo se describen las herramientas utilizadas, los problemas encontrados al desarrollar este trabajo, el modelo de uso y observaciones que se encontraron al instalarlas y trabajar con ellas. Posteriormente se explican las optimizaciones que se aplicaron en la versión final del AES en la *smart card*.

3.1. Descripción de las herramientas

Se describen las herramientas utilizadas en este trabajo debido a que surgieron algunos imprevistos que consumieron bastante tiempo y que al dar un seguimiento a este trabajo se deben considerar.

Para compilar, modificar y trabajar con el *sosse* se tuvo que utilizar el SO Linux Red Hat 7.3, debido a que esta versión incorpora el compilador y las librerías soportadas para compilar este COS, las versiones superiores al gcc versión 3 no son compatibles debido a que la versión del *sosse* utiliza unas funciones que fueron suprimidas en avr-lib 1.2, o superiores. Por lo tanto, se utilizó la versión 1.0, que requiere el compilador gcc 3.3.x

Se listan los paquetes que se requirieron instalar para la compilación de sosse, de acuerdo a lo documentado en el trabajo [5].

- binutils-2.0.4.tar.gz
- gcc-3.0.4.tar.bz2
- avr-lib.
- sosse-20030413.tar.gz
- simulavr

Por comodidad para programar el algoritmo de cifrado AES se trabajó con el programa AVR Studio 4.0 que está hecho para trabajar bajo el SO Windows.

Para instalar el AVR Studio 4.0 no se tuvo que hacer otra cosa mas que seguir el asistente de instalación del *software*. En la figura 3.1 se muestra una pantalla del AVR Studio en la

parte izquierda se muestra la información sobre la ejecución, en la parte central se muestran los registros del código en ejecución y en la parte derecha la memoria *RAM* utilizada de la *smart card*.

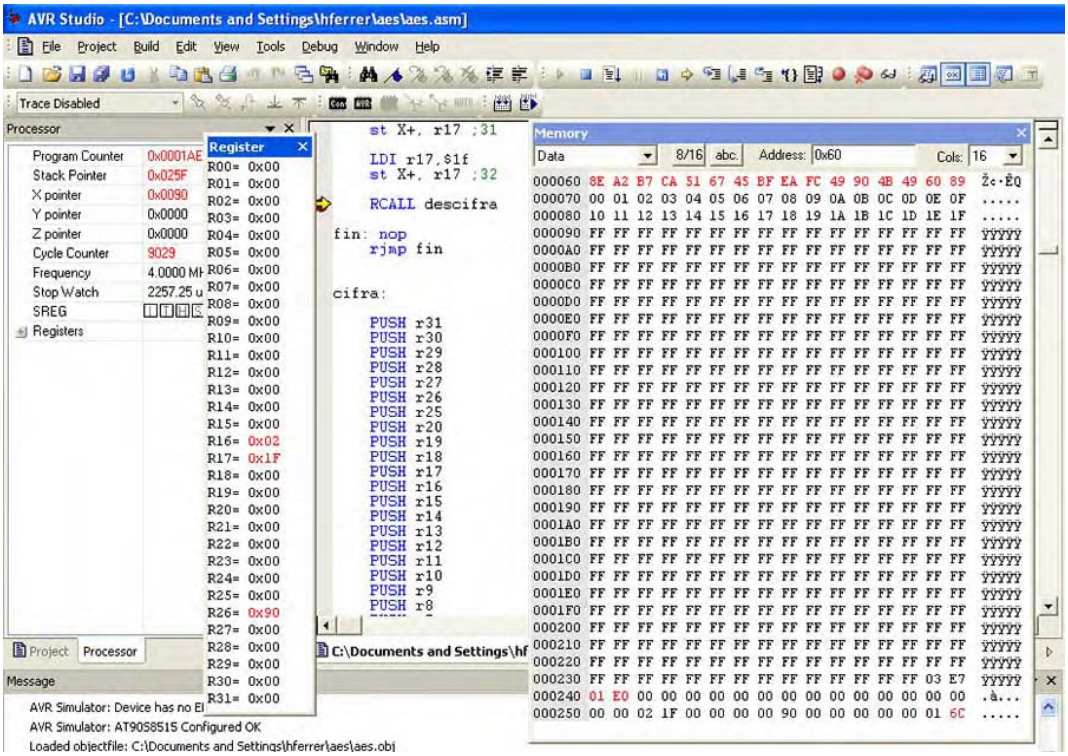

Figura 3.1: *Pantalla al debugear el AES utilizando ARV Studio*

3.2. Modelo de uso

Como el *sosse*, solamente soporta el algoritmo criptográfico TEA, nos enfocamos en mejorar este aspecto débil de *sosse* agregándole el AES a las tarjetas inteligentes, así que se tuvo que adaptar el sistema operativo para que pudiese utilizar nuestra implantación.

Los algoritmos de cifrado de bloques son utilizados por el COS con tres propósitos fundamentales:

- **Autentificación externa**. La tarjeta comprueba que el *host*[1] ha cifrado el reto adecuadamente (demostrando que tiene el secreto).
- **Autentificación interna**. El reto del *host* es cifrado por la tarjeta para demostrar su identidad con la clave secreta destinada para este fin.

[1]El término equipo anfitrión a aquel dispositivo que ofrece servicios

- **Generación de números pseudoaleatorios**. Se mantiene un estado el cual es alterado mediante el cifrador de bloques. Estos números son utilizados para generar el reto de la autenticación externa.

Se decidió extender el uso del AES en el *sosse* al modelo de autentificación externa, también llamado autentificación de reto respuesta, ya que el *host* genera un reto (una cadena aleatoria de bits), que son mandas a la *smart card* junto con la solicitud de autentificación externa, la *smart card* aplica el algoritmo de cifrado AES a la cadena, manda la respuesta al *host*, este aplica el AES a la cadena original y lo compara con la cadena recibida, cabe descartar que *host* conoce la llave de la *smart card* a autentificar, si las cadenas son iguales se acepta la autentificación, de lo contrario se rechaza. (Ver figura 3.2).

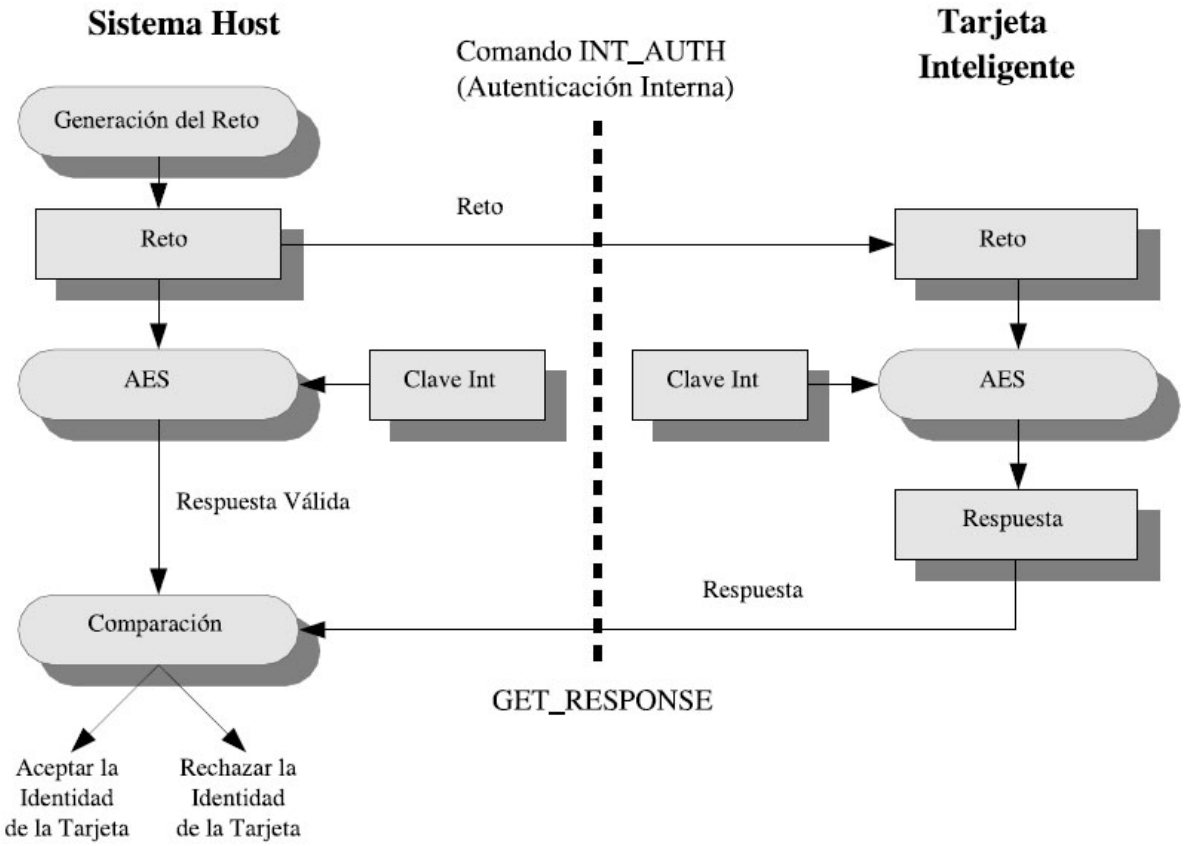

Figura 3.2: *Modelo de autentificación externa*

3.3. Puntos a considerar

Un problema en la implantación del AES mencionado en los trabajos de [5] y [6], es el tamaño de la implementación del algoritmo AES, esta implementación ocupa la mayor parte de la memoria disponible en la *smart card*, con el *sosse* sucede lo mismo. Lo que se recomienda en [5] es recortar el *sosse* y reducir el AES. En esas implantaciones se tuvo la ventaja de que se utilizó una longitud de llave de 128 bits, esto implicó que se pudiera reducir el código de AES aun más. Como en el presente trabajo se implanta una versión del AES

que maneja llaves de 256 bits el código de programa se incrementa, así que se tuvieron que buscar alternativas para reducir el incremento sin afectar el rendimiento de la implantación.

En el primer acercamiento al AES que se hizo, fue mediante una implementación en C, en esta se trabajo las transformaciones *MixColumns()* y *InvMixColumns()* utilizando una *look up tables* para acelerar el proceso. Los resultados obtenidos son muy buenos ya que se trabajo en una PC, trabajar con *look up tables* reduce el tiempo de ejecución pero tiene un costo considerable en el tamaño de la implementación en forma considerable, lo cual es inaceptable para los pocos recursos de una *smart card*. Así que en las versiones en ensamblador se evito trabajar con las *look up tables*.

3.4. Versiones de implementación

A continuación se muestran algunas mejoras que se utilizaron al trabajar con un procesador de 8-bits.

3.4.1. Requerimientos de longitud de la llave

Una implementación del algoritmo de AES utiliza al menos una de las tres longitudes del llave especificadas en la sección 2.3.3 : 128, 192 o 256 bits (es decir, **Nk** = 4, 6, o 8, respectivamente).

3.4.2. *Keying Restrictions*

No se ha encontrado vulnerabilidades en las llaves o semi-vulnerabilidades para el algoritmo de AES y no hay restricción en la selección del tamaño de llave.

3.4.3. Parametrización de la longitud de la llave, tamaño del bloque y número de rondas

Este estándar define explícitamente los valores permitidos para la longitud de llave (Nk), el tamaño de llave (Nk) y el número de rondas (Nr) ver la tabla 2.9. Sin embargo, las versiones futuras de este estándar podían incluir cambios o adiciones a los valores permitidos para esos parámetros. Por lo tanto, los programadores pueden elegir el diseñar sus implementaciones del AES con la flexibilidad de futuras versiones.

3.4.4. Implementaciones sugeridas para diferentes plataformas

Las variaciones de las implementaciones son posibles, en muchos casos pueden ofrecer un mejor funcionamiento u otras ventajas. Tomando la misma llave y datos de entrada (texto plano o texto cifrado), cualquier implementación que produzca la misma salida (texto cifrado o texto plano) que se especificó en este estándar, es una implementación aceptable del AES. Ver bibliografía [14].

3.4.5. Implementación en 8-bits

El desempeño en procesadores de 8-bits es importante cuando se usan las *smart cards* ya que justamente su arquitectura es de 8-bits y además tienen que correr aplicaciones de cifrado y descifrado.

3.4.6. Multiplicación en campos finitos

En el algoritmo de Rinjdael no hay multiplicaciones de dos variables en *GF(2)* ya que las únicas multiplicaciones son entre constantes. De esta forma es más fácil de implementar.

Una multiplicación por el valor 02 puede ser implementada ya que el polinomio asociado a 02 es x. De esta manera, si se multiplica un elemento b por 02 tenemos:

$b \cdot x = b_7 \cdot x^8 + b_6 \cdot x^7 + b_5 \cdot x^6 + b_4 \cdot x^5 + b_3x^4 + b_2 \cdot x^3 + b_1 \cdot x^2 + b_0 \cdot x \pmod{m(x)}$

$= b_6 \cdot x^7 + b_5 \cdot x^6 + b_4 \cdot x^5 + (b_3 \oplus b_7)x^4 + (b_2 \oplus b_7)x^3 + b_1 \cdot x^2 + (b_0 \oplus b_7) \cdot x + b_7$

donde $m(x) = x^8 + x^4 + x^3 + x + 1$

La multiplicación por 02 la denotamos por *xtime(x)*. Esta puede ser implementada con una operación de corrimiento y una condicional XOR. Con el fin de evitar ataques con respecto al tiempo de respuesta esta operación debe tomar el mismo número de ciclos de reloj sin importar el valor. Una solución para esto es implementar una tabla M que contenga M[a]=02$\cdot a$.

Todos los elementos de $GF(2)$ pueden ser escritos como suma de potencias de 02, las multiplicaciones por cualquier constante puede ser implementada usando *xtime()*.

Un ejemplo de la multiplicación de una entrada b por la constante 15 puede ser implementada de la siguiente manera:

$b \cdot 15 = b \cdot (01 \oplus 04 \oplus 10)$

$= b \cdot (01 \oplus 02^2 \oplus 02^4)$

$= b \oplus xtime(xtime(b)) \oplus xtime(xtime(xtime(xtime(b))))$

$= b \oplus xtime(xtime(b \oplus xtime(xtime(b))))$

3.4.7. Cifrado en 8-bits

En un procesador de 8-bits el cifrar utilizando el algoritmo de cifrado Rijndael puede ser programado por una secuencia de sencillas transformaciones. La implementación de

ShiftRows() y *AddRoundKey()* es fácil siguiendo las especificaciones antes vistas. La implementación de *SubBytes()* requiere una tabla de 256 bites almacenada en memoria.

La eficiencia de las transformaciones *AddRoundKey()*, *SubBytes* y *ShiftRows()* se pueden mejorar si se combinan y ejecutan de manera serial. Los costos indirectos de la indexación son reducidos al mínimo al codificar las operaciones sobre el estado.

Al elegir él polinomio de *MixColumns()*, se consideró la eficacia en un procesador de 8-bits. Se ilustra en la tabla 3.1, donde la transformación *Mixcolumns()* se pueden realizar con una pequeña serie de instrucciones, (en la tabla se da el algoritmo.) La única multiplicación usada sobre el campo finito es con el elemento 02, denotada por *xtime*.

