Universidad Nacional Autónoma de México

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

ENCRIPTADO DE DATOS APLICANDO EL ALGORITMO AES PARA SU TRANSMISIÓN

T E S I S

QUE PARA OBTENER AL GRADO DE MAESTRÍA EN INGENIERÍA ELÉCTRICA, Campo: Telecomunicaciones.

P R E S E N T A:

Luis Enrique Aranda Melo

DIRIGIDA POR:
Dr. Carlos Rivera Rivera

MÉXICO, D.F.
Mayo 2006

Agradecimientos:

A mis padres, Hermanos, Maestros,
Amigos y sobre todo a mi Facultad (FI)
Que me dio las herramientas para ser lo que
Hoy soy

Gracias a todos.

Dedico este trabajo a todos los seres que siempre han estado a mi lado ya que sin su apoyo no hubiera terminado.

Gracias

Quien le ponga alas a sus sueños volara.

Índice general

1. Introducción **1**
- 1.1. Antecedentes . . . 1
- 1.2. Cifrado asimétrico y simétrico . . . 2
  - 1.2.1. Cifrado de flujo . . . 3
  - 1.2.2. Cifrado de bloque . . . 4
- 1.3. Cifrado AES . . . 6
- 1.4. Planteamiento del Problema . . . 7

2. Descripción del algoritmo *AES-Rijndael* **9**
- 2.1. Introducción al Algoritmo *AES-Rijndael* . . . 9
- 2.2. Función Adición de la clave (*AddRoundKey*). . . . 13
- 2.3. Sustitución (*ByteSub*). . . . 13
- 2.4. Desplazamiento de filas (*ShiftRow*) . . . 16
- 2.5. Multiplicación de columnas (*MixColumns*). . . . 19
- 2.6. Generación de las subclaves (*Key Schedule).* . . . 23
- 2.7. Numero de rondas. . . . 30

3. Implementación del algoritmo AES. **33**
- 3.1. Cifrado. . . . 36
- 3.2. Función Adición de la clave (*AddRoundKey*). . . . 37
- 3.3. Sustitución (*SubByte*). . . . 38
- 3.4. Desplazamiento de filas (*ShiftRow*). . . . 39
- 3.5. Multiplicación de columnas (*Mixcolumns*). . . . 40
  - 3.5.1. Multiplicación *mul*. . . . 40
- 3.6. Generación de las subclaves (*Key Expansion*). . . . 41
  - 3.6.1. Función de expansión de la clave (*Round Key Selection)* . . . 44
- 3.7. Descifrado . . . 44
  - 3.7.1. Propiedades del Descifrado. . . . 45
- 3.8. Cambiar el orden de las rondas . . . 47
- 3.9. Modificación de la longitud de la clave . . . 49
- 3.10. Cambio en las constante del algoritmo . . . 50

4. Programación de la implementación del algoritmo *AES-Rijndael.* 53
4.1. Diseño de la aplicación. 54
4.1.1. Propiedades algebraicas 56
4.2. Diagrama secuencial del cifrado *AES-Rijndael* 57
4.3. Resistencia al criptoanálisis que posee el Algoritmo *AES-Rijndael* . . 58
4.3.1. Resistencia de la función sustitución (*SubByte*) del algoritmo . 59
4.3.2. Resistencia de la función desplazamiento de filas (*ShitfRow*). . 60
4.3.3. Resistencia de la función multiplicación de columnas (*MaxColumn*). 60
4.3.4. Expansión de la clave. 60
4.4. Implementación del algoritmo *AES-Rijndael* en diferentes tecnologías. 61
4.5. Ventajas y limitaciones del AES. 62
4.5.1. Ventajas 62
4.5.2. Limitaciones. 63

5. Conclusiones 65

A. Representación de un Byte en el campo $GF(2^8)$ 73
A.1. Construcción del campo finito $GF(2^8)$ 73
A.2. Suma de polinomios en el campo $GF(2^8)$. 75
A.3. Multiplicación de polinomios en el campo $GF(2^8)$. 75

B. Representación de palabras en el campo $GF(2^8)$ 79
B.1. El anillo $GF(2^8)$. 81

C. Transformaciones no lineal S-Box 83
C.1. Función sustitución (*SubByte*) 83

D. Criptoanálisis 87
D.1. Texto claro desconocido y clave desconocida. 88
D.2. Texto claro desconocido y clave desconocida. 88
D.3. Criptoanálisis Diferencial 89
D.4. Criptoanálisis Lineal 90
D.5. Ataques por análisis temporal (*Timing Attacks*). 91
D.6. Ataque diferencial (*Impossible Differentials Attack*). 92
D.7. Número de rondas. 92
D.8. Pesos de los conjuntos diferenciales y de los conjuntos lineales. 94
D.8.1. Propiedades de simetría a calves débiles. 94
D.9. Ataque cuadrado (*Square Attack.*) 95
D.10.Ataque por colisiones (*Collision Attack*) 95
D.11.Fracciones continuas 95
D.12.Ataque *XSL* 96

D.13.Ataque por combinaciones (*Embedding*) 96
D.14.Ataque cifrado dual (*Dual Cipher*) 96
D.14.1. Claves débiles como en IDEA. 97
D.15. El conjunto de posibles cifradores para una longitud de bloque y de clave dadas. 97
D.15.1. Factor de trabajo y requisitos de memoria para los ataques . . 97
D.15.2. Factor de trabajo y requisitos de memoria para los ataques . . 98

E. Sistema Seguridad **99**
E.1. Sistema Seguridad IPv6 100
E.2. Certificados X.509 y autoridades de certificación 101
E.3. Seguridad en Internet. 102
E.3.1. Seguridad en redes de comunicaciones. 103
E.3.2. Aplicaciones de seguridad en redes de comunicaciones. 104

RESUMEN

La información ha sido, desde siempre, una de las más preciadas posesiones para el hombre y, por ello, se han utilizado métodos y mecanismos para salvaguardarlas de posibles ataques. A lo largo de los años, la humanidad ha ido buscando mecanismos para transmitir datos o información. Sin embargo, los grandes logros conseguidos en este tema tenían como contrapartida el aumento en la inseguridad de la misma, lo que provocó la aparición de diversos mecanismos de criptoanálisis para intentar romper la confidencialidad de las comunicaciones. De esta forma surgió y prosperó el estudio y utilización de mecanismos criptográficos que permitiesen hacer ininteligible la información a toda persona que no estuviese autorizada.

El algoritmo Rijndael fue elegido por el NIST (National Institute of Standards and Technology), para ser el estándar en los próximos 20 años y es llamado AES (Advanced Encryption Standar), sus principales características fueron su fácil diseño, su versatilidad en ser implementado en diferentes escenarios, así como ser inmune a los ataques conocidos hasta la fecha, soportar bloques de datos de 128 bits y claves de 128bits. La idea básica general es tener un estándar que mejore el performance de TDES (Triple Data Encryption Algorithm), y sea resistente a los ataques conocidos.
La descripción de AES (Advanced Encryption Standar) es simple si se cuentan con todos los elementos los cuales consiste esencialmente en la descripción de las 4 transformaciones básicas de AES: ByteSub, ShiftRow, MixColumns, y AddRoundKey, el proceso de extensión de la clave.

Por lo que este trabajo se enfoco en el estudio a fondo del algoritmo de encriptado AES de datos, siendo implementadas nuevas variantes en su estructura las cuales fueron propuestas por sus autores quedando asi un algoritmo diferente al estándar, sin perder su fortaleza. Todo esto se llevo acabó a través de su implementación en el lenguaje de programación C y una pagina de Internet .

Capítulo 1

Introducción

1.1. Antecedentes

La Criptografía es la ciencia que se ocupa del cifrado seguro de mensajes y está estrechamente relacionada con las matemáticas. Existen dos tipos principales de criptografía de uso común hoy día. La más antigua (usada hasta los años 70), la cual se conoce como criptografía de clave sencilla o de clave secreta (criptografía simétrica), que resulta útil en muchos casos, aunque tiene limitaciones significativas.

Los algoritmos simétricos, o de clave secreta, se caracterizan por ser altamente eficientes (en relación al tamaño de su clave) y robustos. Se les llama así porque se emplea la misma clave para cifrar y descifrar. Ya que se basan en el uso de claves secretas que previamente hay que intercambiar mediante canales seguros, con los riesgos que ello supone. Cada parte debe poseer una copia de la clave para protegida y mantenerla fuera del alcance de los demás. Además, dichas claves no se deben utilizar para cifrar varios mensajes, ya que si se interceptaran algunos de ellos, se podrían encontrar métodos para descifrar datos.

Por sí solo, este tipo de cifrado no es suficiente para desarrollar el pleno potencial del comercio electrónico, el cual debe vincular a un número ilimitado de compradores y vendedores de todas partes del mundo, resulta poco práctico que una gran corporación intercambie claves con miles o incluso millones de clientes o, peor todavía, con posibles clientes con los que nunca ha tratado.

1.2. Cifrado asimétrico y simétrico

La solución a la seguridad en toda red abierta es una forma de codificación novedosa y sofisticada, desarrollada por los matemáticos de *MIT* en los años setenta (Pino 2003), y conocida como clave pública o criptografía asimétrica.

Los algoritmos asimétricos tienen claves distintas para el cifrado y descifrado. Por ello, también se les llama algoritmos de clave pública que permiten eliminar el gran inconveniente de cómo hacer llegar al remitente la clave del cifrado.

En el caso de los algoritmos asimétricos se usa una clave pública (para cifrar) y una secreta (para descifrar). La primera se publica en un tipo de directorio al que el público en general tiene acceso (una especie de guía telefónica); mientras que la privada se mantiene en secreto. Las dos claves funcionan conjuntamente como un curioso dúo.

De esta manera, la intercepción de la clave pública resulta inútil para descifrar el mensaje, puesto que para ello se requiere la clave secreta. Cualquier tipo de datos o información que una de las claves cierre, sólo podrá abrirse con la otra. De forma que, por ejemplo, si queremos enviar a un amigo un mensaje sin que ningún intruso lo lea, buscamos la clave pública del amigo y la utilizamos para realizar el cifrado del texto. Luego, cuando él lo recibe, utiliza su clave privada para descifrar el mensaje.

Si un extraño interceptara este mensaje, no podría descifrarlo porque no tendría la clave privada de ese amigo nuestro. Como desventaja, las claves han de ser de mayor tamaño para ofrecer más seguridad comparable a la de los algoritmos simétricos. También resultan más lentos y producen mensajes cifrados de mayor tamaño.

Para que existan sistemas de clave pública es necesario encontrar funciones de dirección única, es decir, funciones fáciles de calcular, para cifrar, pero cuya inversa que se usa para descifrar, sea prácticamente imposible de calcular a no ser que se conozca la clave secreta.

Por otra parte los algoritmos simétricos se dividen en los de cifrado de flujo o síncronos y los de cifrado en bloque (Pino 2003). Los primeros cifrar el mensaje original bit a bit, mientras que los segundos toman un número de bits (típicamente 128 bits en los algoritmos modernos) y los cifra como si se tratara de una sola unidad.

Generalmente, los algoritmos simétricos se ejecutan más rápido que los asimétricos. En la práctica se suelen usar juntos, de modo que el algoritmo de clave pública se emplea para cifrar una clave generada, y ésta se emplea para cifrar el mensaje usando un algoritmo simétrico.

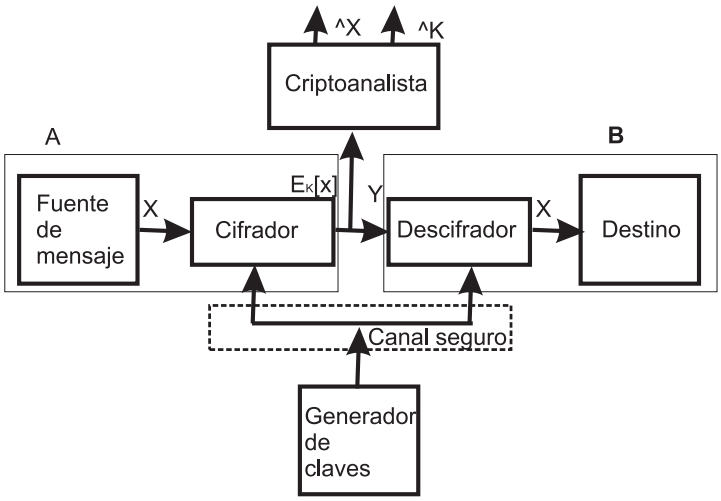

Figura 1.1: Criptosistema simétrico.
X:mensaje claro
Y:mensaje cifrado
$E_k[*]$: Operación de cifrado de mensaje con una clave.
K:clave simétrica
X :Mensaje estimado.

Cifrado Simétrico	Cifrado asimétrico
Confiabilidad	Confiabilidad
Cierto grado de autentificación	Autentificación total
Sin firma digital	Firma digital
Alta velocidad	Baja velocidad

Cuadro 1.1: Cifrado Simétrico y Asimétrico

1.2.1. Cifrado de flujo

En el cifrado de flujo o síncrono la secuencia pseudoaleatoria (clave) generada para cifrar y descifrar un mensaje es independiente de éste. En los cifradores de flujo la secuencia de cifrado es función del mensaje. Así, el emisor y el receptor deben usar la misma clave que, además debe estar sincronizada para poder establecer la comunicación. Por este motivo se utilizan señales de sincronización, que no son necesarias en el cifrado de bloque, ya que al tener una realimentación, en caso de pérdida de sincronismo, éste puede recuperarse transcurrido un tiempo. El Cifrado *Vernam* es un ejemplo de un cifrado de flujo,(Henk 2000).