$t = a[0] \bigoplus a[1] \bigoplus a[2] \bigoplus a[3];$
$u = a[0];$
$v = a[0] \bigoplus a[1];\ v = xtime(v);\ a[0] = a[0] \bigoplus v \bigoplus t;$
$v = a[1] \bigoplus a[2];\ v = xtime(v);\ a[1] = a[1] \bigoplus v \bigoplus t;$
$v = a[2] \bigoplus a[3];\ v = xtime(v);\ a[2] = a[2] \bigoplus v \bigoplus t;$
$v = a[3] \bigoplus u;\ v = xtime(v);\ a[3] = a[3] \bigoplus v \bigoplus t;$

Tabla 3.1: *Implementación eficiente de MixColumns()*

Expansión de la llave. La implementación de la expansión de la llave de la forma más sencilla, se logra ocupando gran cantidad de *RAM* de la *smart card*. Mas aun, en muchas aplicaciones como tarjetas de débito, monederos electrónicos, el tamaño de datos a cifrar ó descifrar, típicamente utiliza pocos bloques por sesión. De allí que no se tenga mucha ganancia en el desempeño aunque se debe contemplar cualquier tipo de aplicación.

En el diseño del esquema de la llave, se tomaron en cuenta las restricciones impuestas por la *smart card*. La expansión de la llave puede ser implementada usando un *buffer* cíclico de $4N_k$ bytes. Cuando todos los bytes en el *buffer* se han utilizado, el contenido del *buffer* es actualizado. Todas las operaciones que actualizan la llave pueden implementarse eficientemente con operaciones a nivel de bytes.

La expansión convierte la llave original de $4N_k$ bytes al esquema de la llave de $4N_b(N_r + 1) = 240$ bytes, como ya se mencionó esto es demasiado para una *smart card*, pero como se puede calcular una clave de ronda a partir de una anterior se puede hacer esto con menor memoria, de hecho se requiere como mínimo $4N_k$ bytes, que es la longitud de una sola clave de ronda.
Una implementación gráfica de esta función se muestra en la figura 3.3.

Las constantes RC[i] pertenecen a $GF(2^8)$. Lo valores de RC están dados en la sección 2.3.9, la función *SubWord* es la utilizada tanto en *SubByte*, como en *KeyExpansion*.

El esquema es derivado del código mostrado en la tabla 3.1, ya que el primer *while* copia el contenido de la llave en el esquema de la llave y en esta implementación este espacio se reutiliza, el segundo *while* contiene el cálculo de las N_k palabras siguientes, esta parte es la descrita en la figura 3.3.

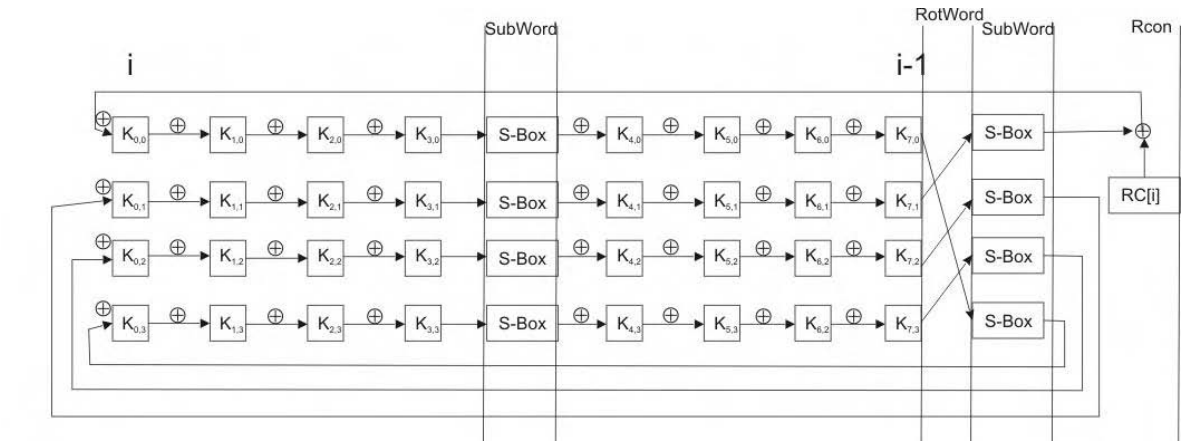

Figura 3.3: *Transformación de KeyExpansion para 8-bit en el momento i*

Cabe destacar que el código de cifrado se ve afectado considerablemente ya que se debe incluir el cálculo del esquema de la llave como parte del código de cifrado. Como la transformación *KeyExpansion* se debe realizar una vez después de dos aplicaciones de *AddRoundKey* esto implica mayores cambios en el código de cifrado, esto nos da como resultado el código para cifrado que se muestra en el anexo C.

```
Cipher(state, word w[Nk]) begin
    AddRoundKey
    for round = 1 step 2 to Nr-2
        SubBytes
        ShiftRows
        MixColumns
        AddRoundKey
        KeyExpansion
        SubBytes
        ShiftRows
        MixColumns
        AddRoundKey
    end for
    SubBytes
    ShiftRows
    MixColumns
    AddRoundKey
    KeyExpansion
    SubBytes
    ShiftRows
    AddRoundKey
end
```

Tabla 3.2: *Pseudo código para el cifrado del AES en un procesador de 8-bits*

3.4.8. Descifrado en 8-bits

Para implementaciones en plataformas de 8-bits, no es bueno seguir la descripción equivalente del algoritmo descrito. Sin embargo, la estrategia se muestra a continuación.

Con respecto a la transformación *InvMixColumns()* se tienen varias sugerencias. Como sabemos podemos ver $c(X)$ como la siguiente matriz:

$$c(X) = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

También sabemos que $d(X)$ es:

$$d(X) = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

Así que se deduce que

$$c(X) \cdot d(X) = 1$$

Al multiplicar $d(X)$ por ambos lados obtenemos

$$c(X) \cdot d^2(X) = d(X)$$

De aquí concluimos dos posibles ventajas, una es que como

$$d(X) = c(X) \cdot f(X)$$

Al evaluar se tiene que

$$d^2(X) = 04X^2 + 05$$

Pero también se tiene que

$$c^2(X) = 04X^2 + 05$$

Así que sustituyendo tenemos

$$d^2(X) = 04X^2 + 05 = c^2(X) => d(X) = c(X) \cdot c^2(X) = c^3(X)$$

Por lo tanto tenemos que

$$d(X) = c^3(X)$$

Lo que nos dice que *InvMixColumns()* puede ser implementada mediante tres llamadas de *MixColumns()* reduciendo a lo mínimo la complejidad y el tamaño de esta transformación.

Otra observación que se puede hacer es implementar *InvMixColumns()* realizando *InvMixColumns()* y multiplicando la matriz $d^2(X)$. De esta manera descifrar queda similar en estructura a cifrar, ya que llama a *MixColumns* y repite este proceso, pero con los coeficientes 05 00 04 00, en vez de 09, 0E, 0B y 0D. Esto aunque parezca demasiado hay que recordar que en una implementación de 8-bits, la multiplicación toma un tiempo considerablemente mayor y como resultado da lugar a una degradación en el desempeño de la implementación en 8-bits.

$$d^2(X) = \begin{pmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{pmatrix}$$

Entonces el utilizar $d^2(x)$ reduce el tiempo de ejecución pero a costa de una mayor complejidad y mayor tamaño de código. En la tabla 3.3 se muestran lo pasos necesarios para realizar este desarrollo.

Otra implementación seria desarrollar las operaciones de $c^3(X)$, cancelar los términos posibles y optimizar mediante la implementación a fin de sólo realizar el mínimo de operaciones.

$u = xtime(xtime(a[0] \bigoplus a[2]));$
$v = xtime(xtime(a[1] \bigoplus a[3]));$
$a[0] = a[0] \bigoplus u;$
$a[1] = a[1] \bigoplus v;$
$a[2] = a[2] \bigoplus u;$
$a[3] = a[3] \bigoplus v;$

Tabla 3.3: *Paso de preprocesamiento para el descifrado*

Expansión de la llave. La operación de expansión de la llave que genera a W obtenida iniciando con las ultimas Nk palabras de la llave de ronda y corriendo al revés la llave de cifrado original. Cuando se necesita calcular varias veces la llave de ronda al vuelo[2], es preferible calcular las últimas Nk palabras de la llave de ronda una vez y almacenarla para después utilizarla. La descripción de los cálculos de como se puede implementar, de forma que la salida de la llave de ronda quede en el orden que se necesita para el proceso de descifrado, se da en las tablas 3.4 y 3.5. Estas dan una descripción de *InvKeyExpansion()* en notación de pseudo código C. Hay que notar que la primera entrada, no es la llave de cifrado. En cambio Ki contiene las últimas Nk columnas de la llave expandida, generada desde la llave de cifrado por *KeyExpansion()*(ver la sección 2.3.9). Después de correr *InvKeyExpansion()*, Wi contiene la llave de ronda de descifrado en el orden requerido para descifrar, por ejemplo la columna con índice menor aparece al principio. Se puede notar que esta es la expansión de la llave que se debe ejecutar conjuntamente con el algoritmo sencillo de descifrado.

[2]En tiempo de ejecución

```
InvKeyExpansion(byte Ki[4][Nk],byte Wi[4][Nb(Nr+1)]) {
/***************** Para Nk <= 6 ********************/
    for(j=0;j<Nk;j++)
        for(i=0; i<4 ;i++) Wi[i][j]=Ki[i][j];
    for(j=Nk;j<Nb(Nr+1);j++) {
        if(j mod Nk ==0) {
            Wi[0][j] = Wi[0][j-Nk] xor S[Wi[i][j-1] xor
                Wi[1][j-2]] xor RC[Nr+1-j/Nk];
            for(i=1;i<4;i++)
                Wi[i][j]=Wi[i][j-Nk] xor S[Wi[1][j-1] xor
                    Wi[1][j-2]] xor RC[Nr+1-j/Nk];
        } else {
            for(i=0;i<4;i++)
                Wi[i][j]=Wi[i][j-Nk] xor Wi[i][j-Nk-1];
        }
    }
}
```

Tabla 3.4: *Algoritmo para la expansión inversa de la llave con Nk* ≤ 6

```
InvKeyExpansion(byte Ki[4][Nk],byte Wi[4][Nb(Nr+1)]) {
/***************** Para Nk > 6 ********************/
    for(j=0;j<Nk;j++)
        for(i=0; i<4 ;i++) Wi[i][j]=Ki[i][j];
    for(j=Nk;j<Nb(Nr+1);j++) {
        if(j mod Nk ==0) {
            Wi[0][j] = Wi[0][j-Nk] xor S[Wi[i][j-1] xor
                Wi[1][j-2]] xor RC[Nr+1-j/Nk];
            for(i=1;i<4;i++)
                Wi[i][j]=Wi[i][j-Nk] xor S[Wi[1][j-1] xor
                    Wi[1][j-2]] xor RC[Nr+1-j/Nk];
        } else if (j mod Nk == 4) {
                for(i=0;i<4;i++)
                    Wi[i][j]=Wi[i][j-Nk] xor S[Wi[i][j-Nk-1]];
            } else {
                for(i=0;i<4;i++)
                    Wi[i][j]=Wi[i][j-Nk] xor Wi[i][j-Nk-1];
            }
    }
}
```

Tabla 3.5: *Algoritmo para la expansión inversa de la llave con Nk* > 6

3.4.9. Integración y pruebas con el *sosse*

La norma ISO7816-4 define los comandos utilizados en la *smart card*, estos comandos son transformados al APDU (*application protocol data unit*),

CLA	INS	P1	P2	P3	...
Clase de comando	Instrucción	Parámetro 1	Parámetro 2	Longitud del block de datos	...

Tabla 3.6: *Forma genérica de una petición APDU*

Las respuestas son también en forma de APDU.

...	SW1	SW2
Block de datos	Estado de la palabra (high)	Estado de la palabra (low)

Tabla 3.7: *Forma genérica de una respuesta a una petición APDU*

En primer lugar, se realizaron pruebas para conocer el funcionamiento del *sosse*. Este sistema operativo trae una herramienta que sirve para poder emular su comportamiento directamente sobre la PC, esto se realiza compilando el proyecto utilizando el *Makefile.emu*, esto genera un ejecutable llamado *sosse-emu*, al ejecutar dicho archivo se presenta una interfaz en modo texto.

En ella aparece la palabra "*Receving*:"que indica que se esta listo para recibir una instrucción o APDU. El APDU que se uso para probar el *sosse* fue la autentificación interna, esta viene documentada en el manual del *sosse* [12], el APDU se muestra en la tabla 3.8, y la respuesta del APDU se muestra en la tabla 3.9, como se indica en el estándar la respuesta contiene el código "90 00"que indica que la instrucción fue ejecutada con éxito.

CLA	INS	P1	P2	P3
80	88	00	00	08

Tabla 3.8: *APDU para autentificación interna*

SW
88 90 00

Tabla 3.9: *Respuesta al APDU cuando no tiene ningún problema*

De acuerdo a la especificación de la librería de avr-lib se definieron los APDU como se muestra en las tablas 3.10 y 3.12 para el cifrado y descifrado respectivamente.

Para el cifrado se utilizó el código de instrucción *80*, quedando el APDU resultante, como se muestra en la tabla 3.10.

CLA	INS	P1	P2	P3	...
80	80	00	00	00	16 Byte de texto plano y en hexadecimal

Tabla 3.10: *APDU para cifrado usando AES*

Para la respuesta del cifrado y dado que se requería utilizar el modo ECB, el resultado se da inmediatamente seguido del código realizado "90 00", como se muestra en la tabla 3.11.

Texto cifrado	SW1	SW2
16 Byte de texto cifrado y en hexadecimal	90	00

Tabla 3.11: *Respuesta al cifrado si no hay problema*

El proceso de descifrado es muy similar, ya que al igual que el cifrado en el descifrado se usa el modo ECB, el código de la instrucción es el *88*, y el APDU resultante se muestra en la tabla 3.12.

CLA	INS	P1	P2	P3	...
80	88	00	00	00	16 Byte de texto cifrado y en hexadecimal

Tabla 3.12: *APDU para el descifrado usando AES*

Para la respuesta del descifrado el resultado se da inmediatamente seguido con el código realizado "90 00", como se muestra en la tabla 3.13.

Texto plano	SW1	SW2
16 Byte de texto plano y en hexadecimal	90	00

Tabla 3.13: *Respuesta al cifrado si no hay problema*

En caso de haber algún error en el proceso de cifrado o descifrado, el resultado es "6A 00"para indicar que el error es especifico al AES.

A continuación en las tablas 3.14, 3.15, 3.16 y 3.17 , se muestra el proceso de cifrado y descifrado utilizando los vectores de prueba proporcionados en el fips-197 [2].

CLA	INS	P1	P2	P3	...
80	80	00	00	00	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF

Tabla 3.14: *Ejemplo de APDU para cifrar usando AES*

...	SW1	SW2
8E A2 B7 CA 51 67 45 BF EA FC 49 90 4B 49 60 89	90	00

Tabla 3.15: *Respuesta al APDU del cifrado*

CLA	INS	P1	P2	P3	...
80	88	00	00	00	8E A2 B7 CA 51 67 45 BF EA FC 49 90 4B 49 60 89

Tabla 3.16: *Ejemplo de APDU para descifrar usando AES*

...	SW1	SW2
00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF	90	00

Tabla 3.17: *Respuesta al APDU del descifrado*

Capítulo 4

Resultados

En el presente capítulo se presenta un análisis de los resultados obtenidos, se dan los avances entre las distintas versiones desarrolladas y sus ventajas. Se comparan estos resultados con otros trabajos similares, se ofrecen métricas para realizar estas comparaciones y se explica el porqué se decidió concluir en estas versiones de desarrollo.