1.2.2. Cifrado de bloque

Para la transmisión de datos confidenciales entre computadoras se desarrolló a principios de la década de 1970 *LUCIFER*, un sistema de cifrado de bloque basado tanto en la sustitución como en la transposición, y en 1976 se elaboró la norma de cifrado de datos o (*Data Encryption Standard*)DES sobre la base del primero.

Este sistema todavía es ampliamente usado, sobre todo, en el campo financiero. El algoritmo DES transforma bloques de mensaje de 64 bits en otros equivalentes de texto cifrado, empleando una clave de 56 bits. Cada usuario elige una clave al azar, que sólo comunica a aquellas personas autorizadas a conocer los datos protegidos.

El mensaje se cifra y descifra automáticamente mediante equipos electrónicos incorporados a las computadoras emisoras y receptoras. Como existen más de 70.000 billones de combinaciones de 56 bits, la probabilidad de descubrir la clave aleatoria sería mínima. Así que es importante tener un método de cifrado altamente resistente frente a ataques de criptoanálisis que más adelante se detallaran.

Sin embargo, algunos expertos han criticado la técnica DES por su vulnerabilidad frente a los potentes métodos de criptoanálisis posibles para las grandes y veloces computadoras y no lo han considerado apropiado para las aplicaciones actuales.

Por desgracia, el tamaño de su clave (56 bits) lo hace vulnerable a ataques de fuerza bruta. Un reciente ataque contra un mensaje con cifrado DES requirió el uso de cientos de computadoras durante 140 días. Pero hay diseños de máquinas que con un costo de un millón de dólares, podrían descifrar mensajes DES en cuestión de minutos.

Quizá por eso el gobierno de *EEUU* lo utiliza solamente para cifrar datos no clasificados. En la actualidad ofrece protección contra el pirata informático habitual; pero no contra un esfuerzo masivo por parte de un usuario con grandes recursos.

Una variante del algoritmo DES, es *Triple-DES (3DES)* (Amparo 2001), basado en el uso de tres veces el algoritmo DES que consiste en una secuencia de cifrado-descifrado-cifrado con tres claves diferentes y no relacionadas entre sí. El sistema 3DES es bastante más seguro que el DES (simple), aunque presenta el inconveniente de ser considerablemente más lento que los modernos sistemas de cifrado en bloque.

Actualmente, a pesar de que DES ya no se considera una solución práctica, es usado a menudo para describir nuevas técnicas de criptoanálisis.

Es importante destacar que incluso hoy, no existe ninguna técnica de criptoanálisis que pueda vulnerar completamente a DES de un modo estructural; de hecho, la única debilidad de DES es su pequeño tamaño de la clave aun nado al pequeño tamaño del bloque.

Actualmente el algoritmo DES resulta obsoleto y, para sustituirlo el *National Institute of Standards and Technology* (NIST) realizo una convocatoria para desarrollar un nuevo algoritmo para enfrentar las debilidades del sistema Triple-DES como una solución temporal.

Los cinco algoritmos, elegidos entre un total de quince, fueron MARS, RC6, *Rijndael*, *Serpent* y *Twofish*.

AES-Rijndael es un cifrador en bloque diseñado por *John Daemen* y *Vincent Rijndael* como algoritmo candidato al (*AES-Advanced Encryption Standard*). Su diseño estuvo fuertemente influenciado por el de un cifrador (*block cipher Square*), que también fue creado por *John Daemen* y *Vincent Rijndalel* y se centraba en el estudio de la resistencia al criptoanálisis (ver apéndice D). El nombre del algoritmo es una combinación de los nombres de sus dos creadores. El cifrador tiene longitudes de bloque y de clave variables y puede ser implementado de forma muy eficiente en una amplia gama en forma de software y hardware. Como todos los candidatos AES, es un algoritmo de cifrado muy seguro y hasta la fecha no se le han encontrado puntos débiles.

La longitud de la clave de *AES-Rijndael*, si bien es variable, Para *AES-Rijndael* debe ser de 128, 192 ó 256 bits, según los requisitos establecidos (Joan 2002). Asimismo, la longitud del bloque puede variar entre 128, 192 ó 256 bits. Todas las posibles combinaciones (nueve en total) entre longitudes de clave y bloque son válidas, aunque la longitud oficial de bloque para *AES-Rijndael* es de 128 bits.

Las longitudes de la clave y el bloque pueden ser fácilmente ampliadas a múltiplos de 32 bits. El número de interacciones o rondas del algoritmo principal puede variar de 10 a 14 y depende del tamaño del bloque y de la longitud de la clave, por lo que se incrementa en numero de rondas.

La implementación de *AES-Rijndael* usa también claves de 192 y 256 bits para bloque de 128 bits. Usando la mayor longitud posible de clave conseguimos la máxima seguridad para el usuario.

La filosofía de *AES-Rijndael* concede mayor importancia a la seguridad que a la velocidad. Si el usuario proporciona una clave de mayor longitud se alteraría el nivel

de seguridad (Amparo 2001), para hacerla de 256 bits. Y aunque acepta tamaños de bloque mayores que 128 bits, no existe ninguna razón para usarlos siendo que este número de bits ha sido elegido como tamaño estándar.

1.3. Cifrado AES

The Advanced Encryption Standard AES es el nuevo estándar de criptografía simétrica adoptado por *Federal Information Processing Standards* (FIPS).

En 1977 apareció la primera versión del estándar *AES-Rijndael* (Joan 2002), que es el nuevo estándar de criptografía simétrica adoptado por FIPS *AES-Rijndael* sustituyendo al algoritmo DES y sus posteriores versiones.

Aun que existieron opiniones controversiales respecto al DES, ya que nunca se produjo un ataque que derivara en la obtención de la clave secreta partiendo de la información cifrada.

A pesar de su corta longitud de clave se comprometía poco a poco. La última reafirmación de DES se realizo en Octubre de 1999 y fue sustituido por TDES, que es una versión múltiple de DES, designado como TDEA (*Triple Data Encryption Algorithm*). Sin embargo ya se tenían planes de encontrar un reemplazo definitivo a DES. A pesar de que existían un gran un número de algoritmos como: *IDEA, RC5, skipjack, 3-way, FEAL, LOKI, SAFER, SHARK*, etc. El NIST decidió convocar a un concurso que tuvo como principales objetivos obtener un algoritmo simétrico que garantizara su seguridad para los próximos 20 años a partir del año 2000. Finalmente el 2 de Octubre del 2000 se adopta el Algoritmo *AES-RIJNDAEL* como AES. Pero fue asumido oficialmente hasta Noviembre 26 del 2001 en el FIPS. A partir de esa fecha se realizan conferencias especiales para analizar la situación actual del AES(Pais 2003).

El algoritmo AES-*Rijndael* fue elegido principalmente por garantizar seguridad, que significa ser inmune a los ataques conocidos, tener un diseño simple, y poder ser implementado en la mayoría de los escenarios posibles. Desde dispositivos con recursos limitados, como smart cards, hasta procesadores paralelos. El tiempo ha permitido que AES sea adaptado poco a poco, desde los protocolos más usados como SSL, hasta las aplicaciones más especializadas, como en IP.

La descripción de AES es simple si se cuenta con todos los elementos. Esta consiste en dos partes, la primera en el proceso de cifrado y la segunda en el proceso de generación de las subclaves, una primera aproximación se muestra en la siguiente Figura 1.2:

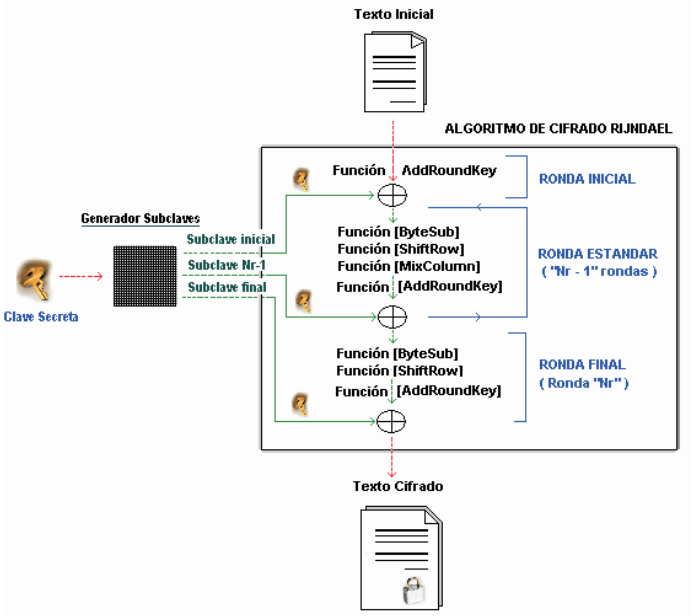

Figura 1.2: Diagrama de encriptado de AES.

1.4. Planteamiento del Problema

El objetivo principal de la criptografía es garantizar la seguridad de las comunicaciones. Cualquier sistema de cifrado de datos tiene puntos débiles, y los modernos (*DES, TDES, RSA, Diffie-Hellman, PGP, AES*) no escapan a esta regla.

En ocasiones un sistema se implementa erróneamente en la práctica, de modo que las ventajas quedan anuladas por el mal uso del usuario. Por ejemplo, casi todos los códigos actuales requieren cadenas de dígitos elegidos aleatoriamente, para lo cual se utilizan generadores de números aleatorios o pseudoaleatorios. Pero si dichos números no son realmente aleatorios, las claves así generadas se vuelven vulnerables. Por ejemplo un fallo de implementación de dicho tipo hizo que las comunicaciones seguras utilizando el navegador Netscape Navigator o explore pudiesen ser leídas en segundos, el sofisticado protocolo SSL resultaba en la práctica inútil porque utilizaba números no tan aleatorios. Cualquier programa de cifrado de datos es susceptible a mil y una fallas de seguridad, (Jesus de Marcelo 2002).

Como se vio, el cifrado de datos es una herramienta muy necesaria en nuestros días es por ésto que el presente trabajo de Tesis se enfoca al cifrado de datos a través del algoritmo AES ya que es muy confiable, seguro y eficiente. La investigación se ha dividido en cinco capítulos en los cuales se estudiara el algoritmo AES y se plantea una aplicación para la transmisión de datos.

En el capítulo uno se presenta una introducción al cifrado de datos, para después estudiar a detalle el algoritmo *AES*. En el capítulo dos se presenta una descripción matemática del algoritmo *AES-RIJNDAEL* así como sus variantes . En el capitulo tres se describe su implementación en un lenguaje de programación (C++), para mostrar en el capítulo cuatro se los resultados e implementación, a través de una página WEB colocada en un servidor instalado en la Facultad de Ingeniera, finalizando con las conclusiones en el capitulo cinco.

Capítulo 2

Descripción del algoritmo *AES-Rijndael*

2.1. Introducción al Algoritmo *AES-Rijndael*

En este Capítulo se da una descripción del algoritmo AES, primero recordemos que *AES-Rijndael* trabaja con bloques de datos de 128 bits y longitudes de claves de 128, 192 ó 256 bits. Además AES tiene 10, 12 ó 14 rondas respectivamente, cada rondas de *AES-Rijndael* consiste en la aplicación de una ronda estándar, entendiendo por rondas al conjunto de 4 transformaciones básicas y la última ronda es especial pues consiste de 3 operaciones básicas, añadiendo siempre una ronda inicial. Por otro lado tenemos el programa de claves o extensión de la clave.

Por ello el algoritmo queda constituido de la siguiente manera: adición de la clave (*AddRoundKey*), sustitución (*SubByte*), desplazamiento de filas (*ShiftRows*), multiplicación de columnas (*MixColumns*), y por último el extensión de las subclaves (*Key Schedule*), las cuales se describirán más adelante. Recordemos que AES utiliza a la entrada bloques de 16 bytes esto es 128 bits (Henk 2000). Los cuales se pueden describir como un vector de tamaño 16 los cuales representan el conjunto de bytes a la entrada del algoritmo, esto es:

$$(a_{00}, a_{01}, a_{02}, a_{03}, a_{04}, a_{05}, a_{06}, a_{07}, a_{08}, a_{09}, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}) \tag{2.1}$$

La entrada y salida de datos en *AES-Rijndael* se realiza mediante un vector unidimensional de 16 bytes. Estos bytes están numerados con valores que parten desde el 0 hasta $(4 * N_b) - 1$, donde N_b es el tamaño del bloque dividido entre 32, (Muñoz 2004) es decir:

$$N_b = \frac{\text{tamaño del bloque en bits}}{32} \tag{2.2}$$

Por lo que para un bloque de 128 bits se tiene de acuerdo a la ecuación 2.2, $N_b = 4$ y estarían numerados desde el 0 hasta el 15. El valor de N_b y el rango de índices correspondientes del vector entrada-salida la cual queda formada por bloques de 128 bits , por lo tanto el algoritmo se basa en aplicar un número de rondas determinado, los cuales se muestran en la figura 2.1.

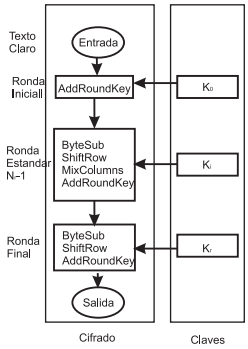

Figura 2.1: Diagrama de Bloques AES.

Donde $[a_{00}]$, representa al primer grupo de 8 bits (1Byte), en la cuadro 2.1 se muestra la relación de estos parámetros (Muñoz 2004).

Numero de Bits	Numero de Bytes	N_b	Long. vector	Rango índice
128	16	4	16	0...15
192	24	6	24	0...23
256	32	8	32	0...31

Cuadro 2.1: Correspondencia entre el tamaño en bits y N_b

Una vez que tengamos el vector de entrada completo, necesitamos formar una matriz de tamaño adecuado, si el bloque es de 128 bits, se tendrá una matriz de [4 x N_b] la cual se denomina matriz de estado y será llenada de la forma siguiente:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{2.3}$$

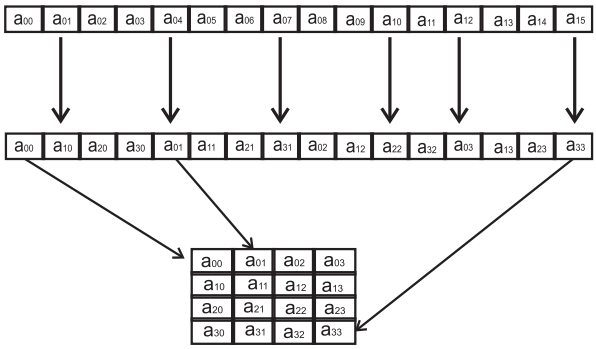

Figura 2.2: Elementos de la matriz de inicio

De acuerdo al cuadro (2.1), para el caso de 128 bits, empezamos con la columna 0 e iremos rellenando hacia abajo hasta el final para luego proceder de igual forma con la columna siguiente, (Joan 2002). Esta matriz se denomina como $[a_{ij}]$ la cual será la matriz de estado.

Para bloques de 192 bits se agregan 2 columnas más a la matriz $[a_{ij}]$. En el caso de 256 bits se agregan 4 columnas a la matriz $[a_{ij}]$. Esto es, para los bloques de 192 y 256 bits, se obtendran las matrices [4 x 6] y [4 x 8] respectivamente.

El número de rondas que tiene el algoritmo es un parámetro variable que depende de los valores N_b y N_k, y estos a su vez dependen del tamaño del bloque y de la longitud de clave respectivamente, siendo (Muñoz 2004).

$$N_k = \frac{\text{longitud de la clave en bits}}{32} \tag{2.4}$$

En donde N_r denota el número de rondas quedando, (Muñoz 2004).

$$N_r = \max(N_k, N_b) + 6 \tag{2.5}$$

El valor de N_r se específica en el cuadro 2.2. Esto es que para un bloque de 128 bits el numero de rondas es de 10, por lo que se generan 9 subclaves, una para cada ronda.

Clave/Bloque	$N_b = 4$(128 bits)	$N_b = 6$(192 bits)	$N_b = 8$(256bits)
N_k=4	10	12	14
N_k=6	12	12	14
N_k=8	14	14	14

Cuadro 2.2: Número de rondas del algoritmo AES según el valor de N_b y N_k

Se puede apreciar que el número de rondas aumenta al incrementarse el tamaño del bloque y la longitud de la clave.

2.2. Función Adición de la clave (*AddRoundKey*).

Esta transformación toma la matriz $[a_{i,j}]$ y realiza un operación XOR byte a byte con la clave, la cual es representada por medio del vector 2.6.

$$(k_{00}, k_{10}, k_{20}, k_{30}, k_{10}, k_{11}, k_{21}, k_{31}, k_{02}, k_{12}, k_{22}, k_{32}, k_{03}, k_{13}, k_{23}, k_{33}) \tag{2.6}$$

Del que se obtiene la matriz $[k_{ij}]$,

$$\begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} \tag{2.7}$$

Por lo que la operación XOR queda:

$$[A_{i,j}] = [a_{i,j}] \oplus [k_{ij}] \tag{2.8}$$

A partir de esta operación se generan las subclaves necesarias para cada una de las rondas, (Joan 2002).

2.3. Sustitución (*ByteSub*).

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{bmatrix} \rightarrow S - Box \rightarrow \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} \tag{2.9}$$

La matriz $S - Box$ se diseño para contrarestar el criptoanálisis lineal y diferencial y, al mismo tiempo otorgar resistencia frente ataques como la manipulación algebraica y de interpolación.

Para ello se creó un procedimiento que tuviése las siguientes características:

• Inversible.

• Minimizar el total de números que no sean combinaciones lineales de los bits de entrada y salida.

• Minimizar la relación entre los bits de entrada y de salida.

• Expresiones algebraicas complejas con coeficientes pertenecientes a $GF(2^8)$.

•Sencillo de describir.

Existen diversa formas para construir la matriz $S - Box$.

El primer paso consiste en definir las tablas de sustitución mediante en cálculo de la inversa multiplicativa en $GF(2^8)$, la cual tiene propiedades de no linealidad. Sin embargo, la simplicidad del algebra del diseño podría permitir ataques de manipulaciones algebraicas.

Para evitar estos ataques se añade una transformación a fin, la cual se constituye por una matriz formada por un polinomio representado en forma de vector (0110011) (Joan 2002), elegido de tal forma que ningún valor de la matriz $S - Box$ ofresca las siguientes relaciones, esto es:

$$S - Box(a) = a \tag{2.10}$$

$$S - Box(a) = \bar{a} \tag{2.11}$$

Existen otras matrices $S - Box$ que satisfacen las características del algoritmo AES (Joan 2002), que garantizan la seguridad ante ataques de tipo diferencial y lineal (Muñoz 2004). (Ver apéndices D.3 y D.4).

Existe una gran variedad de métodos que cumplen las tres primeras condiciones. En (Muñoz 2004) se puede encontrar una lista muy completa de polinomios irreducibles con los que se pueden generar matrices $S - Box$. Para el caso específico del AES se toma el polinomio $m(x) = x^8 + x^4 + x^3 + x + 1$ (Joan 2002), el cual se multiplica por su inverso multiplicativo y el valor de cero queda igual, ya que no tiene inversa.

Visto en forma de matrices queda de la siguiente manera:

$$\begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{2.12}$$

Para el proceso de descifrado se tiene que cumplir la siguiente condición:

$$a(x) \bullet b(x) \bmod x^8 + x^4 + x^3 + x + 1 = 1 \tag{2.13}$$

Lo que garantiza que existe un único inverso.

Así se genera la matriz inversa, la cual se construye realizando la siguiente operación:

$$\begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{2.14}$$

En este caso cada elemento de la matriz $[A_{i,j}]$, que resulta de la operación, $[a_{ij}] \oplus [k_{ij}]$ es sustituido por el elemento correspondiente de la matriz $[s_{ij}]$.

Por lo que la matriz $[A_{i,j}]$ cumple con:

1)Todo byte puede verse como un elemento del campo finito $GF(2^8)$ (como se menciona en el apéndice C). Todo elemento tiene inverso multiplicativo, entonces esta primera función de sustitución (*bytesub*), asocia el inverso multiplicativo en $GF(2^8)$, es decir $a_{ij} \rightarrow a_{ij}^{-1} \in GF(2^8)$ al elemento cero se le asocia el mismo cero. Siendo una matriz $S - Box$, la cual se muestra en el Apéndice C.

Por ejemplo, si tenemos el número 88 en hexadecimal su inverso multiplicativo será 9b de acuerdo al cuadro C.1(ver apéndice C) .

2)La transformación lineal que sigue al inverso se aplica bit a bit de acuerdo con la siguiente regla:

$$b_1 = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i \tag{2.15}$$

Donde $0 \leq i < 8$ y $c = 01100011 = 63_x$. Por lo que de acuerdo con la ecuación C.1, se forma la tabla representada en C.2.

Existen otras tablas $S-Box$ que satisfacen los criterios del diseño, por lo que se pueden reemplazar por otras. La estructura del cifrado y el numero de rondas están definidos incluso para usar tablas $S-Box$ diferentes, de acuerdo con el criterio del diseñador. Aunque que se correría el riesgo de debilitar el algoritmo si no se toman las medidas adecuadas, esto es, que los polinomios generadores no cumplan con las especificaciones requeridas (Joan 2002).

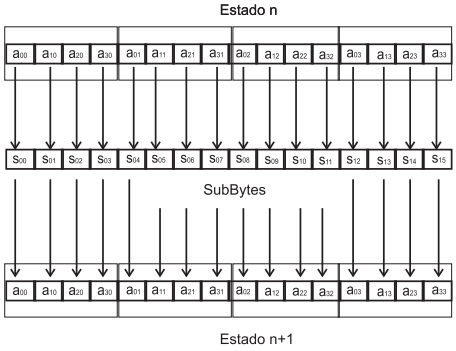

Figura 2.3: Elementos de la matriz sustitución (*SubByte*)

2.4. Desplazamiento de filas (*ShiftRow*)

La transformación desplazamiento de filas (*ShiftRow*) se aplica a la matriz estado $[s_{ij}]$, realizando corrimientos izquierdos circulares de bytes a las filas de la matriz.

Para comprender mejor el funcionamiento de esta transformación vamos a aplicarla al caso del algoritmo AES con $N_b = 4$. En este caso los desplazamientos serán de 1, 2 y 3 para F_1, F_2 y F_3 respectivamente.

La fila 0 no se modifica, los bytes de la fila 1 se desplazan 1 posición a la izquierda, los de la fila 2 se desplazan 2 posiciones a la izquierda y los de la fila 3 se desplazan 3 posiciones. Aunque en este caso coincida el número de fila con el valor del desplazamiento, éstos se efectúan de acuerdo al cuadro 2.3 (Joan 2002).

Tamaño del bloque	F_1	F_2	F_3
128 bits ($N_b = 4$)	1	2	3
192 bits ($N_b = 6$)	1	2	3
256 bits ($N_b = 8$)	1	3	4

Cuadro 2.3: Valor de desplazamiento de las filas de la matriz $[s_{ij}]$

Si $N_b = 4$, shift(1,4)=1, shift(2,4)=2 y shift(3,4)=3.

Esto es:
Se recorren 0 bytes en el primer renglón, 1 byte en el segundo renglón.

$$\begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix} \rightarrow \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{330} \end{matrix} \rightarrow \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix}$$

2 bytes en el tercer renglón.

$$\begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix} \rightarrow \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix} \rightarrow \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{22} & A_{23} & A_{20} & A_{21} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix}$$

Y 3 bytes en el cuarto renglón.

$$\begin{matrix} A_{00} & A_{01} & A_{02} & a_{03} \\ A_{11} & A_{12} & A_{13} & a_{10} \\ A_{20} & A_{21} & A_{22} & a_{23} \\ A_{30} & A_{31} & A_{32} & a_{33} \end{matrix} \rightarrow \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix} \rightarrow \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{33} & A_{30} & A_{31} & A_{32} \end{matrix}$$

El numero de desplazamientos en esta parte se elegió ya que ofrecen resistencia contra ataques denominados como truncamiento diferencial (*truncanted differntials*) y resistencia contra ataques cuadrados (*squar*), por lo que para ciertos desplazamientos se tiene mayor resistencia. Es por este motivo que se deben encontrar los desplazamientos que tengan una mayor resistencia, cuyos valores dependen de la longitud del bloque, que se definen en los cuadros 2.3 y 2.4.

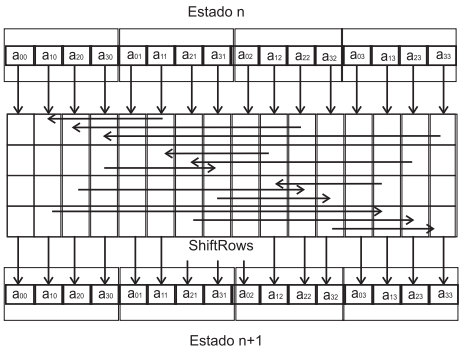

Figura 2.4: Elementos de la matriz desplazamiento (*ShiftRows*)

La estructura del algoritmo fue diseñada para permitir cualquier tamaño de bloque que sea múltiplo de 4 bytes, con un numero mínimo de 16 bytes. Las funciones de adición (*AddRoundKey*), sustitución (*SubByte*), multiplicación de columnas (*MixColumns*), son independientes del tamaño del bloque; sin embargo la transformación desplazamiento de filas (*ShiftRows*) sí depende de la longitud del bloque, siendo necesario definir los valores de F_1, F_2 y F_3, por lo que para bloques de mayor tamaño se tiene:

Tamaño del bloque	F_1	F_2	F_3
160 bits N_b=5	1	2	3
224 bitsN_b=7	1	2	4

Cuadro 2.4: Valores extras de F_i segun el tamaño del bloque

2.5. Multiplicación de columnas (*MixColumns*).

La transformación multiplicación de columnas (*MixColumns*) actúa sobre cada byte de una misma columna de la matriz de estado $[S_{i,j}]$ que resulta de la transformación (*ShiftRow*). Esta función permite una mezcla de los bytes de las columnas. La función multiplicación de columnas se realiza a partir una transformación lineal de [4 x 4] considerando los siguientes criterios:

a)Que sea inversible.
b)Linealidad en $GF(8^2)$.
c)Propiedades de alta difusión.
d)Velocidad en los procesadores.
e)Simetría.
f)Sencillez de descripción.

La elección de los coeficientes y los polinomios permiten una alta difusión entre los Bytes resultantes de la trasformación.