4.1. Implementación

Para realizar el presente proyecto se realizaron varios intentos para lograr una comprensión adecuada y lograr la optimización que se requiere. Para obtener un buen entendimiento del algoritmo AES, no solo basta con revisar la documentación, el entender que le sucede al estado entre una transformación y otra, es de suma importancia, debido a esta razón el primer acercamiento fue una implementación de este algoritmo para la PC, en primera instancia se realizó apagándose a la especificación FIPS-197 [2], aunque fue una versión funcional e ilustra claramente lo que sucede entre cada transformación distó de ser rápida, lo que llevo a revisar y aplicar algunos criterios de optimización para arquitectura basada en 32 bits. Como es el uso de *look up table* para realizar cálculos en frío [1] de productos sobre $GF(2^8)$ con el fin de acelerar la transformación *MixColumns()*, además aporta la ventaja adicional de realizar dichos productos en tiempo constante, caso similar sucede con las transformaciones *SubBytes()* y *ShiftRows()* requiriendo de cuatro tablas que requieren 4 kilobytes.

Al empezar a trabajar con la *smart card* fue bastante obvio que utilizar estos criterios no es recomendable por los escasos recursos que poseen estas tarjetas, aunque en la versión resultante obtenida de [6], indica que usando la memoria *ROM* se pueden incluirse algunas tablas, esta opción deja poco o nulo espacio a los demás programas que pueden requerir las aplicaciones e impiden el almacenamiento de datos.

La primera implantación sobre la arquitectura de 8 bits, se realizó en lenguaje ensamblador y al igual que en la primera realización se apego a la especificación del AES, con el fin de que fuera un punto base de donde partir y refinarlo en versiones posteriores, con la ventaja que se conocía ya bien al algoritmo, no se tuvieron contratiempos al implementarlo, salvo

[1] Son aquellos que se realizan a priori a la compilación del programa y están plasmados en el código del mismo

que no se pudo mantener el esquema de la llave completo en la RAM. La versión lograda es muy didáctica, sirvió para aprender el lenguaje ensamblador y contrastar las bondades que este proporciona, por el contrario, la eficiencia del esta implantación con respecto al tiempo de ejecución es baja.

Para las versiones posteriores se redujo el tamaño de código y el tiempo de ejecución. Algo importante para la optimización en este punto, fue el disminuir las llamadas a funciones, reducir los acceso a memoria, evitar guardar el estado de los registros entre una llamada a una transformación y otra, mezclar transformaciones y optimizaciones en el campo teórico.

En el caso de la función *xtime()*, la optimización consistió en convertirla en una macro, esta función se invoca cuatro veces en *MixColumns()* y siete en *InvMixColumns()*, dado que *xtime()* se implemento en cuatro instrucciones, esto implica que el incremento al código de programa es de 70 bytes.

Al cifrar y descifrar se llaman trece veces las transformaciones *MixColumns()* y *InvMixColumns()*, como en total estas emplean la función *xtime()* once veces y tomando en cuenta que la llamada a una función requiere siente ciclos de reloj, la mejora obtenida es de 1,001 ciclos de reloj al cifrar y descifrar un dato.

La propiedad que indica que el puede intercambiar el orden de ejecución de algunas transformaciones como se explico en el capítulo 3, permite mezclar las transformaciones *AddRoundKey()*, *ShifRows()* y *SubBytes()* en una *ShifRows_SubBytes_Addkey()*, reduciendo el número de accesos a memoria, debido a que cada transformación requiere traer el estado de memoria a los registros, esto permite serializarlas y eliminar el uso de algunos indices.

Como se menciono anteriormente en esta tesis se trabajó con una llave de 256 bits y el esquema de esta se genera bajo demanda. Esto da como resultado que se utilicen únicamente 32 bytes de *RAM* para esta tarea, aunque esto fuerza a modificar el código de cifrado y descifrado, lo que lleva a tener que desdoblar parcialmente los ciclos.

La arquitectura de la familia del procesadores ATMega163 posee, 32 registros de un byte y el estado ocupa 16 bytes, se decidió no mantenerlo en los registros debido que algunos de estos registros forman parte de una tarea específica, como es el caso del registro R0 que sirve como recipiente al recuperar un dato de manera indirecta, o los registros del R26 al R31 que sirven como indices *X*, *Y* y *Z*, para el direccionamiento indirecto y el uso de palabras de 32 bits, otra característica que impidió el mantener el estado fue que solo en los registro del R0 a R15 se pueden realizar operaciones con constantes y que en ocasiones hacen falta registros para mantener resultados parciales. Por estas razones se llego a la conclusión de que es mejor colocar la(s) palabra(s) que se requieran en los registros y el estado en la *RAM*.

En general esta versión cumple con un equilibrio entre eficiencia y volumen de código, en contraste es difícil de leer, comprender y realizar algún depurado. Algunos registros sirven como para pasar valores de manera implícita lo que dificulta el entender que sucede en algún momento específico.

Para el descifrado, se puede optar por dar mayor peso al criterio de reducción el tamaño

de código o al desempeño en tiempo de ejecución. Para el primer caso se puede llamar tres veces a la transformación *MixColumns()*, siendo la implementación de *InvMixColumns()* solo tres líneas, en el segundo caso se implementarla directamente *InvMixColumns()* como se muestra en la tabla 3.3 logrando reducir el tiempo en un 50 % pero la implementación es aproximadamente 60 líneas.

Al igual que en el cifrado se opto que al descifrar se mantenga el uso de 32 bytes de *RAM* para la expansión del esquema de la llave, aunque esto implique utilizar el doble del tiempo en el calculo del esquema de la llave que el empleado al cifrar. La única alternativa encontrada es mantener el esquema de llave completo en la *RAM*, -a diferencia de otros trabajos que utilizaron llaves menores (128 bits)- esto resulta ser demasiado para una *smart card*, porque el utilizar una llave de 256 bits implica manejar un esquema de 240 bytes.

Debido a que el descifrado requiere varios elementos del cifrado se decidió no incrementar más el tamaño código mezclado *AddRoundKey()* con *InvShifRows()* y *InvSubBytes()* y reutilizar lo mas posible el código de cifrado.

La última versión realizada fue una versión en el lenguaje C tomando en cuenta los criterios expuestos anteriormente, al ser compilado no se obtuvieron resultados alentadores, comparados a los obtenidos con la versión en ensamblador, debido a que el tamaño de código se incremente y se dispara el uso de la *RAM* como se muestra en la tabla 4.1, la ventaja obtenida de esta versión es que no está diseñada específicamente para la familia de chips ATMega163, en otras palabras es portable a otras arquitecturas de 8 bits.

Versión *SOSSE*	Text	Data
SOSSE con AES en C	10,042	586
SOSSE con AES en Ensamblador	9,792	48
SOSSE sin AES	7,898	2

Tabla 4.1: *Respuesta al APDU del descifrado*

4.2. Logros

Una forma de comprender mejor los resultados es realizando una comparativa con otros trabajos realizados, desafortunadamente no se encontró documentación de una implantación del AES en una *smart card* que maneje llaves de 256 bits lo que obligó a realizar las comparativas contra versiones que manejen llaves de 128 bits.

Debido a que se encontró un mejor desempeño en la versión implementada en ensamblador se presentan los resultado de esta versión, en la tabla 4.2, se presenta la cantidad de ciclos de reloj, utilizados por las transformaciones al cifrar y al descifrar.

Transformación	Ciclos de reloj
AddRoundKey	175 x 1 = 175
ShifRows_SubBytes_Addkey	248 x 14 = 3,472
MixColumns	229 x 13 = 2,977
KeyExpansion	267 x 7 = 1,869
InvKeyExpansion	273 x 7 = 1,911
IniciaInvKeyExpansion	1,580 x 1 = 1,580
InvMixColumns	359 x 13 = 4,667
InvShifRows_InvSubBytes	203 x 14 = 2,842

Tabla 4.2: *Métricas obtenidas de las transformaciones realizadas*

En la tabla 4.3, se presentan los tiempos promedio al cifrar y descifrar obtenidos en las dos versiones del AES desarrolladas en ensamblador, estos tiempos están dados en milisegundos.

Versión	Ciclos de reloj	Tamaño del Código (bytes)	Memoria *RAM* (bytes)	Tiempo de ejecución (ms)
cifra_v1	15,514	938	48	3.8785
descifra_v1	25,971	1,416	48	6.49275
cifra_v2	8,779	968	48	2.19475
descifra_v2	14,080	1,266	48	3.52

Tabla 4.3: *Métricas obtenidas de los trabajos realizados*

4.3. Comparación con otros trabajos realizados

Al realizar la comparación hay que enfatizar que se esta comparando el trabajo obtenido en esta tesis con otros trabajos que manejan llaves de 128 bits, debido a que al utilizar llaves de 256 bits se incrementa el tamaño del código y el desempeño, esto da la falsa impresión que se puede mejorar en mucho lo realizado, pero el uso de una llave del doble de longitud implica que no solo se invoquen las transformaciones un mayor número de veces, si no que además algunas de ellas deben realizar una tarea mayor, como es el caso de *KeyExpansion()*.

Desafortunadamente no se tuvo acceso a los fuentes de otros trabajos realizados, lo que obligó a tomar los resultados documentados y no poder ratificarlos ni adaptarlos para realizar nuestras comparaciones, en la tabla 4.4 se dan los tiempos documentados en [5], donde se muestra el desempeño de las transformaciones, debido a que se combinó las transformaciones de distinta manera no se pudo comparar como se quisiera los tiempos con los que se obtuvieron en este trabajo. La transformación *MixColumns()* que ocupa tiempos muy similares, se compara con la versión más rápida, la diferencia es de 23 ciclos de reloj, con respecto a la expansión de la llave cabe destacara que tanto la longitud de la llave como la del esquema son del doble, lo que implica el doble de tiempo reportado en otras versiones para poder compararlo con el que se obtuvo en este trabajo. La transformación *calcula_siguiente_clave()* requiere 102 ciclos y al cifrar se llama diez veces, ocupando 1,020 ciclos de reloj, la transformación *KeyExpansion()* requiere 267 ciclos de reloj y es llamada en siete ocasiones, lo que implica 1,869 ciclos. Dado que 1,020 x 2= 2,040 es mayor al tiempo obtenido se considera una buena

implementación. La transformación *AddRoundKey()* se tuvo que modificar para manejar un esquema de llaves de 32 bytes a diferencia de *transposicion()*, lo que implica unas cuantas instrucciones, pero al obtener una diferencia de ocho ciclos de reloj se considera correcta la optimización. Una comparación gráfica de las transformaciones se muestra en la figura 4.1.

Transformación	Ciclos en AES rápido	Ciclos en AES pequeño
transposición	3 x 57 = 171	3 x 57 = 171
xor_sust_recorre	10 x 168 = 1680	10 x 207 = 2070
mezcla_columnas	9 x 206 = 1854	9 x 300 = 2700
calcula_siguiente_clave	10 x 102 = 1020	10 x 139 = 1390
resto_ultima_ronda	129	142

Tabla 4.4: *Desempeño de las transformaciones del AES dadas por el Ing. Alberto Nicolas [5]*

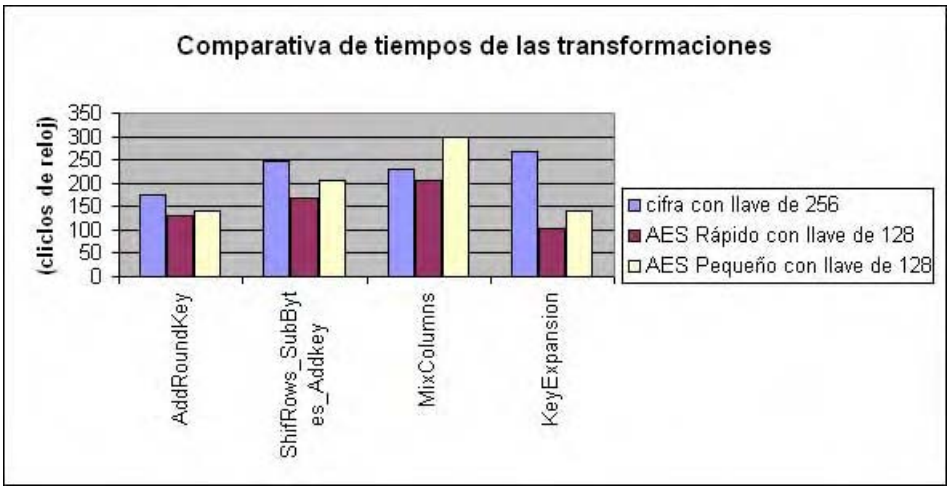

Figura 4.1: *Gráfica comparativa de los tiempos de las transformaciones*

El realizar las comparaciones contra los tiempos totales de los trabajos [5] y [6] es algo forzoso, aunque como ya se menciono ambos trabajos manejan llaves de 128 bits y se documento que en ambos trabajos se desarrollaron dos versiones, las cuales se nombraron como AES Rápido, el AES Pequeño, rijndael_dir_v1 y rijndael_dir_v2, respectivamente.

Se muestra en la tabla 4.5 los tamaños de código y los tiempos documentados en los trabajos ya mencionados.

Versión	Ciclos de reloj	Tamaño del Código (bytes)	Memoria *RAM* (bytes)	Tiempo de ejecución (ms)
AES Rápido	4,911	772	32	1.29
AES Chico	6,553	598	32	1.65
rijndael_dir_v1	3,899	1,252	32	0.97475
rijndael_dir_v2	3,755	1,018	32	0.93875

Tabla 4.5: *Métricas obtenidas para las versiones de cifrado directo del algoritmo AES reportadas en [5] y [6]*

Solo en el trabajo [6] se documentan los tiempos del cifrado inverso o descifrado, estos tiempos se dan en la tabla 4.6.

Versión	Ciclos de reloj	Tamaño del Código (bytes)	Memoria *RAM* (bytes)	Tiempo de ejecución (ms)
rijndael_inv_v1	5,138	1470	32	1.2845
rijndael_inv_v2	4,228	1334	192	1.057

Tabla 4.6: *Métricas obtenidas para las versiones de cifrado inverso del algoritmo AES reportadas en [6]*

Debido a que el *sosse* incluye un esquema de seguridad basado en la implementación del TEA y a este esquema se le agregara la versión del AES, se decidió comparar los resultados obtenidos contra el desempeño y el tamaño de código del algoritmo TEA, como el *sosse* incluye dos versiones de este algoritmo una ensamblador y una en C, se tomaron las métricas de la versión en ensamblador. En la tabla 4.7 se muestran los valores obtenidos tanto en desempeño como en volumen de código.

Versión	Tamaño del Código (bytes)	Tiempo de ejecución (ms)
TEA	318	3.14

Tabla 4.7: *Métricas del TEA*

En la figura 4.2 se muestra de forma gráfica la comparación sobre el desempeño del *AES Rápido*, *AES Chico*, *rijndael_dir_v1*, *rijndael_dir_v2* y *cifra_v2*.

Se muestran en la figura 4.3 las comparaciones sobre el volumen del código requerido por las versiones *AES Rápido*, *AES Chico*, *rijndael_dir_v1*, *rijndael_dir_v2* y *cifra_v2*.

En la gráfica 4.2 se destaca el mal desempeño del TEA, si se compara con otras versiones que también manejan llaves de 128 bits o mayores, el algoritmo TEA es sumamente sencillo, muy compacto pero su desempeño es bajo, otro punto que se encontró a lo largo de este trabajo es que al incrementar la cantidad de código de programa el desempeño mejora y viceversa, como es el caso de las versiones optimizadas con respecto al desempeño. Aunque la versión *AES Chico* tiene balanceados los dos criterios, su desempeño no es bueno. Los resultados obtenidos en las gráficas 4.2 y 4.3 muestran que la versión *cifra_v2* tiene un buen balance entre estos dos criterios logrando, buenos resultados en ambos objetivos.

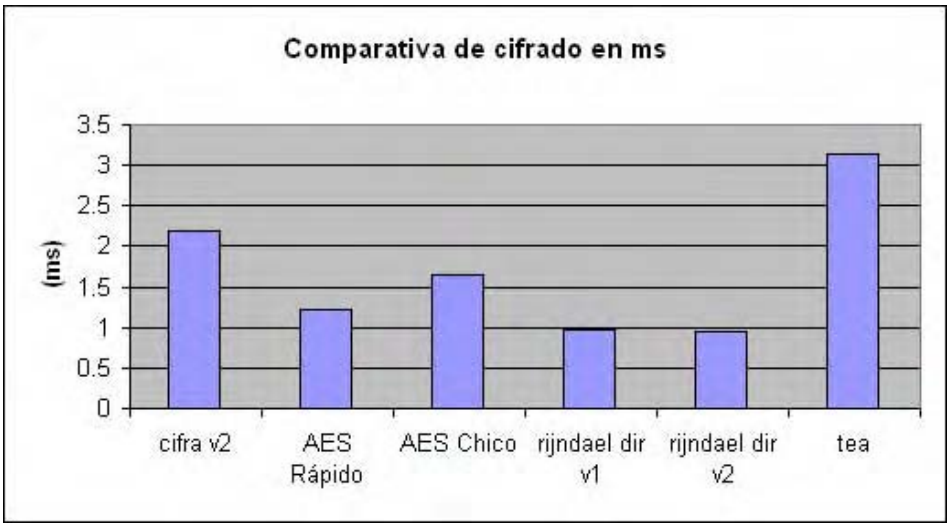

Figura 4.2: *Gráfica comparativa de los tiempos en el cifrado*

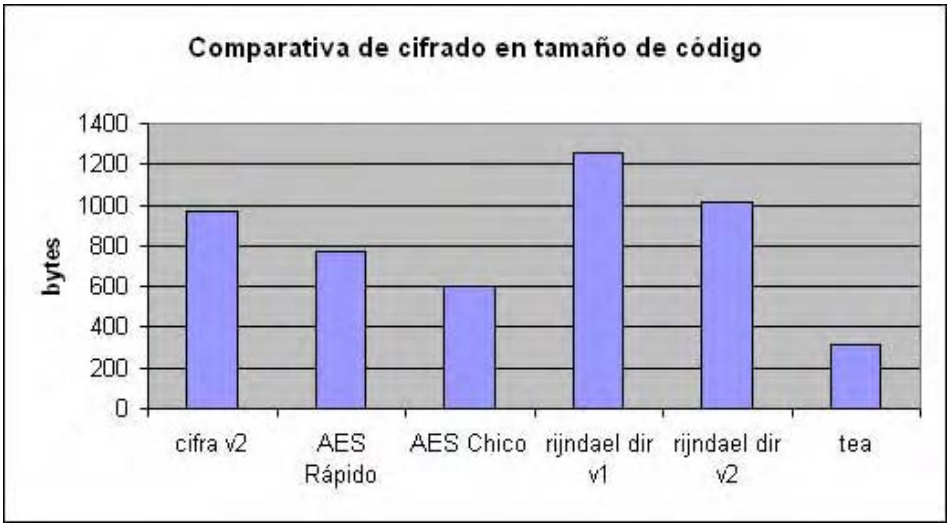

Figura 4.3: *Gráfica comparativa de los tamaños de palabras de código para el cifrado*

En la figura 4.4 se da la gráfica de los tiempos de ejecución mostrados en las tablas 4.5 y 4.6.

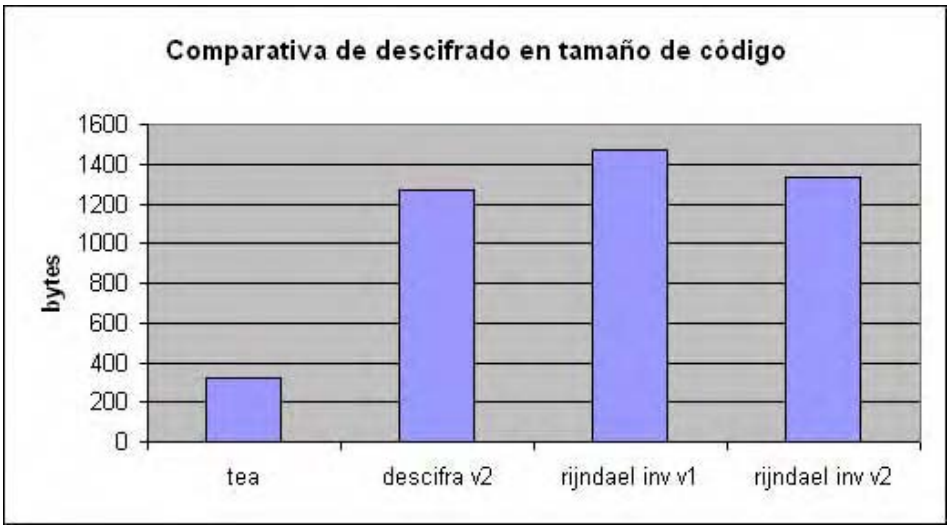

Figura 4.4: *Gráfica comparativa de los tamaños de palabras de código para el descifrado*

La gráfica de los tiempos en el descifrado puede verse en la figura 4.5, estos tiempos fueron dados en las tablas 4.5 y 4.6.

Al comparar el descifrado, se espera que al igual que sucede con el cifrado el tiempo de descifrado se incremente. Aunque el incremento fue considerablemente mayor que en las versiones reportadas en [6], esto se atribuye al hecho que utiliza más de 192 bytes de memoria *RAM*, que el tamaño del código de descifrado es mayor, que mantiene el esquema de llave completo en memoria y al hecho de que la longitud de la llave utilizada es el doble.

Se decidió darle mayor peso al hecho de no extenderse más en tamaño del código debido a que el descifrado no es ocupado en los métodos de autentificación interna, ni externa.

Aparte de utilizarse el descifrado de forma directa, este se puede ocupar para el almacenamiento de información cifrada. Esto requeriría tener mayor espacio en la tarjeta para almacenar otras aplicaciones y/o datos.

Dado que se utilizó una tarjeta ATMega163 y esta posee 16k de memoria de programa, se tuvo la facilidad de mantener el TEA y añadir el AES, esto con la finalidad de mantener la compatibilidad con otras versiones anteriores de este COS.

Tanto las versiones desarrolladas en ensamblador como las versiones en C se compilan con la ejecución el comando *make* proporcionado en el fuente y al ser ejecutado genera los archivos: *sosse*, *sosse.bin*, *sosse.hex*, *eedata.bin*, *eedata.hex*. De los cuales *sosse.bin* y *eedata.bin* están listos para ser copiados en la *smart card*. La versión en C tiene además otro *Makefile.emu* que permite simular el comportamiento del *sosse* desde la PC, para ejecutarlo

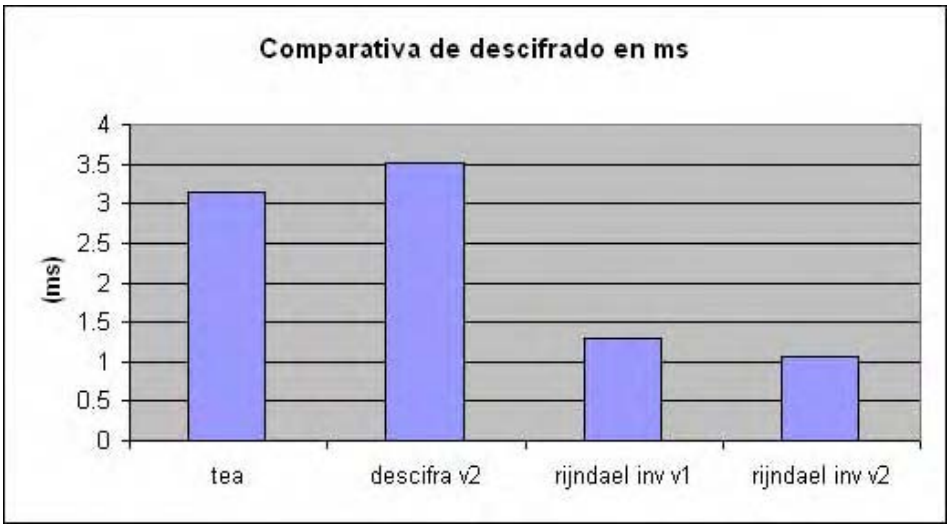

Figura 4.5: *Gráfica comparativa de los tiempos en el descifrado*

se debe dar el comando *make -f Makefile.emu* y debe generar el archivo *sosse-emu* el cual es el emulador. Este emulador se maneja directamente con los *APDUs*, estos los envía a ser ejecutados por el *sosse* y regresa los *APDUs* resultantes.

Los *APDUs* añadidos correspondientes al cifrado empleando el modo ECB, o Autentificación externa con el AES es *80 80 00 00 00 16-bytes* y para el descifrado es el *80 88 00 00 00 16-bytes*

Capítulo 5

Conclusiones

En este trabajo se utilizó la tecnología de las *smart cards* para sus aplicaciones más usadas, así como los protocolos utilizados, sus sistemas operativos, el sistema de archivos y su lenguaje ensamblador. Se analizó el funcionamiento de un sistema operativo para las *smart cards* con un enfoque de seguridad, se investigaron qué comandos se usan para dicha tarea y como se efectúan las funciones de autentificación. Se estudiaron los mecanismos de seguridad que implementan las *smart cards* basados en métodos simétricos. En particular se estudió la especificación del algoritmo AES analizando cada una de sus transformaciones para el cifrado directo e inverso, desde la perspectiva de las operaciones matemáticas con las que se definió y tomando en cuenta la forma de implementarlas de forma eficiente en la arquitectura RISC[1] de 8 bits.

Se desarrollaron varias implantaciones del AES, la primera versión desarrollada fue en C, se comprendió los modos de uso y qué papel juega cada una de transformaciones en el cifrado y descifrado, las versión hecha en ensamblador sirvió para conocer el ensamblador de la tarjeta, su acceso a memoria y poder entender el cómo se podían realizar las optimizaciones, dos versiones fueron optimizadas: una a bajo nivel en ensamblador y otra en C, las cuales cumplieron un criterio de diseño, gracias a esto se tiene la capacidad de funcionar en la arquitectura real, en especial la versión en ensamblador que hace un mejor uso de los recursos, logrando un buen desempeño. De esto se encontró que la versión en C ocupa mayor cantidad de memoria y el desempeño no varia significativamente respecto a la versión en ensamblador.

La tarjeta utilizada fue la ATMega163 que utiliza un lenguaje ensamblador que es claro y fácil de aprender, el código resultante es compacto, tiene un mapeo muy directo desde lenguaje C, y se cuenta con un eficiente conjunto de instrucciones. Tal es el caso de aquellas que permiten el direccionamiento indirecto con desplazamiento y las que efectúan un post incremento, o pre decremento, del índice de 16 bits, muy útiles en la parte de optimización del algoritmo. El código ensamblado de la compilación del código en C requiere mucho más memoria, el desempeño y el volumen de código son ligeramente mayor por esta razón el uso del lenguaje ensamblador es una mejor opción.

Con respecto a las transformaciones se encontró que debido a los pocos recursos de la *smart card* no se pueden aplicar las mismas optimizaciones que en una PC, un ejemplo

[1] Procesador con un conjunto de instrucciones reducido, (*Reduced Instruction Set Computer*)

importante es que en una PC se puede hacer "cálculos en frío" y almacenarlos en arreglos que posteriormente dan gran mejora al desempeño, caso especifico con la multiplicación en el campo $GF(2^8)$ utilizadas en la transformación *SubBytes()*, también debido a los pocos recursos de la *smart card* se tuvo que dar mayor peso al tamaño del código resultante y esto llevo a probar optimizaciones no solo en cuanto al desempeño sino también en tamaño de código y acceso a memoria. Por lo que podemos decir que las optimizaciones a bajo nivel dependen del tipo de arquitectura en la que se trabaje.

Como parte fundamental del objetivo de esta tesis se trata de mejorar la seguridad para el uso de las *smart cards*, esto se realizó incrementando el tamaño de llave utilizada. Trabajos anteriores que se enfocan al uso del AES y usan llaves de 128 bits. En el presente trabajo se logra utilizar de una manera optimizada llaves de 256 bits, desafortunadamente no se encontró documentación sobre los tiempos de alguna otra implementación del AES que use llave de 256 bits para las *smart cards*, por lo tanto se decidió hacer una comparativa a nivel de transformaciones y equiparar de alguna manera las transformaciones, ya que el desempeño se ve directamente ligado al tamaño de llave.

Debido a que las principales aplicaciones de las *smart cards* son la autentificación, su resguardo es de poca información critica y no requiere cifrar de grandes volúmenes de información, los resultados obtenidos son bastante buenos, el nivel de optimización logrado fue equilibrado, tanto en tiempo de ejecución, como en tamaño de código y uso de memoria. En general se trata de aprovechar al máximo los escasos recursos de la *smart card*, esto con el fin de dejar espacio en dicha tarjeta para el incremento de futuras aplicaciones, ya que hay que recordar que el AES es una herramienta que le brinda al *sosse* un mayor grado de seguridad. Por lo que se concluye que el aumento de seguridad se logra cambiando el algoritmo de cifrado y manejando llaves de mayor longitud. El intercambiar el algoritmo no degradó el desempeño del SO.

La mayoría de los problemas que se presentaron en este trabajo fueron debido a que el *hardware* utilizado para interactuar con las tarjetas, corresponde a una versión antigua y encontrar soporte para el mismo no siempre es posible, así que se recomienda utilizar lectores más recientes en trabajos futuros.

5.1. Trabajos futuros

Como se comento en la sección 2.5 las *smart cards* están encapsuladas en una sola pastilla y no tienen un punto interno para atacarlas, en otras palabras no hay forma de saber que hay en cierto momento en la memoria, o que instrucción se esta ejecutando, etc. Esto solo deja que los ataques se puedan realizar midiendo los voltajes de salida y los tiempos de respuesta, por lo que se recomienda realizar un análisis de posibles ataques y ver que tan vulnerable es el uso del AES en una *smart card* bajo este contexto.

Bibliografía

[1] Cryptography and network segurity: priciples and practice " William Stallings "·Segunda edición · "Prentice Hall "· 4th Ed 2005

[2] Federal Information Processing Standards Publication 197 · ADVANCED ENCRYPTION STANDARD (AES) · November 26, 2001

[3] Linear Algebra · Stephen H. Friedberg, Lawrence E. Spence, Arnold J. Insel · Prentice Hall · November 2002

[4] North-Holland Mathematical Library, Vol 16 · The Theory of Error-Correcting Codes · F. J. MacWilliams, N.J.A. Sloane · North-Holland publishing company Ámsterdam · new york · Oxford. July 1998

[5] Tesis de Licenciatura: El algoritmo de cifrado AES: su implementación y optimización de bajo nivel para mejorar los mecanismos de seguridad de una tarjeta inteligente · Ing. Alberto Nicolas Escalante Bañuelos · Julio 2003

[6] Tesis de Maestria: Estudio y realización de mecanismos de seguridad par alas tarjetas inteligentes (*smart cards*), basados en el algoritmo AES (Advanced Encryption Standard) · Mtro. Manuel Alexander Moreno Liy · Agosto 2006

[7] Paul C. Clark and Lance J. Hoffman · "Bits: A Smartcard Protected Operating System"· Communications of the ACM · pp. 66 - 94 · November 1994 Vol 37 Number 11.