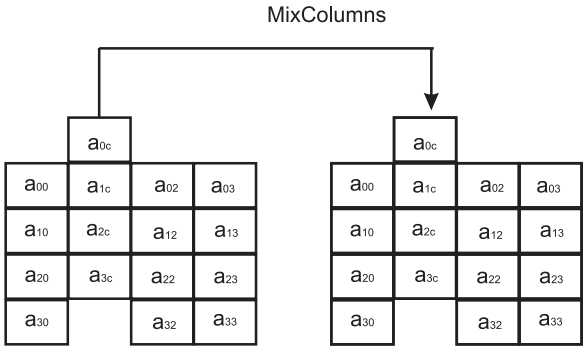

Figura 2.5: Función multiplicación de columnas (*Mixcolumns*)

En esta transformación las columnas de la matriz de estado $[S_{i,j}]$ son consideradas como polinomios con coeficientes pertenecientes a $GF(2^8)$. Estos polinomios se multiplican modulo $M(x)$por un polinomio $C(x)$, recordemos que $M(x) = x^4 + 1$. (Ver apéndice B), cuya operación se define como: (Joan 2002)

$$S'(x) = C(x) \oplus S(x) \tag{2.16}$$

Por lo que la transformación multiplicación de columnas (*MixColumns*) consiste en multiplicar las columnas de byte modulo $M(x) = x^4 + 1$ por el polinomio $C(x)$, siendo $C(x) = 03x^3+01x^2+01x+02$. Este polinomio $C(x)$ es coprimo $M(x) = x^4+1$, lo que permite que sea inversible, quedando:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{2.17}$$

Donde $S'(x)$ representa la matriz de estado resultado de la siguiente transformación y $S(x)$ a la matriz de estado anterior (estado intermedio 3).

Desarrollando la matriz $S'(x)$, obtenemos que cada byte nuevo de la matriz de estado es una combinación de varios bytes de las distintas filas que forman cada una de las columnas especificas, esto es:

$$\begin{aligned} S'_{0,c} &= ([02] \bullet S_{0,c}) \oplus ([03] \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} &= S_{0,c} \oplus ([02] \bullet S_{1,c}) \oplus ([03] \bullet S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} &= S_{0,c} \oplus S_{1,c} \oplus ([02] \bullet S_{2,c}) \oplus ([03] \bullet S_{3,c}) \\ S'_{3,c} &= ([03] \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus ([02] \bullet S_{3,c}) \end{aligned} \tag{2.18}$$

Por ejemplo, si consideramos un estado intermedio 3, quedaría:

$$\begin{bmatrix} D4 & E0 & B8 & 1E \\ BF & B4 & 41 & 27 \\ 5D & 52 & 11 & 98 \\ 30 & AE & F1 & E5 \end{bmatrix} \tag{2.19}$$

Calculando el byte de la fila 0 y la columna 0 del estado intermedio 3 se obtendrá de acuerdo con la ecuación 2.18 :

$$S'_{00} = ([02] \bullet S_{0,c}) \oplus ([03] \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \tag{2.20}$$

$$S'_{00} = ([02] \bullet D4) \oplus ([03] \bullet BF) \oplus 5D \oplus 30 \tag{2.21}$$

Por lo que al realizar las operaciones de acuerdo a los apéndices A y B, queda:

$$\begin{aligned} [02] \bullet [D4] = \\ x \bullet (x^7 + x^6 + x^4 + x^2) \mathrm{mod} x^4 + 1 = \\ x^8 + x^7 + x^5 + x^3 \mathrm{mod} x^4 + 1 = \\ x^7 + x^5 + x^4 + x^3 = \\ 10111000 = [B8] \end{aligned} \tag{2.22}$$

$$\begin{aligned} [03] \bullet [BF] = \\ x + 1 \bullet (x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) \mathrm{mod} x^4 + 1 = \\ x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1 \mathrm{mod} x^4 + 1 = \\ x^7 + x^6 + x^4 + 1 = \\ 11010001 = [D1] \end{aligned} \tag{2.23}$$

De acuerdo con los resultados obtenidos en 2.22 y 2.23, y sustituyendo en 2.21 obtenemos:

$$S'_{00} = B8 \oplus D1 \oplus 5D \oplus 30 = 04 \tag{2.24}$$

Luego el byte $S_{00} = D4$ es sustituido por el byte $S'_{00} = 04$ en la matriz de estados intermedio 4 y así se obtienen los resultados de los demás términos de la matriz. Para descifrar o invertir esta trasformación se debe realizar el mismo procedimiento descrito en esta sección, sólo que el polinomio $D(x)$ debe cumplir con:

$$C(x) \oplus D(x) = 01 \tag{2.25}$$

Esto es que $D(x)$ debe ser el inverso multiplicativo de $C(x)$, por lo que el polinomio es $D(x) = 0Bx^3 + 0Dx^2 + 09x + 0E$, (Joan 2002). La transformada inversa queda en forma matricial de la siguiente manera:

$$S(x) = D(x) \oplus S'(x) \tag{2.26}$$

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} \tag{2.27}$$

Por lo que queda:

$$\begin{aligned} S_{0,c} &= ([0E] \bullet S'_{0,c}) \oplus ([0B] \bullet S'_{1,c}) \oplus ([0D] \bullet S'_{2,c}) \oplus ([09] \bullet S'_{3,c}) \\ S_{1,c} &= ([09] \bullet S'_{0,c}) \oplus ([0E] \bullet S'_{1,c}) \oplus ([0B] \bullet S'_{2,c}) \oplus ([0D] \bullet S'_{3,c}) \\ S_{2,c} &= ([0D] \bullet S'_{0,c}) \oplus ([09] \bullet S'_{1,c}) \oplus ([0E] \bullet S'_{2,c}) \oplus ([0B] \bullet S'_{3,c}) \\ S_{3,c} &= ([0B] \bullet S'_{0,c}) \oplus ([0D] \bullet S'_{1,c}) \oplus ([09] \bullet S'_{2,c}) \oplus ([0E] \bullet S'_{3,c}) \end{aligned} \tag{2.28}$$

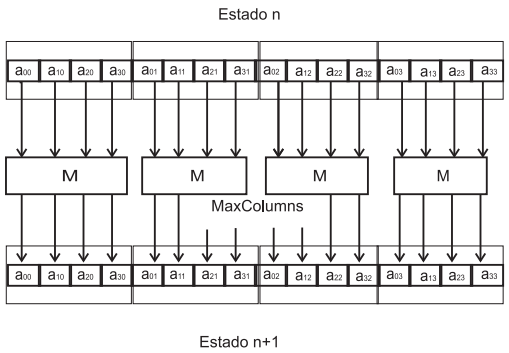

Figura 2.6: Elementos de la matriz de MaxComns

2.6. Generación de las subclaves (*Key Schedule*).

Aunque *AES-Rijndael* esta diseñado para manejar longitudes de clave y de bloques distintos, el estándar sólo permite bloques de 128 bits, y longitudes de clave de 128, 192 y 256 bits.

Por otra parte, una longitud de 128 bits garantiza la seguridad del sistema hasta después del año 2030 (Pino 2003), por lo que consideramos sólo claves de longitud de 128 bits; aunque el algoritmo puede extenderse a otros casos.

La función de selección de claves simplemente toma los valores de la secuencia obtenida por la función de expansión de claves que genera cada subclave k_i, esto es que se toma $N_b * 4$ byte para cada ronda.

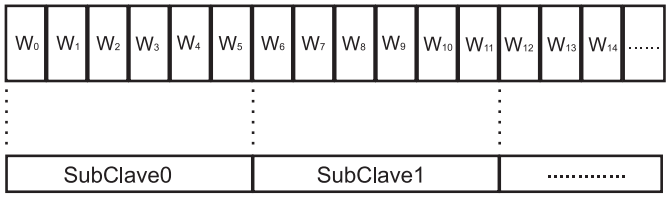

Figura 2.7: Claves expandidas para N_b y N_k

La función de expansión de la clave permite generar bytes útiles como subclaves a partir de la clave principal k, esta función se puede describir como un arreglo denominado W de palabras de 4 bytes y longitud $N_b(N_r + 1)$. Para AES se tiene que $N_b = N_k = 4$ y $N_r = 4$ (Joan 2002).

$$SubClav = N_b * (N_r + 1) \tag{2.29}$$

Donde la clave k_i se expande a una matriz de 4 filas y $N_b * (N_r + 1)$ columnas.

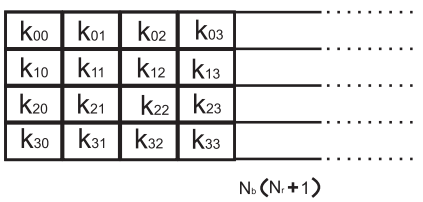

Figura 2.8: Matriz de expansión de la llave

Los siguientes cuadros muestran los parámetros de las claves para 128,192 y 256 bits.

Tamaño de la Clave	Numero de rondas	Numero de Sub-claves
4	10	1408
6	12	1664
8	14	1920

Cuadro 2.5: Número de claves y Sub-claves para bloques de 128

Tamaño de la Clave	Numero de rondas	Numero de Sub-claves
4	12	2496
6	12	2496
8	14	2880

Cuadro 2.6: Número de claves y Sub-claves para bloques de 192

Tamaño de la Clave	Numero de rondas	Numero de Sub-claves
4	14	3840
6	14	3840
8	14	3840

Cuadro 2.7: Número de claves y Sub-claves para bloques de 256

Para generar la subclave se realizan varias operaciones.

Dado que el usuario proporciona una clave aleatoria de 128 bits, que es el caso del algoritmo AES, (Joan 2002) se obtiene un vector de tamaño 16 a la entrada del generador de las subclaves, quedando de la siguiente forma:

$$(k_{00}, k_{10}, k_{20}, k_{30}, k_{10}, k_{11}, k_{21}, k_{31}, k_{02}, k_{12}, k_{22}, k_{32}, k_{03}, k_{13}, k_{23}, k_{33}) \tag{2.30}$$

La clave k_i se expande a una matriz de 4 filas y $N_b(N_r + 1) = 44$ columnas.

$$\begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} \tag{2.31}$$

En AES se toman los parámetros $N_b = N_k = 4$ y $N_r = 10$. Esto es, que de acuerdo con las condiciones del algoritmo se tienen los casos en que i toma el valor:

$$0 \leq i \leq 4$$

$$W[i] = (\text{key}[4 * i], \text{key}[4 * i + 1]), \text{key}[4 * i + 2], \text{key}[4 * i + 3]) \tag{2.32}$$

De acuerdo a la ecuación 2.32 se obtienen los siguientes valores.

Para $i = 0, 1, 2, ..$

$$\begin{aligned} W[0] &= (\text{key}[0], \text{key}[1]), \text{key}[2], \text{key}[3]) \\ W[1] &= (\text{key}[4], \text{key}[5]), \text{key}[6], \text{key}[7]) \\ W[2] &= (\text{key}[8], \text{key}[9]), \text{key}[10], \text{key}[11]) \\ W[3] &= (\text{key}[12], \text{key}[13]), \text{key}[14], \text{key}[15]) \end{aligned} \tag{2.33}$$

Los cuales se representan de la forma :

$$\begin{bmatrix} \text{key}[0] & \text{key}[4] & \text{key}[8] & \text{key}[12] \\ \text{key}[1] & \text{key}[5] & \text{key}[9] & \text{key}[13] \\ \text{key}[2] & \text{key}[6] & \text{key}[10] & \text{key}[14] \\ \text{key}[3] & \text{key}[7] & \text{key}[11] & \text{key}[15] \end{bmatrix} \tag{2.34}$$

Por lo que el vector representado en (2.30), queda distribuido de la siguiente manera:

$$\begin{aligned} \text{key}[0] &= (k_{00}, k_{10}, k_{20}, k_{30}) \\ \text{key}[1] &= (k_{10}, k_{11}, k_{21}, k_{31}) \\ \text{key}[2] &= (k_{02}, k_{12}, k_{22}, k_{32}) \\ \text{key}[3] &= (k_{03}, k_{13}, k_{23}, k_{33}) \end{aligned} \tag{2.35}$$

Los vectores key[i] se colocan en una matriz donde los componentes forman columnas de la siguiente manera

$$\begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} \tag{2.36}$$

Para obtener los valores de la expansión $(W[4], W[5], ..., W[15])$, hay que considerar un caso especial donde i es múltiplo de 4. En caso contrario se realiza la operación $W[i+1]\text{xor}W[i+4]$, por lo que la expansión de la clave se realiza de la siguiente manera.