[8] The Design of Rijndael · AES - The Advanced Encryption Standard · Joan Daemen, Vincent Rijmen · Stringer 2001 Stringer

[9] ISO 7816-3 *Smart Card* Standard · Part 3· Electronic Signals and Transmission Protocols ISO/IEC 7816-3

[10] G. Keating · "Performance Analysis of AES candidates onthe 6805 CPU core"· http://www-08.nist.gov/CryptoToolkit/aes/round1/comments/990415-gkeating.pdf

[11] InvMixColumn Decomposition and Multilevel Resource Sharing in AES Implementatios · Viktor Fisher, Miloš Drutarovský · http://ieeexplore.ieee.org/iel5/92/32379/01512188.pdf

Ligas relevantes

[12] SOSSE Reference Manual· Doxygen 1.2.14· http://www.mbsks.franken.de/sosse/ · sosse-manual.pdf

[13] Naccache, David and David M'Raïhi. 1996. Cryptographic Smart Cards. IEEE Micro 6:14, 16-19, 21 - 24 · http://ieeexplore.ieee.org/iel5/92/32379/01512188.pdf

[14] AES page available via http://www.nist.gov/CryptoToolkit.

Apéndice A

Glosario

$\cdot\ni\cdot$ Símbolo matemático que indica tal que.

$\forall$ Símbolo matemático que indica para todo elemento.

$\exists$ Símbolo matemático que indica existencia de un elemento.

$\oplus$ Operación de o-exclusiva.

AES Es el algoritmo de Rijndael, que es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos, *Advanced Encryption Standard*.

APDU Es el protocolo de comandos que se utiliza para ejecutar una aplicación, *Application Protocol Data Units*.

BIOS El sistema Básico de entrada/salida es un código de interfaz que localiza y carga el sistema operativo en la RAM, *Basic Input-Output System*.

Cálculos en frío Son aquellos que se realizan a priori a la compilación del programa y están plasmados en el código del mismo.

CBC Es un modo de cifrado, antes de ser cifrado, a cada bloque de texto se le aplica una operación XOR con el previo bloque ya cifrado, *Cipher-Block Chaining*.

CBC-MAC Es el modo de cifrado CBC que incluye autentificación de mensaje.

CFB Modos que hace que el cifrado en bloque opere como una unidad de flujo de cifrado: se generan bloques de flujo de claves, que son operados con XOR y el texto en claro para obtener el texto cifrado, *Cipher FeedBack*.

$CIPH_k$ Notación para reprecentar el cifrado de un bloque utilizando AES.

$CIPH_k^{-1}$ Notación para reprecentar el descifrado de un bloque utilizando AES.

CMAC *Cipher-based Message Authentication Code*.

COS Sistema operativo para las tarjetas inteligentes, *Card Operating System*.

CPU Unidad central de procesamiento, *Central Processing Unit*.

CTR Modo contador, convierte una unidad de cifrado por bloques en una unidad de flujo de cifrado. Genera el siguiente bloque en el flujo de claves cifrando valores sucesivos de un contador.

DES Es un algoritmo de cifrado, es decir, un método para cifrar información, escogido como FIPS en los Estados Unidos en 1976, *Data Encryption Standard*.

ECB Es el método más simple, en el cual el mensaje es partido en bloques, cada uno de los cuales es cifrado de manera separada, *Electronic CodeBook*.

EEPROM Memorias programables reescribirles eléctricamente, *Electrically erasable programmable memory*.

EPROM Memorias programables eléctricamente,*Electrically programmable memory*.

FIPS-197 Se refiere al estándar donde se define el algoritmo de cifrado AES FIPS por las siglas *Federal Information Processing Standards*.

GF Campos de Galois, *Galois Field*.

Generar al vuelo Que se calcule en tiempo de ejecución.

HOST El término equipo anfitrión a aquel dispositivo que ofrece servicios.

ISO Es la Organización Internacional para la Estandarización, se encarga de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales, *International Organization for Standardization*.

Lap Laptop o computadora personal portátil.

MAC Código de autenticación de mensajes es un código que se genera a partir de un mensaje de longitud arbitraria y de una clave secreta compartida entre remitente y destinatario, y que sirve para autenticar el mensaje, *Message Authentication Code*.

MOD Función de módulo.

NIST El Instituto Nacional de Normas y Tecnología es una agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos, promueve la innovación y la competencia industrial mediante avances, *National Institute of Standards and Technology*.

OFB Modo que hace que el cifrado en bloque opere como una unidad de flujo de cifrado: se generan bloques de flujo de claves, que son operados con XOR y el texto en claro para obtener el texto cifrado, *Output FeedBack*.

PC Computadora Personal, *Personal Computer*.

RAM Memoria de acceso aleatorio o memoria de acceso directo, *Random Access Memory*.

Red de Feistel Es un método de cifrado en bloque con una estructura particular, presentan la ventaja de ser reversible por lo que las operaciones de cifrado y descifrado son idénticas, requiriendo únicamente invertir el orden de las subclaves utilizadas, el más popular de estos es el DES.

Reto respuesta También llamado modelo de autentificación externa, ya que el *host* genera un reto, este se manda a la *smart card*, la smart card aplica el algoritmo de cifrado a la cadena, manda la respuesta al *host*, este cifra aplica la cadena original y lo compara con la cadena recibida, si las cadenas son iguales se acepta la autentificación, de lo contrario se rechaza.

RISC Procesador con un conjunto de instrucciones reducido , *Reduced Instruction Set Computer*.

ROM Memoria de solo lectura, *Read Only Memory*.

SIM Módulo de Identificación del Suscriptor es una tarjeta inteligente desmontable usada en teléfonos móviles que almacena de forma segura la clave de servicio del suscriptor usada para identificarse ante la red, *Subscriber Identify Module*.

SO Sistema operativo, *Operative system*.

Sosse Sistema operativo para las tarjetas inteligentes con licencia de código abierto,*Simple Operating System for Smartcard Education*.

S-box En criptografía, una S-Box o caja de sustitución es un componente básico de los algoritmos de cifrado de clave simétrica, *substitution box*.

TDES Véase Triple DES.

TEA Pequeño algoritmo de cifrado, *Tiny Encryption Algorithm*.

Triple DES Triple DES se llama al algoritmo que hace triple cifrado del DES. También es conocido como TDES.

Apéndice B

$\boldsymbol{GF}(2^8)$

En esta sección se indica el método con el cual se generan los elementos del $GF(2^8)$ y además se listan los mismos, estos elementos son los utilizados en el proceso de cifrado y descifrado en el algoritmo AES, este GF es generado utilizando el polinomio $m(x)$ donde

$$m(x) = 1 + x + x^3 + x^4 + x^8$$

El elemento generador del GF es elemento es el $p = 1 + x$, el proceso para generar a los elemento de este campo es el siguiente:

1) Se incluye el elemento cero.
2) Se incluye el elemento generador p como primer elemento
3) Se toma al elemento anterior, se multiplica por p y se saca el módulo $m(x)$. Se incluye en los elementos de GF
4) Se repite el paso anterior hasta repetir a el elemento p y termina, note que este elemento no se incluye debido a que ya esta.

En otras palabras después de haber incluido el cero y p los siguientes elementos son potencias de p, p^2, p^3, etc., via módulo $m(x)$. Para ejemplificar este proceso se generan los primeros elementos:

$$\begin{aligned} p^2 \bmod(m(x)) &= p * p \bmod(m(x)) \\ &= (1+x) * (1+x) \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x + x + x^2 \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x^2 \end{aligned}$$

$$\begin{aligned} p^3 \bmod(m(x)) &= p^2 * p \bmod(m(x)) \\ &= (1+x^2) * (1+x) \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x^2 + x + x^3 \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x + x^2 + x^3 \end{aligned}$$

A continuación se listan los elementos del $GF(2^8)$:

0	0
1	1+x
2	$1+x^2$
3	$1+x+x^2+x^3$
4	$1+x^4$
5	$1+x+x^4+x^5$
6	$1+x^2+x^4+x^6$
7	$1+x+x^2+x^3+x^4+x^5+x^6+x^7$
8	$x+x^3+x^4$
9	$x+x^2+x^3+x^5$
10	$x+x^4+x^5+x^6$
11	$x+x^2+x^4+x^7$
12	$1+x^5+x^7$
13	$x^3+x^4+x^5+x^6+x^7$
14	$1+x+x^4$
15	$1+x^2+x^4+x^5$
16	$1+x+x^2+x^3+x^4+x^6$
17	$1+x^5+x^6+x^7$
18	$x^3+x^4+x^5$
19	x^3+x^6
20	$x^3+x^4+x^6+x^7$
21	$1+x+x^4+x^5+x^6$
22	$1+x^2+x^4+x^7$
23	$x^2+x^5+x^7$
24	$1+x+x^2+x^4+x^5+x^6+x^7$
25	x
26	$x+x^2$
27	$x+x^3$
28	$x+x^2+x^3+x^4$
29	$x+x^5$
30	$x+x^2+x^5+x^6$
31	$x+x^3+x^5+x^7$
32	$1+x^2+x^5+x^6+x^7$
33	$x^2+x^4+x^5$
34	$x^2+x^3+x^4+x^6$
35	$x^2+x^5+x^6+x^7$
36	$1+x+x^2+x^4+x^5$
37	$1+x^3+x^4+x^6$
38	$1+x+x^3+x^5+x^6+x^7$
39	$x+x^2+x^5$
40	$x+x^3+x^5+x^6$
41	$x+x^2+x^3+x^4+x^5+x^7$
42	$1+x^3+x^4+x^6+x^7$
43	$x^4+x^5+x^6$

44	x^4+x^7
45	$1+x+x^3+x^5+x^7$
46	$x+x^2+x^5+x^6+x^7$
47	$1+x^4+x^5$
48	$1+x+x^4+x^6$
49	$1+x^2+x^4+x^5+x^6+x^7$
50	x^2
51	x^2+x^3
52	x^2+x^4
53	$x^2+x^3+x^4+x^5$
54	x^2+x^6
55	$x^2+x^3+x^6+x^7$
56	$1+x+x^2+x^3+x^6$
57	$1+x^4+x^6+x^7$
58	$x^3+x^5+x^6$
59	$x^3+x^4+x^5+x^7$
60	$1+x+x^4+x^6+x^7$
61	$x+x^2+x^3+x^5+x^6$
62	$x+x^4+x^5+x^7$
63	$1+x^2+x^3+x^6+x^7$
64	$x^2+x^3+x^6$
65	$x^2+x^4+x^6+x^7$
66	$1+x+x^2+x^5+x^6$
67	$1+x^3+x^5+x^7$
68	$x^5+x^6+x^7$
69	$1+x+x^3+x^4+x^5$
70	$1+x^2+x^3+x^6$
71	$1+x+x^2+x^4+x^6+x^7$
72	$x+x^5+x^6$
73	$x+x^2+x^5+x^7$
74	$1+x^4+x^5+x^6+x^7$
75	x^3
76	x^3+x^4
77	x^3+x^5
78	$x^3+x^4+x^5+x^6$
79	x^3+x^7
80	$1+x+x^7$
81	$x+x^2+x^3+x^4+x^7$
82	$1+x^3+x^4+x^5+x^7$
83	$x^4+x^6+x^7$
84	$1+x+x^3+x^5+x^6$
85	$1+x^2+x^3+x^4+x^5+x^7$
86	$x^2+x^3+x^4+x^6+x^7$
87	$1+x+x^2+x^3+x^4+x^5+x^6$

88	$1+x^7$
89	$x^3+x^4+x^7$
90	$1+x+x^4+x^5+x^7$
91	$x+x^2+x^3+x^6+x^7$
92	$1+x^3+x^6$
93	$1+x+x^3+x^4+x^6+x^7$
94	$x+x^2+x^4+x^5+x^6$
95	$x+x^3+x^4+x^7$
96	$1+x^2+x^4+x^5+x^7$
97	$x^2+x^6+x^7$
98	$1+x+x^2+x^4+x^6$
99	$1+x^3+x^4+x^5+x^6+x^7$
100	x^4
101	x^4+x^5
102	x^4+x^6
103	$x^4+x^5+x^6+x^7$
104	$1+x+x^3$
105	$1+x^2+x^3+x^4$
106	$1+x+x^2+x^5$
107	$1+x^3+x^5+x^6$
108	$1+x+x^3+x^4+x^5+x^7$
109	$x+x^2+x^4+x^6+x^7$
110	$1+x^5+x^6$
111	$1+x+x^5+x^7$
112	$x+x^2+x^3+x^4+x^5+x^6+x^7$
113	$1+x^3+x^4$
114	$1+x+x^3+x^5$
115	$1+x^2+x^3+x^4+x^5+x^6$
116	$1+x+x^2+x^7$
117	$x+x^4+x^7$
118	$1+x^2+x^3+x^5+x^7$
119	$x^2+x^3+x^5+x^6+x^7$
120	$1+x+x^2+x^3+x^5$
121	$1+x^4+x^5+x^6$
122	$1+x+x^4+x^7$
123	$x+x^2+x^3+x^5+x^7$
124	$1+x^3+x^5+x^6+x^7$
125	x^5
126	x^5+x^6
127	x^5+x^7
128	$1+x+x^3+x^4+x^5+x^6+x^7$
130	$x+x^3+x^4+x^5$
131	$x+x^2+x^3+x^6$
132	$x+x^4+x^6+x^7$

133	$1+x^2+x^3+x^5+x^6$
134	$1+x+x^2+x^4+x^5+x^7$
135	$x+x^6+x^7$
136	$1+x^2+x^3+x^4+x^6$
137	$1+x+x^2+x^5+x^6+x^7$
138	$x+x^4+x^5$
139	$x+x^2+x^4+x^6$
140	$x+x^3+x^4+x^5+x^6+x^7$
141	$1+x^2+x^4$
142	$1+x+x^2+x^3+x^4+x^5$
143	$1+x^6$
144	$1+x+x^6+x^7$
145	$x+x^2+x^3+x^4+x^6$
146	$x+x^5+x^6+x^7$
147	$1+x^2+x^3+x^4+x^5$
148	$1+x+x^2+x^6$
149	$1+x^3+x^6+x^7$
150	x^6
151	x^6+x^7
152	$1+x+x^3+x^4+x^6$
153	$1+x^2+x^3+x^5+x^6+x^7$
154	$x^2+x^3+x^5$
155	$x^2+x^4+x^5+x^6$
156	$x^2+x^3+x^4+x^7$
157	$1+x+x^2+x^3+x^4+x^5+x^7$
158	$x+x^3+x^4+x^6+x^7$
159	$1+x^2+x^4+x^5+x^6$
160	$1+x+x^2+x^3+x^4+x^7$
161	$x+x^3+x^4+x^5+x^7$
162	$1+x^2+x^4+x^6+x^7$
163	$x^2+x^5+x^6$
164	$x^2+x^3+x^5+x^7$
165	$1+x+x^2+x^3+x^5+x^6+x^7$
166	$x+x^3+x^5$
167	$x+x^2+x^3+x^4+x^5+x^6$
168	$x+x^7$
169	$1+x^2+x^3+x^4+x^7$
170	$x^2+x^3+x^4+x^5+x^7$
171	$1+x+x^2+x^3+x^4+x^6+x^7$
172	$x+x^3+x^4+x^5+x^6$
173	$x+x^2+x^3+x^7$
174	$1+x^3+x^7$
175	x^7
176	$1+x+x^3+x^4+x^7$

177	$x+x^2+x^4+x^5+x^7$
178	$1+x^6+x^7$
179	$x^3+x^4+x^6$
180	$x^3+x^5+x^6+x^7$
181	$1+x+x^5$
182	$1+x^2+x^5+x^6$
183	$1+x+x^2+x^3+x^5+x^7$
184	$x+x^3+x^5+x^6+x^7$
185	$1+x^2+x^5$
186	$1+x+x^2+x^3+x^5+x^6$
187	$1+x^4+x^5+x^7$
188	$x^3+x^6+x^7$
189	$1+x+x^6$
190	$1+x^2+x^6+x^7$
191	$x^2+x^4+x^6$
192	$x^2+x^3+x^4+x^5+x^6+x^7$
193	$1+x+x^2+x^3+x^4$
194	$1+x^5$
195	$1+x+x^5+x^6$
196	$1+x^2+x^5+x^7$
197	$x^2+x^4+x^5+x^6+x^7$
198	$1+x+x^2$
199	$1+x^3$
200	$1+x+x^3+x^4$
201	$1+x^2+x^3+x^5$
202	$1+x+x^2+x^4+x^5+x^6$
203	$1+x^3+x^4+x^7$
204	$x^4+x^5+x^7$
205	$1+x+x^3+x^6+x^7$
206	$x+x^2+x^6$
207	$x+x^3+x^6+x^7$
208	$1+x^2+x^6$
209	$1+x+x^2+x^3+x^6+x^7$
210	$x+x^3+x^6$
211	$x+x^2+x^3+x^4+x^6+x^7$
212	$1+x^3+x^4+x^5+x^6$
213	$1+x+x^3+x^7$
214	$x+x^2+x^7$
215	$1+x^4+x^7$
216	$x^3+x^5+x^7$
217	$1+x+x^5+x^6+x^7$
218	$x+x^2+x^3+x^4+x^5$
219	$x+x^6$
220	$x+x^2+x^6+x^7$

221	$1+x^4+x^6$
222	$1+x+x^4+x^5+x^6+x^7$
223	$x+x^2+x^3$
224	$x+x^4$
225	$x+x^2+x^4+x^5$
226	$x+x^3+x^4+x^6$
227	$x+x^2+x^3+x^5+x^6+x^7$
228	$1+x^3+x^5$
229	$1+x+x^3+x^4+x^5+x^6$
230	$1+x^2+x^3+x^7$
231	$x^2+x^3+x^7$
232	$1+x+x^2+x^3+x^7$
233	$x+x^3+x^7$
234	$1+x^2+x^7$
235	$x^2+x^4+x^7$
236	$1+x+x^2+x^5+x^7$
237	$x+x^4+x^5+x^6+x^7$
238	$1+x^2+x^3$
239	$1+x+x^2+x^4$
240	$1+x^3+x^4+x^5$
241	$1+x+x^3+x^6$
242	$1+x^2+x^3+x^4+x^6+x^7$
243	$x^2+x^3+x^4+x^5+x^6$
244	x^2+x^7
245	$1+x+x^2+x^4+x^7$
246	$x+x^5+x^7$
247	$1+x^2+x^3+x^4+x^5+x^6+x^7$
248	$x^2+x^3+x^4$
249	x^2+x^5
250	$x^2+x^3+x^5+x^6$
251	$x^2+x^4+x^5+x^7$
252	$1+x+x^2+x^6+x^7$
253	$x+x^4+x^6$
254	$x+x^2+x^4+x^5+x^6+x^7$
255	1
256	$1+x$

Apéndice B

$\boldsymbol{GF}(2^8)$

En esta sección se indica el método con el cual se generan los elementos del $GF(2^8)$ y además se listan los mismos, estos elementos son los utilizados en el proceso de cifrado y descifrado en el algoritmo AES, este GF es generado utilizando el polinomio $m(x)$ donde

$$m(x) = 1 + x + x^3 + x^4 + x^8$$

El elemento generador del GF es elemento es el $p = 1 + x$, el proceso para generar a los elemento de este campo es el siguiente:

1) Se incluye el elemento cero.
2) Se incluye el elemento generador p como primer elemento
3) Se toma al elemento anterior, se multiplica por p y se saca el módulo $m(x)$. Se incluye en los elementos de GF
4) Se repite el paso anterior hasta repetir a el elemento p y termina, note que este elemento no se incluye debido a que ya esta.

En otras palabras después de haber incluido el cero y p los siguientes elementos son potencias de p, p^2, p^3, etc., via módulo $m(x)$. Para ejemplificar este proceso se generan los primeros elementos:

$$\begin{aligned} p^2 \bmod(m(x)) &= p * p \bmod(m(x)) \\ &= (1+x) * (1+x) \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x + x + x^2 \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x^2 \end{aligned}$$

$$\begin{aligned} p^3 \bmod(m(x)) &= p^2 * p \bmod(m(x)) \\ &= (1+x^2) * (1+x) \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x^2 + x + x^3 \bmod(1 + x + x^3 + x^4 + x^8) \\ &= 1 + x + x^2 + x^3 \end{aligned}$$

A continuación se listan los elementos del $GF(2^8)$:

0	0
1	1+x
2	$1+x^2$
3	$1+x+x^2+x^3$
4	$1+x^4$
5	$1+x+x^4+x^5$
6	$1+x^2+x^4+x^6$
7	$1+x+x^2+x^3+x^4+x^5+x^6+x^7$
8	$x+x^3+x^4$
9	$x+x^2+x^3+x^5$
10	$x+x^4+x^5+x^6$
11	$x+x^2+x^4+x^7$
12	$1+x^5+x^7$
13	$x^3+x^4+x^5+x^6+x^7$
14	$1+x+x^4$
15	$1+x^2+x^4+x^5$
16	$1+x+x^2+x^3+x^4+x^6$
17	$1+x^5+x^6+x^7$
18	$x^3+x^4+x^5$
19	x^3+x^6
20	$x^3+x^4+x^6+x^7$
21	$1+x+x^4+x^5+x^6$
22	$1+x^2+x^4+x^7$
23	$x^2+x^5+x^7$
24	$1+x+x^2+x^4+x^5+x^6+x^7$
25	x
26	$x+x^2$
27	$x+x^3$
28	$x+x^2+x^3+x^4$
29	$x+x^5$
30	$x+x^2+x^5+x^6$
31	$x+x^3+x^5+x^7$
32	$1+x^2+x^5+x^6+x^7$
33	$x^2+x^4+x^5$
34	$x^2+x^3+x^4+x^6$
35	$x^2+x^5+x^6+x^7$
36	$1+x+x^2+x^4+x^5$
37	$1+x^3+x^4+x^6$
38	$1+x+x^3+x^5+x^6+x^7$
39	$x+x^2+x^5$
40	$x+x^3+x^5+x^6$
41	$x+x^2+x^3+x^4+x^5+x^7$
42	$1+x^3+x^4+x^6+x^7$
43	$x^4+x^5+x^6$

44	x^4+x^7
45	$1+x+x^3+x^5+x^7$
46	$x+x^2+x^5+x^6+x^7$
47	$1+x^4+x^5$
48	$1+x+x^4+x^6$
49	$1+x^2+x^4+x^5+x^6+x^7$
50	x^2
51	x^2+x^3
52	x^2+x^4
53	$x^2+x^3+x^4+x^5$
54	x^2+x^6
55	$x^2+x^3+x^6+x^7$
56	$1+x+x^2+x^3+x^6$
57	$1+x^4+x^6+x^7$
58	$x^3+x^5+x^6$
59	$x^3+x^4+x^5+x^7$
60	$1+x+x^4+x^6+x^7$
61	$x+x^2+x^3+x^5+x^6$
62	$x+x^4+x^5+x^7$
63	$1+x^2+x^3+x^6+x^7$
64	$x^2+x^3+x^6$
65	$x^2+x^4+x^6+x^7$
66	$1+x+x^2+x^5+x^6$
67	$1+x^3+x^5+x^7$
68	$x^5+x^6+x^7$
69	$1+x+x^3+x^4+x^5$
70	$1+x^2+x^3+x^6$
71	$1+x+x^2+x^4+x^6+x^7$
72	$x+x^5+x^6$
73	$x+x^2+x^5+x^7$
74	$1+x^4+x^5+x^6+x^7$
75	x^3
76	x^3+x^4
77	x^3+x^5
78	$x^3+x^4+x^5+x^6$
79	x^3+x^7
80	$1+x+x^7$
81	$x+x^2+x^3+x^4+x^7$
82	$1+x^3+x^4+x^5+x^7$
83	$x^4+x^6+x^7$
84	$1+x+x^3+x^5+x^6$
85	$1+x^2+x^3+x^4+x^5+x^7$
86	$x^2+x^3+x^4+x^6+x^7$
87	$1+x+x^2+x^3+x^4+x^5+x^6$

88	$1+x^7$
89	$x^3+x^4+x^7$
90	$1+x+x^4+x^5+x^7$
91	$x+x^2+x^3+x^6+x^7$
92	$1+x^3+x^6$
93	$1+x+x^3+x^4+x^6+x^7$
94	$x+x^2+x^4+x^5+x^6$
95	$x+x^3+x^4+x^7$
96	$1+x^2+x^4+x^5+x^7$
97	$x^2+x^6+x^7$
98	$1+x+x^2+x^4+x^6$
99	$1+x^3+x^4+x^5+x^6+x^7$
100	x^4
101	x^4+x^5
102	x^4+x^6
103	$x^4+x^5+x^6+x^7$
104	$1+x+x^3$
105	$1+x^2+x^3+x^4$
106	$1+x+x^2+x^5$
107	$1+x^3+x^5+x^6$
108	$1+x+x^3+x^4+x^5+x^7$
109	$x+x^2+x^4+x^6+x^7$
110	$1+x^5+x^6$
111	$1+x+x^5+x^7$
112	$x+x^2+x^3+x^4+x^5+x^6+x^7$
113	$1+x^3+x^4$
114	$1+x+x^3+x^5$
115	$1+x^2+x^3+x^4+x^5+x^6$
116	$1+x+x^2+x^7$
117	$x+x^4+x^7$
118	$1+x^2+x^3+x^5+x^7$
119	$x^2+x^3+x^5+x^6+x^7$
120	$1+x+x^2+x^3+x^5$
121	$1+x^4+x^5+x^6$
122	$1+x+x^4+x^7$
123	$x+x^2+x^3+x^5+x^7$
124	$1+x^3+x^5+x^6+x^7$
125	x^5
126	x^5+x^6
127	x^5+x^7
128	$1+x+x^3+x^4+x^5+x^6+x^7$
130	$x+x^3+x^4+x^5$
131	$x+x^2+x^3+x^6$
132	$x+x^4+x^6+x^7$

133	$1+x^2+x^3+x^5+x^6$
134	$1+x+x^2+x^4+x^5+x^7$
135	$x+x^6+x^7$
136	$1+x^2+x^3+x^4+x^6$
137	$1+x+x^2+x^5+x^6+x^7$
138	$x+x^4+x^5$
139	$x+x^2+x^4+x^6$
140	$x+x^3+x^4+x^5+x^6+x^7$
141	$1+x^2+x^4$
142	$1+x+x^2+x^3+x^4+x^5$
143	$1+x^6$
144	$1+x+x^6+x^7$
145	$x+x^2+x^3+x^4+x^6$
146	$x+x^5+x^6+x^7$
147	$1+x^2+x^3+x^4+x^5$
148	$1+x+x^2+x^6$
149	$1+x^3+x^6+x^7$
150	x^6
151	x^6+x^7
152	$1+x+x^3+x^4+x^6$
153	$1+x^2+x^3+x^5+x^6+x^7$
154	$x^2+x^3+x^5$
155	$x^2+x^4+x^5+x^6$
156	$x^2+x^3+x^4+x^7$
157	$1+x+x^2+x^3+x^4+x^5+x^7$
158	$x+x^3+x^4+x^6+x^7$
159	$1+x^2+x^4+x^5+x^6$
160	$1+x+x^2+x^3+x^4+x^7$
161	$x+x^3+x^4+x^5+x^7$
162	$1+x^2+x^4+x^6+x^7$
163	$x^2+x^5+x^6$
164	$x^2+x^3+x^5+x^7$
165	$1+x+x^2+x^3+x^5+x^6+x^7$
166	$x+x^3+x^5$
167	$x+x^2+x^3+x^4+x^5+x^6$
168	$x+x^7$
169	$1+x^2+x^3+x^4+x^7$
170	$x^2+x^3+x^4+x^5+x^7$
171	$1+x+x^2+x^3+x^4+x^6+x^7$
172	$x+x^3+x^4+x^5+x^6$
173	$x+x^2+x^3+x^7$
174	$1+x^3+x^7$
175	x^7
176	$1+x+x^3+x^4+x^7$

177	$x+x^2+x^4+x^5+x^7$
178	$1+x^6+x^7$
179	$x^3+x^4+x^6$
180	$x^3+x^5+x^6+x^7$
181	$1+x+x^5$
182	$1+x^2+x^5+x^6$
183	$1+x+x^2+x^3+x^5+x^7$
184	$x+x^3+x^5+x^6+x^7$
185	$1+x^2+x^5$
186	$1+x+x^2+x^3+x^5+x^6$
187	$1+x^4+x^5+x^7$
188	$x^3+x^6+x^7$
189	$1+x+x^6$
190	$1+x^2+x^6+x^7$
191	$x^2+x^4+x^6$
192	$x^2+x^3+x^4+x^5+x^6+x^7$
193	$1+x+x^2+x^3+x^4$
194	$1+x^5$
195	$1+x+x^5+x^6$
196	$1+x^2+x^5+x^7$
197	$x^2+x^4+x^5+x^6+x^7$
198	$1+x+x^2$
199	$1+x^3$
200	$1+x+x^3+x^4$
201	$1+x^2+x^3+x^5$
202	$1+x+x^2+x^4+x^5+x^6$
203	$1+x^3+x^4+x^7$
204	$x^4+x^5+x^7$
205	$1+x+x^3+x^6+x^7$
206	$x+x^2+x^6$
207	$x+x^3+x^6+x^7$
208	$1+x^2+x^6$
209	$1+x+x^2+x^3+x^6+x^7$
210	$x+x^3+x^6$
211	$x+x^2+x^3+x^4+x^6+x^7$
212	$1+x^3+x^4+x^5+x^6$
213	$1+x+x^3+x^7$
214	$x+x^2+x^7$
215	$1+x^4+x^7$
216	$x^3+x^5+x^7$
217	$1+x+x^5+x^6+x^7$
218	$x+x^2+x^3+x^4+x^5$
219	$x+x^6$
220	$x+x^2+x^6+x^7$