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] & \\ k_{00} & k_{01} & k_{02} & & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} & \downarrow \\ k_{20} & k_{21} & k_{22} & k_{23} & \\ k_{30} & k_{31} & k_{32} & k_{33} & \end{bmatrix} \longrightarrow \begin{bmatrix} W[0] & W[1] & W[2] & W[3] & \\ k_{00} & k_{01} & k_{02} & k_{13} & \\ k_{10} & k_{11} & k_{12} & k_{23} & \\ k_{20} & k_{21} & k_{22} & k_{33} & \downarrow \\ k_{30} & k_{31} & k_{32} & k_{03} & \end{bmatrix} \tag{2.37}$$

Ahora se obtiene el inverso $W[3] \rightarrow W[3]^{-1}$ de acuerdo a la Tabla C.1, que es la matriz S Box del AES.

$$\begin{bmatrix} k_{13} \\ k_{23} \\ k_{33} \\ k_{03} \end{bmatrix} \rightarrow \begin{bmatrix} k_{13}^{-1} \\ k_{23}^{-1} \\ k_{33}^{-1} \\ k_{03}^{-1} \end{bmatrix} \tag{2.38}$$

Tomando como ejemplo una clave de 128 bits (AES) se obtiene el vector $[k_{ij}]$, que tiene la siguiente clave.

$$clave = 2b, 7e, 15, 16, 28, ae, d2, a6, ab, f7, 15, 88, 09, cf, af, 3c$$

Tomando en cuenta la ecuación (2.30) y colocados de acuerdo a (2.35) tenemos:

$$W[0] = (2b, 7e, 15, 16)$$
$$W[1] = (28, ae, d2, a6)$$
$$W[2] = (ab, f7, 15, 88)$$
$$W[3] = (09, cf, 4f, 3c)$$

Por lo que se tendrá:

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2b & 28 & ab & 09 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{bmatrix}$$

Aplicando los corrimientos propuestos en 2.37 tenemos que:

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2b & 28 & ab & cf \\ 7e & ae & f7 & 4f \\ 15 & d2 & 15 & 3c \\ 16 & a6 & 88 & 09 \end{bmatrix}$$

Aplicando la función sustitución (*SubByte*) que es la substitución de byte, de acuerdo a C.1. Esto es que para obtener el primer término de $W[3]$, se toma el termino de la intersección de la fila c con la columna f de la tabla C.1, en caso de $cf = 8a$ y de acuerdo a C.1 se tiene:

$$\begin{bmatrix} W[3] \\ cf \\ 4f \\ 3c \\ 09 \end{bmatrix} \rightarrow \begin{bmatrix} W[3]^{-1} \\ 8a \\ 84 \\ eb \\ 01 \end{bmatrix}$$

Una vez obtenido el inverso se aplica la operacion XOR, a este movimiento se le denomina como *Rcon* propuesto por (Joan 2002) donde esta función se define:

$$Rcon[i] = (RC[i], 00, 00, 00) \tag{2.39}$$

Con

$$Rcon[1] = 1 \tag{2.40}$$

$$Rcon[i] = x \bullet Rc[i-1] = x^{i-1} \tag{2.41}$$

Es decir, desde el punto de vista de un polinomio se tienen que aplicar las propiedades de los campos $GF(2^8)$ (apéndices A y B):

$$\begin{aligned}
\text{RC}[1] &= 1 \\
\text{RC}[2] &= x = 2 \\
\text{RC}[3] &= x^2 = 4 \\
\text{RC}[4] &= x^3 = 8 \\
\text{RC}[5] &= x^4 = 10 \\
\text{RC}[6] &= x^5 = 20 \\
\text{RC}[7] &= x^6 = 40 \\
\text{RC}[8] &= x^7 = 80 \\
\text{RC}[9] &= x^8 = x^4 + x^3 + x + 1 = 1B \\
\text{RC}[10] &= x^9 = x^5 + x^4 + x^2 + x + x = 36
\end{aligned} \tag{2.42}$$

Así la operación XOR queda de la siguiente manera:

$$\begin{bmatrix} 8A \\ 84 \\ EB \\ 01 \end{bmatrix} \oplus \begin{bmatrix} 01 \\ 00 \\ 00 \\ 00 \end{bmatrix} = \begin{bmatrix} 8B \\ 84 \\ EB \\ 01 \end{bmatrix}$$

Finalmente, para obtener la primera extensión debemos realizar una operación XOR con $W[i-4]$.

$$\begin{bmatrix} 2B \\ 7E \\ 15 \\ 16 \end{bmatrix} \oplus \begin{bmatrix} 8B \\ 84 \\ EB \\ 01 \end{bmatrix} = \begin{bmatrix} A0 \\ FA \\ FE \\ 17 \end{bmatrix}$$

Una vez que hemos obtenido $W[4]$ de la clave proporcionada por el usuario, las siguientes columnas $i = 5, 6, 7$ no son múltiplos de 4, por lo que su cálculo se reduce a una función XOR con la columna $W[i-4]$, esto es :

$$\begin{bmatrix} 8a \\ 84 \\ eb \\ 01 \end{bmatrix} \oplus \begin{bmatrix} 01 \\ 00 \\ 00 \\ 00 \end{bmatrix} = \begin{bmatrix} 8b \\ 84 \\ eb \\ 01 \end{bmatrix}$$

Por lo que la clave k_i expandida para múltiplos diferentes de 4 es:

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] & W[4] \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2b & 28 & ab & 09 & a0 \\ 7e & ae & f7 & cf & fa \\ 15 & d2 & 15 & 4f & fe \\ 16 & a6 & 88 & 3c & 17 \end{bmatrix}$$

Para poder obtener $W[5]$ se realiza la operación $W[i] = [i-4]\text{XOR}W[i-1]$.

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] & W[4] \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2b & 28 & ab & 09 & a0 \\ 7e & ae & f7 & cf & fa \\ 15 & d2 & 15 & 4f & fe \\ 16 & a6 & 88 & 3c & 17 \end{bmatrix} \longrightarrow \begin{bmatrix} W[5] \\ \downarrow \\ 88 \\ 54 \\ 2c \\ b1 \end{bmatrix} = \begin{bmatrix} W[1] \\ \downarrow \\ 28 \\ ae \\ d2 \\ a6 \end{bmatrix} \oplus \begin{bmatrix} W[4] \\ \downarrow \\ a0 \\ fa \\ fe \\ 17 \end{bmatrix}$$

Para $W[6]$ se tendrá:

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] & W[4] & W[5] \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2b & 28 & ab & 09 & a0 & 88 \\ 7e & ae & f7 & cf & fa & 54 \\ 15 & d2 & 15 & 4f & fe & 2c \\ 16 & a6 & 88 & 3c & 17 & b1 \end{bmatrix} \longrightarrow \begin{bmatrix} W[6] \\ \downarrow \\ 23 \\ a3 \\ 39 \\ 39 \end{bmatrix} = \begin{bmatrix} W[2] \\ \downarrow \\ ab \\ f7 \\ 15 \\ 88 \end{bmatrix} \oplus \begin{bmatrix} W[5] \\ \downarrow \\ 88 \\ 54 \\ 2c \\ b1 \end{bmatrix}$$

Para $W[7]$:

$$\begin{bmatrix} W[0] & W[1] & W[2] & W[3] & W[4] & W[5] & W[6] \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2b & 28 & ab & 09 & a0 & 88 & 23 \\ 7e & ae & f7 & cf & fa & 54 & a3 \\ 15 & d2 & 15 & 4f & fe & 2c & 39 \\ 16 & a6 & 88 & 3c & 17 & b1 & 39 \end{bmatrix} \longrightarrow \begin{bmatrix} W[7] \\ \downarrow \\ 2a \\ 6c \\ 76 \\ 05 \end{bmatrix} = \begin{bmatrix} W[3] \\ \downarrow \\ 03 \\ cf \\ 4f \\ 3c \end{bmatrix} \oplus \begin{bmatrix} W[6] \\ \downarrow \\ 23 \\ a3 \\ 39 \\ 39 \end{bmatrix}$$

Para obtener $W[8]$ se tiene que realizar todo el procedimiento descrito para $W[4]$ por ser múltiplo de cuatro.

Para el caso de AES(Joan 2002), se considera $N_b = N_k = 4$ y $N_r = 10$ por lo que al sustituirlos en la ecuación (2.29), se tendran 44 subclaves representadas en la matriz de [4x44].

2.7. Numero de rondas.

El funcionamiento del algoritmo *AES* consiste en una serie de rondas formadas por las cuatro transformaciones descritas con anterioridad.

Como ya se señaló a mayor número de rondas aumenta el nivel de seguridad frente a los ataque mencionados, sin embargo aun con 6 rondas se tiene una buena seguridad (Joan 2002).

Si bien con dos rondas se produce una difusión completa en el sentido que, cada bit depende de las dos rondas anteriores. Es decir un cambio en un bit es similar a cambiar la mitad de los bits de un estado después de dos rondas.

Para claves mayores el numero de rondas se incrementa de una por cada 32 bits adicionales de la clave de cifrado, en el caso de bloques mayores a 128 bits el numero de rondas aumenta una por cada 32 bits adicionales (Muñoz 2004).

Capítulo 3

Implementación del algoritmo AES.

La implementación del algoritmo *AES-Rijndael*, se realizada en el lenguaje ANSI C por su facilidad, para posteriormente instalarlo en un servidor, a través de una aplicación WEB desarrollada mediante lenguaje HTML, y así poder transmitir datos a nivel de red.

Para implementar este algoritmo se debe considerar que en muchos cifradores de bloque la operación de descifrado no se realiza.

En una ronda se aplican las cuatro transformaciones que permiten cifrar el texto, cada ciclo adicional que el cifrador usa puede dar más seguridad; sin embargo la velocidad para generar el texto cifrado disminuye. El número apropiado de rondas lo determina el diseñador del cifrado mediante pruebas empíricas.

Para el proceso de cifrado, se tiene $x = x_1, x_2, ..., x_n$, donde x_n es un texto a cifrar del bloque x del texto. Por lo que al aplicar las cuatro transformaciones; adición de la clave(*AddRoundKey*), Sustitución (*SubByte*), desplazamiento de filas (*ShiftRows*) y multiplicación de columnas (*MixColumns*) con la clave k al bloque x_i, descritas en el capitulo 2,se genera a la salida el vector

$$c_(x_i) = e(k, x_i)$$

.

Un bloque cifrado puede operar de cuatro diferentes formas:

• Modo *Electronic code Block (ECB)*: en este modo se aplican las transformaciones a cada bloque y se concatenan los resultados en el mismo orden para obtener el texto cifrado.

$$c_i = e_k(x_i) \text{ con } c = c_1 + c_2$$

• Modo *Cipher Block Chaining (CBC)*: en este modo se obtiene el primer bloque cifrado c_1 de la siguiente forma:

$$c_1 = e_k(x_i \oplus x_0)$$

Para los siguientes c_i:

$$c_i = e_k(x_i \oplus c_{i-1})$$

Y el texto cifrado sera:

$$c = c_1 + c_2$$

• Modo *Cipher FeedBack (CFB)*: en este modo se obtiene el primer bloque cifrado C_1 de la siguiente forma:

$$c_i = e_k(x_0) \text{ con } c_1 = x_1 \oplus z_1$$

Para los siguientes:

$$z_i = e_k(c_{i-1}) \text{ con } c_i = x_i \oplus z_i$$

Y el texto cifrado sera.

$$c = c_1 + c_2$$

• Modo *Output FeedBack (OFB)*: en este modo se obtiene el primer bloque cifrado C_1 de la siguiente forma:

$$z_1 = e_k(x_0) \text{ con } c_1 = x_1 \oplus z_1$$

para los siguientes:

$$z_i = e_k(z_{i-1}) \text{ con } c_i = x_i \oplus z_i$$

Y el texto cifrado sera:

$$c = c_1 + c_2$$

En el cuadro 3.1 se muestra los tiempos obtenidos en la ejecución de *AES-Rijndael* bajo procesadores de 32 bits (Counterpane. 2002). Las columnas de la izquierda hacen referencia a la ejecución de *AES-Rijndael* en una máquina equipada con un procesador Pentium III a 200 Mhz con sistema operativo Windows. La implementación corresponde a un código ANSI C del algoritmo *AES-Rijndael*. Las columnas de la derecha hacen referencia a la ejecución de *Rijndael* en máquinas que incorporaran procesadores de la familia Pentium Pro 200 Mhz, aunque en estos tiempos se pueden exportar a máquinas con procesadores Pentium iV.

Longitud clave/bloque	Numero de ciclos	RAM (Necesaria)	Longitud código
(128,128)	8390 ciclos	36 bytes	919 bytes
(192,128)	10780 ciclos	44 bytes	1170 bytes
(256,128)	12490 ciclos	52 bytes	1135 bytes

Cuadro 3.1: Especificaciones de AES

Se debe tomar en cuenta el tiempo que se necesita en el proceso de obtención de las subclaves, ya que el tiempo de inicialización del algoritmo, de la clave y cambio del valor ésta serán similares entre ambas pruebas.

Longitud clave/bloque	velocidad (Mbits/seg)	Num. ciclos/bloque
(128,128)	27.0	950
(192,128)	22.8	1125
(256,128)	19.8	1295

Cuadro 3.2: Especificaciones de AES en lenguaje C

El proceso de cifrado consiste en la aplicación de 4 transformaciones, adición de la clave (*AddRoundKey*), sustitución (*SubByte*), desplazamiento de filas(*ShiftRows*) y Multiplicación de columnas (*MixColumns*) sobre la información que se desea cifrar.

Estas transformaciones se realizan de forma reiterativa para cada ronda. Las cuales se representan en el siguiente diagrama.

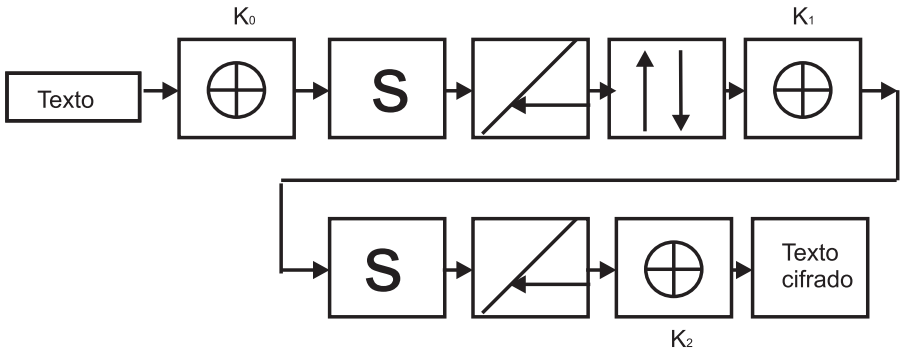

Figura 3.1: Diagrama simple AES

3.1. Cifrado.

El cifrador Rijndael se puede estructurar de la siguiente manera:

- Una operación adición de la clave (*AddRoundKey*) inicial.
- $N_r - 1$ rondas.
- Ronda Final.