221	$1+x^4+x^6$
222	$1+x+x^4+x^5+x^6+x^7$
223	$x+x^2+x^3$
224	$x+x^4$
225	$x+x^2+x^4+x^5$
226	$x+x^3+x^4+x^6$
227	$x+x^2+x^3+x^5+x^6+x^7$
228	$1+x^3+x^5$
229	$1+x+x^3+x^4+x^5+x^6$
230	$1+x^2+x^3+x^7$
231	$x^2+x^3+x^7$
232	$1+x+x^2+x^3+x^7$
233	$x+x^3+x^7$
234	$1+x^2+x^7$
235	$x^2+x^4+x^7$
236	$1+x+x^2+x^5+x^7$
237	$x+x^4+x^5+x^6+x^7$
238	$1+x^2+x^3$
239	$1+x+x^2+x^4$
240	$1+x^3+x^4+x^5$
241	$1+x+x^3+x^6$
242	$1+x^2+x^3+x^4+x^6+x^7$
243	$x^2+x^3+x^4+x^5+x^6$
244	x^2+x^7
245	$1+x+x^2+x^4+x^7$
246	$x+x^5+x^7$
247	$1+x^2+x^3+x^4+x^5+x^6+x^7$
248	$x^2+x^3+x^4$
249	x^2+x^5
250	$x^2+x^3+x^5+x^6$
251	$x^2+x^4+x^5+x^7$
252	$1+x+x^2+x^6+x^7$
253	$x+x^4+x^6$
254	$x+x^2+x^4+x^5+x^6+x^7$
255	1
256	$1+x$

Apéndice C

Implementación en ensamblador del AES

A continuación se muestra la implementación del algoritmo de cifrado AES.

```
.NOLIST
.INCLUDE "8515def.inc"
.LIST
.DSEG
.CSEG
.ORG 0000
    ; "estado", como se almacenan en memoria segun su dir.
    ;+--------------+
    ;|96|100|104|108|
    ;|97|101|105|109|
    ;|98|102|106|110|
    ;|99|103|107|111|
    ;+--------------+
.EQU direccionestado = 0x0060 ; es usan 16 bytes
    ; "direccionkey", como se almacenan en memoria según su dir.
    ;   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32
    ;+-----------------------------------------------------------------------------------------------------------------------------------+
    ;|112|113|114|115|116|117|118|119|120|121|122|123|124|125|126|127|128|129|130|131|132|133|134|135|136|137|138|139|140|141|142|143|
    ;+-----------------------------------------------------------------------------------------------------------------------------------+
    ;+-------------------------------+
    ;|112|116|120|124|128|132|136|140|
    ;|113|117|121|125|129|133|137|141|
    ;|114|118|122|126|130|134|138|142|
    ;|115|119|123|127|131|135|139|143|
    ;+-------------------------------+
.EQU direccionkey = 0x0070 ; se usan 32 bytes
.EQU Nk = $08
.EQU Nb = $04
.EQU Nr = $0E;14 en decimal

/**
    El registro r25 se considera como contador de ronda en las funciones
    de cifra y descifra
*/

;Inicializa el stack pointer
    ldi r16,LOW(RAMEND)
    out SPL,r16
    ldi r16,HIGH(RAMEND)
    out SPH,r16
;Fin de inicialización del stack pointer
    rjmp    main

aes_sbox:
 .DB  0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76
 .DB  0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0
 .DB  0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15
 .DB  0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75
 .DB  0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84
 .DB  0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF
 .DB  0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8
 .DB  0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2
 .DB  0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73
 .DB  0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB
 .DB  0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79
 .DB  0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08
 .DB  0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A
 .DB  0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E
 .DB  0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF
```

```
 .DB  0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16

Rcon:
 .DB  0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80


/*  macro xtime(y)
    Toma del registro r15 y lo multiplica por x deacuerdo
    al polinomios usado en el fips-196 el resultado lo deja
    en el mismo registro.
*/
.MACRO xtime ; se usan 7 ciclos por llamada
; define XTIME(x) ( ( x <<  1 ) ^ ( ( x & 0x80 ) ? 0x1B : 0x00 ) )
    LDI r20, 0x1b
    LSL r16 ;  r16 <<1

    BRCC xtimesi
    EOR r16, r20 ; si no r16^0x1b
xtimesi:

.ENDMACRO


/*  macro SubWord @0,@1,@2,@3
    Función usada en la rutina de expansión de llave
    toma cuatro bits una palabra, aplica la S-box y
    regresa una palabra en los mismos registros.
*/
.MACRO SubWord

    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)

    add ZL,@0
    adc ZH, r1
    LPM
    MOV @0, r0

    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)
    add ZL,@1
    adc ZH, r1
    LPM
    MOV @1, r0

    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)
    add ZL,@2
    adc ZH, r1
    LPM
    MOV @2, r0

    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)
    add ZL,@3
    adc ZH, r1
    LPM
    MOV @3, r0

.ENDMACRO

main:
    RCALL cifra
;    RCALL descifra
fin: nop
    rjmp fin

cifra:
    PUSH r31
    PUSH r30
    PUSH r29
    PUSH r28
    PUSH r27
    PUSH r26
    PUSH r25
    PUSH r20
    PUSH r19
    PUSH r18
    PUSH r17
    PUSH r16
    PUSH r15
    PUSH r14
    PUSH r13
    PUSH r12
    PUSH r11
    PUSH r10
    PUSH r9
    PUSH r8
    PUSH r7
    PUSH r6
```

```
    PUSH r5
    PUSH r4
    PUSH r3
    PUSH r2
    PUSH r1
    PUSH r0
    ; inicializa el contador de ronda r25
    CLR r25
    CLR r2 ; en ves del valor en direccionRconi
    CLR r1 ; siembre vale 0

cifra_for:
        RCALL ShifRows_SubBytes_Addkey
        RCALL MixColumns
        INC r25

        RCALL ShifRows_SubBytes_Addkey
        RCALL KeyExpansion

        RCALL MixColumns
        INC r25

        CPI r25, Nr-2
    BRNE cifra_for  ; del for

    RCALL ShifRows_SubBytes_Addkey
    RCALL MixColumns
    INC r25

    RCALL ShifRows_SubBytes_Addkey
    RCALL KeyExpansion

    INC r25
    RCALL AddRoundKey

    POP r0
    POP r1
    POP r2
    POP r3
    POP r4
    POP r5
    POP r6
    POP r7
    POP r8
    POP r9
    POP r10
    POP r11
    POP r12
    POP r13
    POP r14
    POP r15
    POP r16
    POP r17
    POP r18
    POP r19
    POP r20
    POP r25
    POP r26
    POP r27
    POP r28
    POP r29
    POP r30
    POP r31
    RET

/*  función AddRoundKey()
    Es añadida la llave de ronda al estado por la aplicación de XOR
*/
AddRoundKey:

    LDI XH, HIGH(direccionestado)
    LDI XL, LOW(direccionestado)
    LDI YH, HIGH(direccionkey)
    LDI YL, LOW(direccionkey)

    MOV r16,r25
    ANDI r16,$01
    BREQ AddRoundKey_a
        ADIW YL:YH,$10
    AddRoundKey_a:

    LDI r16, Nb*Nb

AddRoundKey_while:

    LD r0,X ;1
    LD r3,Y+ ;2
    EOR r0,r3
    ST X+,r0
```

```
    DEC r16
    BRNE AddRoundKey_while

    RET

/*  función MixColumns()
    Opera sobre el estado, columna por columna, tratando
    cada columna como un polinomio de grado cuatro sobre
    GF(2^8) modulo x^4 -1
*/
MixColumns:
;   1 5  9 13
;   2 6 10 14
;   3 7 11 15
;   4 8 12 16
    LDI XH, HIGH(direccionestado)
    LDI XL, LOW(direccionestado)

    LDI r19,$04
MixColumns_for:

    LD r3,X+;r12, X+
    LD r4,X+;r13, X+
    LD r5,X+;r14, X+
    LD r6,X ;r15, X
    SBIW XL:XH, $03

    MOV r7, r3
    EOR r7, r4

    MOV r8, r5
    EOR r8, r6

    MOV r16, r7

    xtime

    EOR r16, r4
    EOR r16, r8

    ST X+,r16

    MOV r16, r4
    EOR r16, r5

    xtime

    EOR r16, r8
    EOR r16, r3

    ST X+,r16

    MOV r16, r8

    xtime

    EOR r16, r7
    EOR r16, r6
    ST X+,r16

    MOV r16, r3
    EOR r16, r6
    xtime
    EOR r16, r7
    EOR r16, r5
    ST X+,r16

    DEC r19
    BRNE MixColumns_for

    RET


/*  función KeyExpansion()
    Genera el esquema de llave mediante bloques de 32 bits
*/

KeyExpansion:
;       i                           i-1
    ;|112|116|120|124| 128|132|136|140|
    ;|113|117|121|125| 129|133|137|141|
    ;|114|118|122|126| 130|134|138|142|
    ;|115|119|123|127| 131|135|139|143|
;    i-Nk=0

    LDI YH, HIGH(direccionkey+$1C)
    LDI YL, LOW(direccionkey+$1C)

    LD r15, Y+
    LD r12, Y+
```

```
LD r13, Y+
LD r14, Y+

LDI YH, HIGH(direccionkey) ; w[i-Nk] =w[0]
LDI YL, LOW(direccionkey)

LD r16, Y+
LD r17, Y+
LD r18, Y+
LD r19, Y+

SubWord r12, r13, r14, r15

LDI ZH, HIGH(2*Rcon)
LDI ZL, LOW(2*Rcon)

ADD ZL, r2
INC r2 ; cambios para incremento de i
LPM

EOR r16,r12
EOR r16,r0

EOR r17,r13
EOR r18,r14
EOR r19,r15

SBIW YL:YH,$04

ST Y+,r16 ;i
ST Y+,r17
ST Y+,r18
ST Y+,r19

LD r12, Y+
LD r13, Y+
LD r14, Y+
LD r15, Y+

EOR r16,r12
EOR r17,r13
EOR r18,r14
EOR r19,r15

SBIW YL:YH,$04

ST Y+,r16 ;i+1
ST Y+,r17
ST Y+,r18
ST Y+,r19

LD r12, Y+
LD r13, Y+
LD r14, Y+
LD r15, Y+

EOR r16,r12
EOR r17,r13
EOR r18,r14
EOR r19,r15

SBIW YL:YH,$04

ST Y+,r16 ;i+2
ST Y+,r17
ST Y+,r18
ST Y+,r19

LD r12, Y+
LD r13, Y+
LD r14, Y+
LD r15, Y+

EOR r16,r12
EOR r17,r13
EOR r18,r14
EOR r19,r15

SBIW YL:YH,$04

ST Y+,r16 ;i+3
ST Y+,r17
ST Y+,r18
ST Y+,r19

LD r12, Y+
LD r13, Y+
LD r14, Y+
LD r15, Y+
```

```
    SubWord r16, r17, r18, r19

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    SBIW YL:YH,$04

    ST Y+,r16 ;i+4
    ST Y+,r17
    ST Y+,r18
    ST Y+,r19

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    SBIW YL:YH,$04

    ST Y+,r16 ;i+5
    ST Y+,r17
    ST Y+,r18
    ST Y+,r19

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    SBIW YL:YH,$04

    ST Y+,r16 ;i+6
    ST Y+,r17
    ST Y+,r18
    ST Y+,r19

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    SBIW YL:YH,$04

    ST Y+,r16 ;i+7
    ST Y+,r17
    ST Y+,r18
    ST Y+,r19

    RET

/*  función ShifRows_SubBytes()
    Esta operación junta ShifRows_SubBytes
;   1 5  9 13
;   2 6 10 14
;   3 7 11 15
;   4 8 12 16
    a
;   1   5  9 13
;   6  10 14  2
;   11 15  3  7
;   16  4  8 12

ShifRows_SubBytes:

    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)
    LDI XH, HIGH(direccionestado)
    LDI XL, LOW(direccionestado)

    LD r16, X  ;1
    add ZL, r16
    adc ZH, r1
    LPM
```

```
ST X+,r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r9, X+  ;2
add ZL, r9
adc ZH, r1
LPM
MOV r9, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r3, X+  ;3
add ZL, r3
adc ZH, r1
LPM
MOV r3, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r4, X+  ;4
add ZL, r4
adc ZH, r1
LPM
MOV r4, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r16, X  ;5
add ZL, r16
adc ZH, r1
LPM
ST X+,r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r6, X+  ;6
add ZL, r6
adc ZH, r1
LPM
MOV r6, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r7, X+  ;7
add ZL, r7
adc ZH, r1
LPM
MOV r7, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r8, X+  ;8
add ZL, r8
adc ZH, r1
LPM
MOV r8, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r16, X  ;9
add ZL, r16
adc ZH, r1
LPM
ST X+,r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r10, X+ ;10
add ZL, r10
adc ZH, r1
LPM
MOV r10, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r11, X+  ;11
add ZL, r11
adc ZH, r1
LPM
MOV r11, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r12, X+  ;12
add ZL, r12
adc ZH, r1
LPM
MOV r12, r0
LDI ZH, HIGH(2*aes_sbox)
```

```
    LDI ZL, LOW(2*aes_sbox)

    LD r16, X  ;13
    add ZL, r16
    adc ZH, r1
    LPM
    ST X+,r0
    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)

    LD r14, X+  ;14
    add ZL, r14
    adc ZH, r1
    LPM
    MOV r14, r0
    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)

    LD r15, X+  ;15
    add ZL, r15
    adc ZH, r1
    LPM
    MOV r15, r0
    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)

    LD r16, X  ;16
    add ZL, r16
    adc ZH, r1
    LPM
    MOV r16, r0

    ST X,r12
    ST -X,r7
    ST -X,r9
    SBIW XL:XH, $02
    ST X,r8
    ST -X,r3
    ST -X,r14
    SBIW XL:XH, $02
    ST X,r4
    ST -X,r15
    ST -X,r10
    SBIW XL:XH, $02
    ST X,r16
    ST -X,r11
    ST -X,r6

    RET

/*  función ShifRows_SubBytes_Addkey()
    Esta operación junta Addkey ShifRows SubBytes

;   1 5  9 13
;   2 6 10 14
;   3 7 11 15
;   4 8 12 16

    a

;   1   5  9 13
;   6  10 14  2
;   11 15  3  7
;   16  4  8 12

*/
ShifRows_SubBytes_Addkey:

    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)
    LDI YH, HIGH(direccionkey)
    LDI YL, LOW(direccionkey)
    LDI XH, HIGH(direccionestado)
    LDI XL, LOW(direccionestado)

    MOV r16,r25
    ANDI r16,$01
    BREQ AddRoundKey_as
        ADIW YL:YH,$10
    AddRoundKey_as:

    LD r16, X  ;1
    LD r13,Y+
    EOR r16,r13
    add ZL, r16
    adc ZH, r1
    LPM
    ST X+,r0
    LDI ZH, HIGH(2*aes_sbox)
    LDI ZL, LOW(2*aes_sbox)
```

```
LD r9, X+  ;2
LD r13,Y+
EOR r9,r13
add ZL, r9
adc ZH, r1
LPM
MOV r9, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r3, X+  ;3
LD r13,Y+
EOR r3,r13
add ZL, r3
adc ZH, r1
LPM
MOV r3, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r4, X+  ;4
LD r13,Y+
EOR r4,r13
add ZL, r4
adc ZH, r1
LPM
MOV r4, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r16, X  ;5
LD r13,Y+
EOR r16,r13
add ZL, r16
adc ZH, r1
LPM
ST X+,r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r6, X+  ;6
LD r13,Y+
EOR r6,r13
add ZL, r6
adc ZH, r1
LPM
MOV r6, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r7, X+  ;7
LD r13,Y+
EOR r7,r13
add ZL, r7
adc ZH, r1
LPM
MOV r7, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r8, X+  ;8
LD r13,Y+
EOR r8,r13
add ZL, r8
adc ZH, r1
LPM
MOV r8, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r16, X  ;9
LD r13,Y+
EOR r16,r13
add ZL, r16
adc ZH, r1
LPM
ST X+,r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)

LD r10, X+ ;10
LD r13,Y+
EOR r10,r13
add ZL, r10
adc ZH, r1
LPM
MOV r10, r0
LDI ZH, HIGH(2*aes_sbox)
LDI ZL, LOW(2*aes_sbox)
```

```
   LD r11, X+  ;11
   LD r13,Y+
   EOR r11,r13
   add ZL, r11
   adc ZH, r1
   LPM
   MOV r11, r0
   LDI ZH, HIGH(2*aes_sbox)
   LDI ZL, LOW(2*aes_sbox)

   LD r12, X+  ;12
   LD r13,Y+
   EOR r12,r13
   add ZL, r12
   adc ZH, r1
   LPM
   MOV r12, r0
   LDI ZH, HIGH(2*aes_sbox)
   LDI ZL, LOW(2*aes_sbox)

   LD r16, X  ;13
   LD r13,Y+
   EOR r16,r13
   add ZL, r16
   adc ZH, r1
   LPM
   ST X+,r0
   LDI ZH, HIGH(2*aes_sbox)
   LDI ZL, LOW(2*aes_sbox)

   LD r14, X+  ;14
   LD r13,Y+
   EOR r14,r13
   add ZL, r14
   adc ZH, r1
   LPM
   MOV r14, r0
   LDI ZH, HIGH(2*aes_sbox)
   LDI ZL, LOW(2*aes_sbox)

   LD r15, X+  ;15
   LD r13,Y+
   EOR r15,r13
   add ZL, r15
   adc ZH, r1
   LPM
   MOV r15, r0
   LDI ZH, HIGH(2*aes_sbox)
   LDI ZL, LOW(2*aes_sbox)

   LD r16, X  ;16
   LD r13,Y+
   EOR r16,r13
   add ZL, r16
   adc ZH, r1
   LPM
   MOV r16, r0

   ST X,r12
   ST -X,r7
   ST -X,r9
   SBIW XL:XH, $02
   ST X,r8
   ST -X,r3
   ST -X,r14
   SBIW XL:XH, $02
   ST X,r4
   ST -X,r15
   ST -X,r10
   SBIW XL:XH, $02
   ST X,r16
   ST -X,r11
   ST -X,r6
   RET
```