Su pseudocódigo en C es:

```
int EncryptBlock (word8 a[4][MAXBC], word8
rk[MAXROUNDS+1][4][MAXBC]) {
     /* Cifrado de un bloque */
     int r;
     /* Empieza con la adicion de una clave */
     AddRoundKey(a,rk[0]);
     /* ROUNDS-1 rondas comunes */
     for(r = 1; r < ROUNDS; r++)
     {
           SubBytes(a,S);
           ShiftRows(a,0);
           MixColumns(a);
           AddRoundKey(a,rk[r]);
     }
     /* La ultima ronda es especial: no hay MixColumns */
     SubBytes(a,S);
```

```
     ShiftRows(a,0);
     AddRoundKey(a,rk[ROUNDS]);
     return 0;
}
```

Una variante consistiría en hallar la subclave de manera previa a la ejecución del algoritmo *AES-Rijndael*. La ventaja de esta variante está en lograr expresar el algoritmo en función de una sola clave, con lo que se lograría homogeneidad en el código y facilitará la comprensión del mismo. La desventaja es que se pierde modularidad, el algoritmo no sería tan fácilmente exportable, ya que no sería un código cerrado. Lo cual se verá a continuación.

3.2. Función Adición de la clave (*AddRoundKey*).

De acuerdo a la sección 2.2, la primera transformación que se aplica es la llamada adición de la clave (*KeyAddition*), esto es aplicar la operación $[a_{ij}] \oplus [k_{ij}]$(ecuación 2.8). La cual contiene los vector de entrada.

Esta transformación recibe la matriz $[a_{ij}]$ y $[k_{ij}]$, en donde $N_r = BC = 4$, y regresa los valores de la matriz $[A_{i,j}]$ haciendo una operación XOR.

Para realizar la operación XOR, el usuario proporciona una clave la cual tendrá las características mencionadas en la sección 2.6, por lo que la expansión de las subclaves se calculan con el siguiente pseudocódigo en C:

```
void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
     /* XOR correspondiente al texto de entrada y a los bytes de entrada del
      * round key
      */
     int i, j;
     for(i = 0; i < 4; i++)
          for(j = 0; j < BC; j++)
               a[i][j] ^= rk[i][j];
}
```

3.3. Sustitución (*SubByte*).

Como se menciono en la sección 2.3 sólo hay que reemplazar cada byte de la martrz $[A_{i,j}]$ por el correspondiente valor en la matriz $S - Box$, es decir:

Sbox(00)=S[0], box(01)=S[1], box(11)=S[17], como se ve en la siguiente función:

```
void SubBytes(word8 a[4][MAXBC], word8 box[256]) {
     /* Reemplaza cada byte de la entrada por el byte en esa posicion en la
      * S-box no lineal
      */
     int i, j;
     for(i = 0; i < 4; i++)
          for(j = 0; j < BC; j++)
               a[i][j] = box[a[i][j]];
}
```

unsigned char S[256] = [99, 124, 119, 123, 242, 107, 111, 197, 48, 1, 103, 43, 254, 215, 171, 118, 202, 130, 201, 125, 250, 89, 71, 240, 173, 212, 162, 175, 156, 164, 114, 192, 183, 253, 147, 38, 54, 63, 247, 204, 52, 165, 229, 241, 113, 216, 49, 21, 4, 199, 35, 195, 24, 150, 5, 154, 7, 18, 128, 226, 235, 39, 178, 117, 9, 131, 44, 26, 27, 110, 90, 160, 82, 59, 214, 179, 41, 227, 47, 132, 83, 209, 0, 237, 32, 252, 177, 91, 106, 203, 190, 57, 74, 76, 88, 207, 208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2, 127, 80, 60, 159, 168, 81, 163, 64, 143, 146, 157, 56, 245, 188, 182, 218, 33, 16, 255, 243, 210, 205, 12, 19, 236, 95, 151, 68, 23, 196, 167, 126, 61, 100, 93, 25, 115, 96, 129, 79, 220, 34, 42, 144, 136, 70, 238, 184, 20, 222, 94, 11, 219, 224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172, 98, 145, 149, 228, 121, 231, 200, 55, 109, 141, 213, 78, 169, 108, 86, 244, 234, 101, 122, 174, 8, 186, 120, 37, 46, 28, 166, 180, 198, 232, 221, 116, 31, 75, 189, 139, 138, 112, 62, 181, 102, 72, 3, 246, 14, 97, 53, 87, 185, 134, 193, 29, 158, 225, 248, 152, 17, 105, 217, 142, 148, 155, 30, 135, 233, 206, 85, 40, 223, 140, 161, 137, 13, 191, 230, 66, 104, 65, 153, 45, 15, 176, 84, 187, 22,] ;

Observamos que en la primera fila están colocados los elementos de 0 al 15 en hexadecimal, en la segunda del 00 al 0f, en la tercera del 30,..,3f, en la cuarta 40,..,4f, en la i-ésima i0,..,if, en la última, f0,..,ff, respectivamente, entonces el valor $S - Box$ correspondiente a $a[i][j] = xy$ en hexadecimal, que es el valor que está en la fila x y la columna y.

Lo importante de esta parte es conocer la manera en que se construye la matriz *Sbox*(ver apéndice C.1). La descripción de AES dice que la transformación sustitución *SubByte* $a[i][j]$ es igual a el resultado de aplicar dos funciones, primero como

$a[i][j]$ es un elemento de campo finito $GF(2^8)$, se le asocia su inverso multiplicativo, posteriormente se aplica una función lineal.

3.4. Desplazamiento de filas (*ShiftRow*).

En esta transformación descrita en la sección 2.4 se aplica a toda la matriz $[A_{i,j}]$ del estado intermedio 1. Dejando la primera fila igual, a la segunda fila aplica un corrimiento izquierdo de bytes circular, a la tercera fila se aplica 2 corrimientos izquierdos de bytes circulares, y a la cuarta fila se aplican 3 bytes de corrimiento (cuadro 3.7).

```
void ShiftRows(word8 a[4][MAXBC], word8 d) {
     /* El renglon 0 permanece sin cambios
      * Los otros tres renglones son rotados cierta cantidad variable
      */
     word8 tmp[MAXBC];
     int i, j;
     if (d == 0)
     {
           for(i = 1; i < 4; i++)
           {
                 for(j = 0; j < BC; j++)
                      tmp[j] = a[i][(j + shifts[BC-4][i]) % BC];
                 for(j = 0; j < BC; j++)
                      a[i][j] = tmp[j];
           }
     }
     else
     {
           for(i = 1; i < 4; i++)
           {
                 for(j = 0; j < BC; j++)
                      tmp[j] = a[i][(BC + j - shifts[BC-4][i]) % BC];
                 for(j = 0; j < BC; j++)
                      a[i][j] = tmp[j];
           }
     }
}
```

3.5. Multiplicación de columnas (*Mixcolumns*).

En esta transformación es considerado un polinomio en $GF(2^8)$ y multiplicado por el polinomio constante $C(x) = x^4 + 1$ (ver sección 2.5):

```
void MixColumns(word8 a[4][MAXBC]) {
     /* Mezcla los cuatro bytes de cada columna en una forma lineal */
     word8 b[4][MAXBC];
     int i, j;
     for(j = 0; j < BC; j++)
          for(i = 0; i < 4; i++)
               b[i][j] = mul(2,a[i][j])
                       ^ mul(3,a[(i + 1) % 4][j])
                       ^ a[(i + 2) % 4][j]
                       ^ a[(i + 3) % 4][j];
     for(i = 0; i < 4; i++)
          for(j = 0; j < BC; j++)
               a[i][j] = b[i][j];
}
```

El XOR representa la suma, posteriormente se hace lo mismo con las siguientes columnas haciendo un corrimiento circular de un byte al primer renglón, lo que corresponde: $i, i+1, i+2, i+3 \bmod 4$.

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

3.5.1. Multiplicación *mul*.

Hemos observado que en el anterior código se hace uso de la función *mul*, que recibe 2 bytes y regresa el producto de ellos en el campo $GF(2^8)$.

```
word8 mul(word8 a, word8 b) {
     /* multiplica dos elementos de GF(256) requerido por MixColumns y
      * InvMixColumns
      */
     if (a && b)
          return Alogtable[(Logtable[a] + Logtable[b])%255];
     else
```

```
        return 0;
}
```

La función anterior hace uso de dos tablas, *Logtable[]*, y *Alogtable[]*, la idea es multiplicar dos elementos del campo finito $GF(2^8)$, se hace uso de los logaritmos y antilogaritmos, es decir la tabla *Logtable* tiene todos los logaritmos de todos los elementos del campo finito $GF(2^8)$, que son 256.

```
/* Definición de los Logaritmos de todos los elementos de GF(2^8)
base 3 */ unsigned char Logtable[256] = { 0, 0, 25, 1, 50, 2, 26,
198, 75, 199, 27, 104, 51, 238, 223, 3, 100, 4, 224, 14, 52, 141,
129, 239, 76, 113, 8, 200, 248, 105, 28, 193, 125, 194, 29, 181,
249, 185, 39, 106, 77, 228, 166, 114, 154, 201, 9, 120, 101, 47,
138, 5, 33, 15, 225, 36, 18, 240, 130, 69, 53, 147, 218, 142, 150,
143, 219, 189, 54, 208, 206, 148, 19, 92, 210, 241, 64, 70, 131, 56,
102, 221, 253, 48, 191, 6, 139, 98, 179, 37, 226, 152, 34, 136, 145,
16, 126, 110, 72, 195, 163, 182, 30, 66, 58, 107, 40, 84, 250, 133,
1, 186, 43, 121, 10, 21, 155, 159, 94, 202, 78, 212, 172, 229, 243,
115, 167, 87, 175, 88, 168, 80, 244, 234, 214, 116, 79, 174, 233,
213, 231, 230, 173, 232, 44, 215, 117, 122, 235, 22, 11, 245, 89,
203, 95, 176, 156, 169, 81, 160, 127, 12, 246, 111, 23, 196, 73,
236, 216, 67, 31, 45, 164, 118, 123, 183, 204, 187, 62, 90, 251, 96,
177, 134, 59, 82, 161, 108, 170, 85, 41, 157, 151, 178, 135, 144,
97, 190, 220, 252, 188, 149, 207, 205, 55, 63, 91, 209, 83, 57, 132,
60, 65, 162, 109, 71, 20, 42, 158, 93, 86, 242, 211, 171, 68, 17,
146, 217, 35, 32, 46, 137, 180, 124, 184, 38, 119, 153, 227, 165,
103, 74, 237, 222, 197, 49, 254, 24, 13, 99, 140, 128, 192, 247,
112, 7, };
```

3.6. Generación de las subclaves (*Key Expansion*).

La subclave proviene de la clave principal de cifrado (sección 2.6). Ya que esta transformación recibe la clave K con la que se construye la matriz $k_{i,j}$ y regresa un arreglo de 4 x 44 ($W[N_b * (N_r + 1)] = 44$). Estas subclaves se puede crear en memoria usando un buffer de N_k palabras y así ahorrar trabajo de computación en implementaciones donde la memoria RAM sea escasa. Las primeras N_k palabras contienen la clave de cifrado principal.

Las restantes palabras son definidas recursivamente en términos de palabras con índices cada vez más pequeños. La transformación subclave depende del valor de N_k (número de columnas de la matriz representante de la clave, o lo que es lo mismo, número de palabras de 4 bytes que contiene la clave).

Rijndael diferencia el caso en el que $Nk \leq 6$ del caso $Nk > 6$ y aplica distintos procesos.

Caso $Nk \leq 6$

Recordemos que la transformación sustitución ($SubByte$) es una función que devuelve palabras de 4 bytes en las cuales los bytes han sido permutados. La función $RotByte(W)$ devuelve una palabra en la que los bytes han sufrido una permutación cíclica de forma que una palabra (a, b, c, d) pasaría a ser la palabra (b, c, d, a), o lo que es lo mismo, los bytes se desplazan una posición a la izquierda.

Se puede apreciar en el pseudocódigo que las primeras N_k palabras de la clave expandida están formadas por la clave principal. Supongamos que queremos hallar la palabra de la posición i, es decir, $W[i]$. Realizamos una operación $EXOR$ entre la palabra anterior a la que queremos hallar $(W[i-1])$ y la palabra situada N_k posiciones atrás $(W[i-N_k])$. Si la posición es múltiplo de N_k, antes de aplicar esta función se lleva a cabo otra transformación que se aplica a $W[i-1]$. Esta transformación consiste en la aplicación de la función $RotByte(W)$ seguida de la aplicación de la función sustitución ($SubByte$) a los bytes de esa palabra. Para finalizar se hace una función $EXOR$ con una constante.

```
int KeyExpansion (word8 k[4][MAXKC], word8 W[MAXROUNDS+1][4][MAXBC])
{
     /* Calcula las round keys requeridas */
     int i, j, t, RCpointer = 1;
     word8 tk[4][MAXKC];
     for(j = 0; j < KC; j++)
          for(i = 0; i < 4; i++)
               tk[i][j] = k[i][j];
     t = 0;
     /* copia los valores en el arreglo round key */
     for(j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++)
          for(i = 0; i < 4; i++)
               W[t / BC][i][t % BC] = tk[i][j];
```

```
    while (t < (ROUNDS+1)*BC)
    {
          /* mientras no sea suficiente lo calculado por round key,
           * calcular nuevos valores
           */
          for(i = 0; i < 4; i++)
               tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
          tk[0][0] ^= RC[RCpointer++];
          if (KC <= 6)
               for(j = 1; j < KC; j++)
                    for(i = 0; i < 4; i++)
                         tk[i][j] ^= tk[i][j-1];
          else
          {
               for(j = 1; j < 4; j++)
                    for(i = 0; i < 4; i++)
                         tk[i][j] ^= tk[i][j-1];
               for(i = 0; i < 4; i++)
                    tk[i][4] ^= S[tk[i][3]];
               for(j = 5; j < KC; j++)
                    for(i = 0; i < 4; i++)
                         tk[i][j] ^= tk[i][j-1];
          }
          /* copia valores en el arreglo round key */
          for(j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++)
               for(i = 0; i < 4; i++)
                    W[t / BC][i][t % BC] = tk[i][j];
    }
    return 0;
}
```

Caso $Nk > 6$

La diferencia radica en que la transformación aplicada en el caso anterior a $W[i-1]$ cuando la posición i sera múltiplo de N_k se aplica también ahora en el caso de que $i-4$ sea múltiplo de N_k. Las constantes empleadas son independientes del valor de N_k. Se pueden definir en forma de vector de la siguiente manera:

$Rcon[i] = (RC[i], 00, 00, 00)$

donde $RC[i]$ es un elemento perteneciente a $GF(2^8)$ con un valor de $x^{(i-1)}$ o lo que es lo mismo:

$RC[1] = 1$ $RC[i] = x \cdot (RC[i-1]) = x^{(i-1)}$

3.6.1. Función de expansión de la clave (*Round Key Selection)*

Las claves secundarias proceden de la clave expandida. Para seleccionar la subclave i se seleccionan las palabras de la clave expandida.

$$W[i] = (\text{key}[4 * i], \text{key}[4 * i + 1]), \text{key}[4 * i + 2], \text{key}[4 * i + 3]) \tag{3.1}$$

3.7. Descifrado

La implementación del descifrador *AES-Rijndael* es igual al cifrado (ver sección 3.1), ya que tiene una estructura similar, solo que las transformaciones serán inversas por lo que se tendrá, la transformación inverza de la multiplicación de columnas (*invMixColumn*) que emplea un polinomio $D(x)$, el cual cumple con la ecuación 2.25.

La implementación se hace mediante tablas de valores $S - Box$ inverza, es esencial que el único paso no lineal de la operación sustitución (*invSubByte*) sea la primera transformación en cada ronda y que los byte de las filas sean desplazados (*invShiftRow*) antes de aplicar la transformación multiplicación de columnas (*invMixColumn*). En la inversa de cada ronda, el orden de las transformaciones es el contrario al orden de las operaciones establecidas, por tanto, el paso no lineal será el último del proceso y los bytes de las filas son permutados después de aplicar la inversa de la transformación multiplicación de columnas (*invMixColumn*).

La inversa de cada ronda no se puede implementar con las tablas de valores del proceso de cifrado. Esto es una característica del propio diseño de *AES-Rijndael*(Joan 2002). La estructura interna de *AES-Rijndael* es tal que la secuencia de transformaciones de la inversa es igual al proceso de cifrado en sí, con las transformaciones reemplazadas por sus propias inversas y un cambio en la extensión de las subclaves (*Key Schedule*).

3.7.1. Propiedades del Descifrado.

Para obtener el descifrador *AES-Rijndael* se ha hecho uso de propiedades algebraicas de la composición de transformaciones.

• En primer lugar, el orden de las operaciones desplazamiento de filas (*ShiftRow*) y sustitución (*SubByte*) es indiferente. Esto es debido a que la operación desplazamiento de filas (*ShiftRow*) afecta solo la posición de los byte, pero no modifica su valor. En el caso de sustitución (*ByteSub*), el campo de trabajo es el valor de los bytes, independientemente de cuál sea su posición en la matriz estado(Muñoz 2004).

Por lo que se puede modificar la secuencia de las trasformaciones sin perder la fortaleza del cifrado, y así se pueden realizar los siguientes cambios (Joan 2002) .

```
AddRoundKey (State, RoundKey);
InvMixColumn (State);
```

Es reemplazada por:

```
InvMixColumn (State);
AddRoundKey (State, InvRoundKey);
```

La transformación de adición de la clave (*InvRoundKey*) se halla aplicando la transformación inversa de multiplicación de columnas (*InvMixColumn*) a la subclave *subclav*.

Este reemplazo se puede hacer tomando como base una propiedad algebraica de las aplicaciones lineales. Dada una aplicación A, se puede demostrar que $A(x+k) = A(x) + A(k)$ (Pais 2003).

Por lo que se reescribir el descifrador *AES-Rijndael* como:

```
AddRoundKey(State, ExpandedKey + 2 * Nb);
InvByteSub(State);
InvShiftRow(State);
InvMixColumn(State);
AddRoundKey(State,
InvExpandedKey + Nb);
InvByteSub(State);
InvShiftRow(State);
AddRoundKey(State, ExpandedKey);
```

Retomando la estructura del algoritmo *AES-Rijndael* (Capítulo 2), se tiene la transformación *AddRoundKey* al inicio y una ronda final. Esta estructura tienen la misma estructura en el descifrador que el cifrador. Hay que notar que las rondas se escriben de la siguiente manera:

```
I_Round (State, I_RoundKey)
{
InvByteSub(State);
InvShiftRow(State);
InvMixColumn(State);
AddRoundKey(State, InvExpandedKey);
}
I_FinalRound (State, I_RoundKey)
{
InvByteSub(State);
InvShiftRow(State); AddRoundKey(State, ExpandedKey);
}
```

Visto esto, el descifrador *AES-Rijndael* puede ser expresado de la siguiente manera:

```
int DencryptBlock (word8 a[4][MAXBC], word8
rk[MAXROUNDS+1][4][MAXBC]) {
     int r;
     /* Para descifrar:
      * Aplicar las operaciones inversas de la rutina de cifrado, en orden
      * opuesto
      *
      * - AddRoundKey es igual a su inversa
      * - La inversa de SubBytes con la tabla S es SubBytes con la inversa
      *   de la tabla S
      * - La inversa de Shiftrows es Shiftrows sobre una distancia variable
      */
     /* Primero la ronda especial:
      * Sin InvMixColumns
      * Sin extra AddRoundKey
      */
     AddRoundKey(a,rk[ROUNDS]);
     SubBytes(a,Si);
     ShiftRows(a,1);
```

```
      /* ROUNDS-1 rondas comunes */
      for(r = ROUNDS-1; r > 0; r--)
      {
            AddRoundKey(a,rk[r]);
            InvMixColumns(a);
            SubBytes(a,Si);
            ShiftRows(a,1);
      }/* Terminar con la adicion de la clave extra */
      AddRoundKey(a,rk[0]);
      return 0;
}
```

La función expiación de clave (*KeyExpansion*) para el descifrado se compone de los siguientes pasos:

- Aplicación de la función expiación de clave (*KeyExpansion*).
- Aplicación de la función inversa de multiplicación de columnas (*InvMixColumn*) a todas las (*subclaves*) excepto a la primera y a la última.

```
I_KeyExpansion (CipherKey, I_ExpandedKey);
{

KeyExpansion (CipherKey, I_ExpandedKey);

InvMixColumn(I_ExpandedKey + Nb * i); }
```

3.8. Cambiar el orden de las rondas

La estructura del algoritmo *AES-Rijndael* es conocida, en lo que respecta a su estructura algebraica la cual se compone de las transformaciones mencionadas en el capitulo 2. Este numero de rondas fue determinado por los autores (Joan 2002). Ya que con un numero de rondas mínimo se ofrece una seguridad maxíma frente a ataques conocidos (apéndice D.7), ya que es seguro con al menos 8 rondas (Muñoz 2004).

La aplicación que acompaña a esta tesis se ha diseñado con el fin de comprender el algoritmo de cifrado *AES-Rijndael* y así proponer modificaciones en las rondas del algoritmo como la adición de la clave (*AddRoundKey*), sustitución (*SubByte*), desplazamiento de filas (*ShiftRows*), multiplicación de columnas (*MixColumns*), sin perder resistencias a los criptoanálisis existentes (ver apéndice D) con el fin de poder tener un proceso de cifrado resistente, en el cual se cuenta con un texto que desea cifrar-descifrar, junto con la clave.

Este tipo de aplicación permitirá comprobar muchas de las características de los algoritmos que han sido estudiadas en este trabajo, ofreciendo una demostración empírica de muchas ellas, algunas de las cuales son más apreciables desde el punto de vista de la ejecución del programa, como puede ser, la velocidad de cifrado del algoritmo ó resistencia a los criptoanálisis.

Un aspecto importante del algoritmo es el relacionado con los tiempos que tarda en cifrar o descifrar datos, ya que esto determinara su velocidad en siguiente cuadro 3.4, muestra estos aspectos los cuales fueron obtenidos por (Pais 2003).

Longitud	ANSI C		Visual C++	
	Cifrado	Descifrado	Cifrado	Descifrado
128	2100	2900	305	1389
192	2600	3600	277	1595
256	2800	3800	374	1960

Cuadro 3.3: Numero de ciclos necesarios para la exacción de la clave *AES*

Longitud	ANSI C [Mbits/seg]		Visual C++ [Mbits/seg]	
	Cifrado	Descifrado	Cifrado	Descifrado
128	27	950	70.5	363
192	22.8	1125	59.3	432
19.8	1295	8	51.2	500

Cuadro 3.4: Tiempos de ejecución *AES*

Como se vio en los capítulos anteriores se puede modificar este algoritmo y así poder obtener un algoritmo similar al propuesto por (Joan 2002), con los resultados publicados en (Pais 2003), (Jose de Jesus 2005) y (Muñoz 2004). Se puede tener un algoritmo fuerte y sobre todo que estos cambios no afectan mucho al algoritmo.

3.9. Modificación de la longitud de la clave

El tamaño del bloque de cifrado o la longitud de la clave, puede ser reducidos hasta 80 bits, esto significa tomar solo 3 columnas en lugar de 4, es decir tomar una longitud de 96 bits (Muñoz 2004). En el algoritmo *AES-Rijndael*, los parámetros de las claves son 128,192 y 256. Los siguientes cuadros muestral en numero de subclaves que se generan dependiendo al tamano de la clave y el numero de rondas.

Tamaño de la Clave	Numero de rondas	Numero de Sub-claves
4	10	1408
6	12	1664
8	14	1920

Cuadro 3.5: Número de claves y Sub-claves para bloques de 128

Tamaño de la Clave	Numero de rondas	Numero de Sub-claves
4	12	2496
6	12	2496
8	14	2880

Cuadro 3.6: Número de claves y Sub-claves para bloques de 192

Tamaño de la Clave	Numero de rondas	Numero de Sub-claves
4	14	3840
6	14	3840
8	14	3840

Cuadro 3.7: Número de claves y Sub-claves para bloques de 256

3.10. Cambio en las constante del algoritmo

Un estudio más completo lo dieron *E. Barkan*, y *E. Biham* (Elad 2002) al mostrar que si cambiamos todas las constantes involucradas en el algoritmo *AES-Rijndael* no tendremos alteración importante alguna, lo que da una gran posibilidad de elecciones. La razón principal para poder obtener estas modificaciones se debe a que la función cuadrado es lineal en el campo finito $GF(2^8)$.

Tomando en cuenta los siguientes apartados se pude lograr un buen cifrado. Para explicar esto con detalle damos las siguientes definiciones:

Dos algoritmos E y E' son Duales si son isomorfos, es decir, si existen funciones invertibles f, g y h tales que:
Para cualquier texto claro P y clave k, cumple que.

$$f(E_K(P)) = E'_{g(k)}(h(P)) \tag{3.2}$$

Particularmente si f,g,h son la función cuadrado entonces podemos mostrar que los algoritmos E, E^2 son duales, es decir son equivalentes, por lo que la ecuación 3.3.

$$E^2_{K_2}(P) = (E_k(P))^2 \tag{3.3}$$

Es decir que si usamos AES con los textos claros (que son elementos de un campo finito) al cuadrado, con la clave K al cuadrado y aplicamos E^2 , entonces el resultado sería el mismo al usar el original *AES-Rijndael* y elevar la salida al cuadrado. E^2 significa cambiar todas las constantes de E, deben de elevarse al cuadrado como elementos del campo finito $GF(2^8)$. Las variables involucradas en *AES-Rijndael* son:

1.- En la transformación sustitución (*SubByte*) la matriz y la constante de la aplicación lineal. Para la transformación afín $Ax+b$ la debemos cambiar por $QAQ^{-1}x+b^2$(Jose de Jesus 2005).

2.-En la multiplicación de columnas *MixColumns*, el polinomio $C(x) = 03x^3 + 01x^2 + 01x + 02$ debe ser cambiado por $Z(x) = 05x^3 + 01x^2 + 01x + 04$, en el proceso de descifrado el polinomio $D(x) = 0bx^3 + 0dx^2 + 09x + 0e$ se cambiar por $Q(x) = 0dx^3 + 0bx^2 + 0ex + 09$. Por lo que de acuerdo con la sección 2.5 se tiene:

$$S'(x) = Z(x) \oplus S(x) \tag{3.4}$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 04 & 05 & 01 & 01 \\ 01 & 04 & 05 & 01 \\ 01 & 01 & 04 & 05 \\ 05 & 01 & 01 & 04 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{3.5}$$

La transformada inversa (descifrado) queda según la sección 2.5 descrita como.

$$S(x) = Q(x) \oplus S'(x) \tag{3.6}$$

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} \tag{3.7}$$

Ya que los polinomios $Z(x)$ y $Q(x)$ cumplen con

$$Z(x) \oplus Q(x) = 01 \tag{3.8}$$

3.- En el programa de claves, las constantes *Rcon*. Se cambiarían las constantes de *Rcon* $(02)^{i-1}$ por las constantes $(03)^{i-1}$.

4.-El polinomio irreducible puede ser cambiado por alguno de los 30 que existen y que están listados en (Pais 2003).

Con los anteriores cambios se pueden lograr en principio 240 diferentes formas de AES. Considerando los diferentes polinomios del campo $GF(2^8)$ y sus diferentes subcampos podemos tener hasta 1170 representaciones diferentes de *AES-Rijndael* (Jose de Jesus 2005).