Para obtener el código de descifrado solo se debe agregar el siguiente código al del cifrado del AES.

```
aes_invsbox:
  .DB  0x52,0x09,0x6A,0xD5,0x30,0x36,0xA5,0x38,0xBF,0x40,0xA3,0x9E,0x81,0xF3,0xD7,0xFB
  .DB  0x7C,0xE3,0x39,0x82,0x9B,0x2F,0xFF,0x87,0x34,0x8E,0x43,0x44,0xC4,0xDE,0xE9,0xCB
  .DB  0x54,0x7B,0x94,0x32,0xA6,0xC2,0x23,0x3D,0xEE,0x4C,0x95,0x0B,0x42,0xFA,0xC3,0x4E
  .DB  0x08,0x2E,0xA1,0x66,0x28,0xD9,0x24,0xB2,0x76,0x5B,0xA2,0x49,0x6D,0x8B,0xD1,0x25
  .DB  0x72,0xF8,0xF6,0x64,0x86,0x68,0x98,0x16,0xD4,0xA4,0x5C,0xCC,0x5D,0x65,0xB6,0x92
  .DB  0x6C,0x70,0x48,0x50,0xFD,0xED,0xB9,0xDA,0x5E,0x15,0x46,0x57,0xA7,0x8D,0x9D,0x84
  .DB  0x90,0xD8,0xAB,0x00,0x8C,0xBC,0xD3,0x0A,0xF7,0xE4,0x58,0x05,0xB8,0xB3,0x45,0x06
  .DB  0xD0,0x2C,0x1E,0x8F,0xCA,0x3F,0x0F,0x02,0xC1,0xAF,0xBD,0x03,0x01,0x13,0x8A,0x6B
```

```
 .DB  0x3A,0x91,0x11,0x41,0x4F,0x67,0xDC,0xEA,0x97,0xF2,0xCF,0xCE,0xF0,0xB4,0xE6,0x73
 .DB  0x96,0xAC,0x74,0x22,0xE7,0xAD,0x35,0x85,0xE2,0xF9,0x37,0xE8,0x1C,0x75,0xDF,0x6E
 .DB  0x47,0xF1,0x1A,0x71,0x1D,0x29,0xC5,0x89,0x6F,0xB7,0x62,0x0E,0xAA,0x18,0xBE,0x1B
 .DB  0xFC,0x56,0x3E,0x4B,0xC6,0xD2,0x79,0x20,0x9A,0xDB,0xC0,0xFE,0x78,0xCD,0x5A,0xF4
 .DB  0x1F,0xDD,0xA8,0x33,0x88,0x07,0xC7,0x31,0xB1,0x12,0x10,0x59,0x27,0x80,0xEC,0x5F
 .DB  0x60,0x51,0x7F,0xA9,0x19,0xB5,0x4A,0x0D,0x2D,0xE5,0x7A,0x9F,0x93,0xC9,0x9C,0xEF
 .DB  0xA0,0xE0,0x3B,0x4D,0xAE,0x2A,0xF5,0xB0,0xC8,0xEB,0xBB,0x3C,0x83,0x53,0x99,0x61
 .DB  0x17,0x2B,0x04,0x7E,0xBA,0x77,0xD6,0x26,0xE1,0x69,0x14,0x63,0x55,0x21,0x0C,0x7D

descifra:
    ; inicializa el contador de ronda r25
    PUSH r31
    PUSH r30
    PUSH r29
    PUSH r28
    PUSH r27
    PUSH r26
    PUSH r25
    PUSH r20
    PUSH r19
    PUSH r18
    PUSH r17
    PUSH r16
    PUSH r15
    PUSH r14
    PUSH r13
    PUSH r12
    PUSH r11
    PUSH r10
    PUSH r9
    PUSH r8
    PUSH r7
    PUSH r6
    PUSH r5
    PUSH r4
    PUSH r3
    PUSH r2
    PUSH r1
    PUSH r0

    RCALL IniciaInvKeyExpansion

    RCALL AddRoundKey
    DEC r25
    RCALL InvKeyExpansion

    RCALL InvShifRows_InvSubBytes

    RCALL AddRoundKey
    DEC r25

    RCALL InvMixColumns
    RCALL InvShifRows_InvSubBytes
descifra_for:

        RCALL AddRoundKey
        DEC r25
        RCALL InvMixColumns

        RCALL InvKeyExpansion
        RCALL InvShifRows_InvSubBytes


        RCALL AddRoundKey
        DEC r25
        RCALL InvMixColumns
        RCALL InvShifRows_InvSubBytes
        CPI r25, 0
    BRNE descifra_for   ; del for

    rcall AddRoundKey

    POP r0
    POP r1
    POP r2
    POP r3
    POP r4
    POP r5
    POP r6
    POP r7
    POP r8
    POP r9
    POP r10
    POP r11
    POP r12
    POP r13
    POP r14
    POP r15
    POP r16
    POP r17
```

```
    POP r18
    POP r19
    POP r20
    POP r25
    POP r26
    POP r27
    POP r28
    POP r29
    POP r30
    POP r31
    RET

InvKeyExpansion:

    LDI YH, HIGH(direccionkey+$18)
    LDI YL, LOW(direccionkey+$18)

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    LD r16, Y+
    LD r17, Y+
    LD r18, Y+
    LD r19, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    ST -Y,r19 ;i
    ST -Y,r18
    ST -Y,r17
    ST -Y,r16

    SBIW YL:YH,$08

    LD r16, Y+
    LD r17, Y+
    LD r18, Y+
    LD r19, Y+

    EOR r12,r16
    EOR r13,r17
    EOR r14,r18
    EOR r15,r19

    ST Y+,r12 ;i+1
    ST Y+,r13
    ST Y+,r14
    ST Y+,r15

    SBIW YL:YH,$0C

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    ST Y+,r16
    ST Y+,r17
    ST Y+,r18
    ST Y+,r19

    SBIW YL:YH,$0C

    LD r16, Y+
    LD r17, Y+
    LD r18, Y+
    LD r19, Y+

    SubWord r16, r17, r18, r19

    EOR r12,r16
    EOR r13,r17
    EOR r14,r18
    EOR r15,r19

    ST Y+,r12 ;i+1
    ST Y+,r13
    ST Y+,r14
    ST Y+,r15
```

```
    SBIW YL:YH,$0C

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    LD r16, Y+
    LD r17, Y+
    LD r18, Y+
    LD r19, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    ST -Y,r19 ;i
    ST -Y,r18
    ST -Y,r17
    ST -Y,r16

    SBIW YL:YH,$08

    LD r16, Y+
    LD r17, Y+
    LD r18, Y+
    LD r19, Y+

    EOR r12,r16
    EOR r13,r17
    EOR r14,r18
    EOR r15,r19

    ST Y+,r12 ;i+1
    ST Y+,r13
    ST Y+,r14
    ST Y+,r15

    SBIW YL:YH,$0C

    LD r12, Y+
    LD r13, Y+
    LD r14, Y+
    LD r15, Y+

    EOR r16,r12
    EOR r17,r13
    EOR r18,r14
    EOR r19,r15

    ST Y+,r16
    ST Y+,r17
    ST Y+,r18
    ST Y+,r19

    LDI YH, HIGH(direccionkey+$1C)
    LDI YL, LOW(direccionkey+$1C)

    LD r19, Y+
    LD r16, Y+
    LD r17, Y+
    LD r18, Y+

    SubWord r16, r17, r18, r19

    LDI ZH, HIGH(2*Rcon)
    LDI ZL, LOW(2*Rcon)

    DEC r2
    ADD ZL, r2
    LPM

    EOR r12,r16
    EOR r12,r0
    EOR r13,r17
    EOR r14,r18
    EOR r15,r19

    LDI YH, HIGH(direccionkey)
    LDI YL, LOW(direccionkey)

    ST Y+,r12
    ST Y+,r13
    ST Y+,r14
    ST Y+,r15

    RET

IniciaInvKeyExpansion:
```

```
    CLR r2
    LDI r25,$0E
    RCALL KeyExpansion
    RCALL KeyExpansion
    RCALL KeyExpansion
    RCALL KeyExpansion
    RCALL KeyExpansion
    RCALL KeyExpansion
    RCALL KeyExpansion
    RET

/*  función InvShifRows_InvSubBytes()
    Esta operación junta InvShifRows InvSubBytes

*/
/*
InvShifRows_InvSubBytes:

    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)
    LDI XH, HIGH(direccionestado)
    LDI XL, LOW(direccionestado)

    LD r16, X  ;1
    add ZL, r16
    adc ZH, r1
    LPM
    ST X+,r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r9, X+  ;2
    add ZL, r9
    adc ZH, r1
    LPM
    MOV r9, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r3, X+  ;3
    add ZL, r3
    adc ZH, r1
    LPM
    MOV r3, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r4, X+  ;4
    add ZL, r4
    adc ZH, r1
    LPM
    MOV r4, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r16, X  ;5
    add ZL, r16
    adc ZH, r1
    LPM
    ST X+,r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r6, X+  ;6
    add ZL, r6
    adc ZH, r1
    LPM
    MOV r6, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r7, X+  ;7
    add ZL, r7
    adc ZH, r1
    LPM
    MOV r7, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r8, X+  ;8
    add ZL, r8
    adc ZH, r1
    LPM
    MOV r8, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r16, X  ;9
    add ZL, r16
    adc ZH, r1
```

```
    LPM
    ST X+,r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r10, X+ ;10
    add ZL, r10
    adc ZH, r1
    LPM
    MOV r10, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r11, X+  ;11
    add ZL, r11
    adc ZH, r1
    LPM
    MOV r11, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r12, X+  ;12
    add ZL, r12
    adc ZH, r1
    LPM
    MOV r12, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r16, X  ;13
    add ZL, r16
    adc ZH, r1
    LPM
    ST X+,r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r14, X+  ;14
    add ZL, r14
    adc ZH, r1
    LPM
    MOV r14, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r15, X+  ;15
    add ZL, r15
    adc ZH, r1
    LPM
    MOV r15, r0
    LDI ZH, HIGH(2*aes_invsbox)
    LDI ZL, LOW(2*aes_invsbox)

    LD r16, X  ;16
    add ZL, r16
    adc ZH, r1
    LPM
    MOV r16, r0

    ST X,r4
    ST -X,r7
    ST -X,r10
    SBIW XL:XH, $02
    ST X,r16
    ST -X,r3
    ST -X,r6
    SBIW XL:XH, $02
    ST X,r12
    ST -X,r15
    ST -X,r9
    SBIW XL:XH, $02
    ST X,r8
    ST -X,r11
    ST -X,r14

    RET
/*
    InvMixComns
*//*
InvMixColumns:

    LDI XH, HIGH(direccionestado)
    LDI XL, LOW(direccionestado)

    LDI r19,$04
InvMixColumns_for:

;   aux  = a0 ^ a1 ^ a2 ^ a3;
;   aux1 = xtime(a0 ^ a2);
;   aux2 = xtime(a1 ^ a3);
```

```
;   aux3 = xtime( xtime( aux1 ^ aux2 ) ^ aux;
;   d0   = xtime(a0 ^ a1 ^ aux1);
;   d1   = xtime(a1 ^ a2 ^ aux2);
;   d2   = xtime(a2 ^ a3 ^ aux1);
;   a'0   ^= d0 ^ aux3;
;   a'1   ^= d1 ^ aux3;
;   a'2   ^= d2 ^ aux3;
;   a'3    = a'0 ^ a'1 ^ a'2 ^ aux;
;
    LD r3,X+;a0
    LD r4,X+;a1
    LD r5,X+;a2
    LD r6,X ;a3
    SBIW XL:XH, $03

    MOV r7, r3
    EOR r7, r4
    EOR r7, r5
    EOR r7, r6

    MOV r16, r3
    EOR r16, r5
    xtime
    MOV r8, r16

    MOV r16, r4
    EOR r16, r6
    xtime
    MOV r9, r16

    MOV r16, r8
    EOR r16, r9
    xtime
    xtime
    EOR r16, r7
    MOV r10, r16

    rjmp InvMixColumns_for2
InvMixColumns_for1:
    rjmp InvMixColumns_for
InvMixColumns_for2:

    MOV r16, r3
    EOR r16, r4
    EOR r16, r8
    xtime
    MOV r11, r16

    MOV r16, r4
    EOR r16, r5
    EOR r16, r9
    xtime
    MOV r12, r16

    MOV r16, r5
    EOR r16, r6
    EOR r16, r8
    xtime
    MOV r13, r16

    EOR r3, r11
    EOR r3, r10

    EOR r4, r12
    EOR r4, r10

    EOR r5, r13
    EOR r5, r10

    EOR r7, r3
    EOR r7, r4
    EOR r7, r5

    ST X+,r3
    ST X+,r4
    ST X+,r5
    ST X+,r7

    DEC r19
    BRNE InvMixColumns_for1

    RET
```