Capítulo 4

Programación de la implementación del algoritmo *AES-Rijndael.*

La eficiencia del algoritmo AES depende de su implementación, entre otras cosas. Se tiene entonces que eficiencia esta dividido en dos campos que son software y hardware.

El algoritmo *AES-Rijndael* es conocido ya que los autores muestran a detalle el funcionamiento de cada una de sus transformaciones básicas (Joan 2002). Este código tiene por objetivo ser el más eficiente, sin embargo analizando cada una de sus partes podemos saber cuales de ellas pueden ser optimizadas.

La operación para obtener los inversos multiplicativos del campo $GF(8^2)$(apéndice A) es la que requiere mayor tiempo computacional, en este caso la operación es calculada con anterioridad y substituida por un consulta a una matriz, la cual esta representada por $S - Box$, y se definen en C.1 y C.2.

Los autores al momento de diseñar el algoritmo definen que este debe cumplir con las siguientes características (Joan 2002):

I) Cifrado simétrico.
II) Cifra bloques de 128 bits.
III) Claves de 128,192 y 256 bits.
IV) Se utiliza un polinomio irreducible.

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

V) Los desplazamientos para AES con $N_b = 4$, son mostrados en el cuadro 2.3.

VI) En la transformación multiplicación de columnas se utiliza.

$$c(x) = 03x^3 + 01x^2 + 01x + 02$$

VII) El numero de rondas así como la generación de las subclaves es de 10

AES (*Advanced Encryption Standard*) es un algoritmo diseñado con ciertas características definidas por sus diseñadores ya que *Vincent Rijndael* y *Joan Daemen* definen que se pueden utilizar otros parámetros en su algoritmo, como los mencionados a lo largo de este trabajo (Joan 2002). Los cuales permiten modificar su estructura, sin perder de vista lo esencial del algoritmo, que es la seguridad.

4.1. Diseño de la aplicación.

El proceso de cifrado consiste, como ya se menciono, esencialmente en 4 transformaciones básicas : sustitución (*ByteSub*), desplazamiento de filas (*ShiftRow*), multiplicación de columnas (*MixColumns*), adición de la clave (*AddRoundKey*), y adicionalmente un proceso de extensión de la clave. Es importante mencionar que en el caso de *AES-Rijndael* las funciones o transformaciones básicas son ligeramente diferentes en el proceso de descifrado, sin embargo es poco el esfuerzo necesario para poder comprender dicho proceso.

Los autores del algoritmo describen en su libro con mayor detalle varios aspectos aquí expuestos (Joan 2002).

Recordemos que el proceso esta formado por dos partes, las cuales son cifrado, y generación de subclaves o extensión de la subclaves K_i. El bloque de cifrado tiene una longitud de 128 bits, la longitud de la clave K varia de 128, 192 y 256 bits, en el caso *AES-Rijndael* se tienen 10 rondas respectivamente(Jose de Jesus 2005).

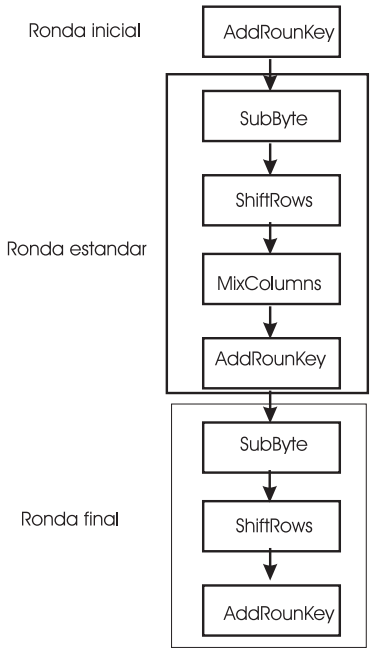

Figura 4.1: Cifrado *AES*

De acuerdo al apartado 3.1, el pseudocódigo para el cifrado queda:

```
int EncryptBlock (word8 a[4][MAXBC], word8
rk[MAXROUNDS+1][4][MAXBC]) {
     /* Cifrado de un bloque */
     int r;
     /* Empieza con la adicion de una clave */
     AddRoundKey(a,rk[0]);
     /* ROUNDS-1 rondas comunes */
     for(r = 1; r < ROUNDS; r++)
     {
           SubBytes(a,S);
           ShiftRows(a,0);
           MixColumns(a);
           AddRoundKey(a,rk[r]);
     }
     /* La ultima ronda es especial: no hay MixColumns */
     SubBytes(a,S);
     ShiftRows(a,0);
     AddRoundKey(a,rk[ROUNDS]);
     return 0;
}
```

4.1.1. Propiedades algebraicas

Al cambiar el orden de las aplicaciones hechas en el proceso *AES-Rijndael*, es decir que en lugar de aplicar la secuencia de sustitución (*ByteSub*), desplazamiento de filas (*ShiftRow*), multiplicación de columnas (*MixColumn*), (*AddRoundKey*), se puede iniciar cambia por la de sustitución (*ByteSub*), multiplicación de columnas (*MixColumn*), desplazamiento de filas (*ShiftRow*), adición de la clave (*AddRoundKey*). Esto no alteraría de ninguna manera la seguridad de *AES-Rijndael*, y sólo cambiaría la salida, (Muñoz 2004).

Ya que el orden de las operaciones desplazamiento de filas (*ShiftRow*) y sustitución (*ByteSub*) es indiferente. Esto es debido a que la operación desplazamiento de filas (*ShiftRow*) solo afecta a la posición de los bytes, pero no modifica su valor. En el caso de la sustitución (*ByteSub*), el campo de trabajo es el valor de los bytes y es independiente de cual sea su posición en la matriz de estado $S - Box$.

Tambien podemos apreciar que la secuencia cifrado y descifrado en *AES-Rijndael* es indiferente (Pais 2003), ya que haciendo uso de las propiedades algebraicas de la composición de transformaciones, el orden de las operaciones desplazamiento de filas (*ShiftRows*) y sustitución (*SubByte*) son permutables.

De acuerdo a esto se puede tener que para el descifrado:

```
int DencryptBlock (word8 a[4][MAXBC], word8
rk[MAXROUNDS+1][4][MAXBC]) {
     int r;
     /* Para descifrar:
      * Aplicar las operaciones inversas de la rutina de cifrado, en orden
      * opuesto
      *
      * - AddRoundKey es igual a su inversa
      * - La inversa de SubBytes con la tabla S es SubBytes con la inversa
      *   de la tabla S
      * - La inversa de Shiftrows es Shiftrows sobre una distancia variable
      */
     /* Primero la ronda especial:
      * Sin InvMixColumns
      * Sin extra AddRoundKey
      */
     AddRoundKey(a,rk[ROUNDS]);
     SubBytes(a,Si);
     ShiftRows(a,1);
```

```
    /* ROUNDS-1 rondas comunes */
    for(r = ROUNDS-1; r > 0; r--)
    {
          AddRoundKey(a,rk[r]);
          InvMixColumns(a);
          ShiftRows(a,1);
          SubBytes(a,Si);
    }/* Terminar con la adicion de la clave extra */
    AddRoundKey(a,rk[0]);
    return 0;
}
```

4.2. Diagrama secuencial del cifrado *AES-Rijndael*

Una arquitectura cliente-servidor favorece el carácter de divulgación y didáctica en este proyecto, ya que este algoritmo estará totalmente preparado para ser instalado en una red de computadoras o incluso en internet, donde cualquier persona interesada podrá acceder a la página web, y podra leer el contenido de este proyecto, además de probar el programa de descifrado para mejorar sus conocimientos sobre la criptografía aplicada.

Para el desarrollo de la página web del proyecto se ha utilizado el editor de páginas web Microsoft *FrontPage 2003*. El cual se ha seleccionado debido a su fácil uso y la absoluta compatibilidad que ofrece con el navegador Microsoft Internet Explorer, que es uno de los navegadores más usados.

El descifrado comienza cuando el usuario selecciona la opción *AES-Rijndael* en el cuadro de diálogo de descifrado, para después introducir la clave, y así realizar el proceso de descifrado, para posteriormente presentar al usuario una pantalla con el texto descifrado. Si por error el usuario no introduce la clave correcta no podrá descifrar el texto.

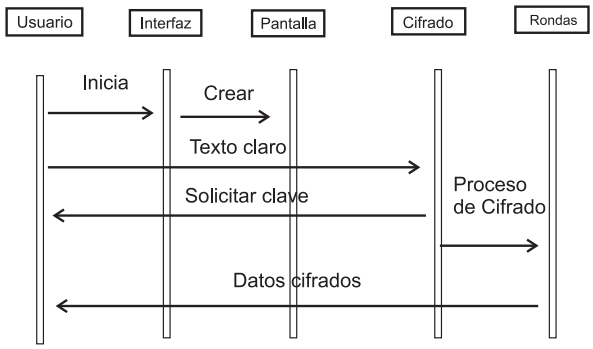

Figura 4.2: Secuencia de AES

4.3. Resistencia al criptoanálisis que posee el Algoritmo *AES-Rijndael*

En esta sección se analizan los parámetros del diseño del algoritmo *AES-Rijndael*, así como los motivos de su elección y la influencia que tiene sobre la seguridad del mismo, teniendo en cuenta esto, el algoritmo *AES-Rijndael* se estructura en tres axiomas, los cuales son:

- Resistencia contra todos los ataques conocidos (ver apéndice D).

- Rapidez y compatibilidad en todas las plataformas.

- Sencillez en el diseño.

Teniendo en cuenta estos principios se puede ver que el diseño de las funciones matemáticas que utiliza el algoritmo, está en función del número de rondas y la estructura algebraica del algoritmo (Capítulo 2).

AES tiene una gran resistencia a ataques del tipo: Criptoanálisis lineal y Criptoanálisis diferencial ya que AES cuenta con las siguientes características:

•Una capa de mezcla lineal, que garantiza una alta difusión de la información a través de la aplicación de las rondas intermedias.

•Una capa no lineal.

•Una capa de adición de claves, que consiste en una simple operación or-exclusiva, entre las subclaves de cada ronda en los estados intermedios.

Dadas estas características se le puede considerar un algoritmo resistente al criptoanálisis de cualquier tipo, sin perder de vista que sus debilidades pueden estar en la estructura algebraica de cada una de sus partes.

4.3.1. Resistencia de la función sustitución (*SubByte*) del algoritmo .

La transformacion sustitución (*SubByte*) consiste de una capa no lineal, diseñada específicamente para ofrecer resistencia a ataques como el criptoanálisis lineal, diferencial, ataques de interpolación y ataques de tipo *Square*. Los criterios adoptados para su diseño fueron:

•Que la función sustitución (*SubByte*) sea reversible.

•Minimizar la relación lineal entre los bits de entrada y bits de salida.

•Complejidad mediante expresiones algebraicas en $GF(2^8)$

•Sencillez del diseño.

Otro aspecto importante es el relacionado con la matriz (*S-Box*), ya que dicha matriz contiene las inversas multiplicativas en $GF(2^8)$, la cual tiene propiedades de no linealidad (Apéndice C.1). Sin embargo, la simplicidad algebraica de este diseño podría permitir ataques para descifrar los mensajes.

Para evitar un ataque se añadió una trasformación reversible adicional. Dicha trasformación está formada por una matriz que representa una sencilla expresión, la cual involucra una compleja expresión algebraica, mas un vector del tipo 011000111 (Joan 2002). El cual debe tener la característica de ser no lineal, esto es, que la tabla *S-Box* no debe cumplir con las siguientes características:

$$S - Box(a) = a$$
$$S - Box(a) = \bar{a}$$

Ya que de lo contrario se puede crear una puerta trasera en el cifrado, por lo que la tabla obtenida en Apéndice C.1 cumple con esta propiedad (Pais 2003).

4.3.2. Resistencia de la función desplazamiento de filas (*ShitfRow*).

El número a desplazar en esta transformación es elegido por ofrecer una resistencia contra ataques conocidos como los denominados *truncated dufferentials* y *Square*, ya que estos pueden descifrar algunas de las subclaves y así encontrar una debilidad. Esto se puede ver con claridad en la sección 2.4.

4.3.3. Resistencia de la función multiplicación de columnas (*MaxColumn*).

La función multiplicación de columnas (*MaxColumn*) ha sido seleccionada como una transformación lineal de 4 en 4 bytes tomando encuenta los siguientes criterios:

•Que sea reversible.

•Linealidad en $GF(2^8)$.

•Propiedades de alta difusión.

•Alta velocidad en cualquier tipo de procesadores.

•Simetría.

•Sencillez de operación.

La elección de coeficientes y polinomios permiten una gran difusión entre los bytes resultantes de la transformación.

4.3.4. Expansión de la clave.

Esta función permite derivar la subclave a partir de una clave dada, como se describe el la sección 2.6, dando paso a la generación de subclaves. Su utilidad reside en permitir la resistencia a ataques del tipo:

•Ataques donde parte de la clave es conocida.

•Ataques donde se puede tener parte de las subclaves, por lo que se encontrarían las subclaves y así obtener la clave general.

•Ataques por claves relacionadas, esto es, evitar que distintas claves del cifrado puedan producir un gran conjunto de subclaves comunes.

La función extensión de las subclaves (*Key Schedule*) es vital ya que también debe cumplir con las siguientes características.

•Simetría en una ronda: esta ronda trata a todos los bytes de un estado n en general de la misma forma, la simetría es alterada gracias a la generación de subclaves ya que en cada ronda cambia esta.

•Simetría entre rondas: la ronda de trasformación es idéntica para todas. Esta igualdad es alterada obteniendo subclaves que dependen de cada una de las ronda generadas anteriormente.

El algoritmo *AES-Rijndael* es el sistema de cifrado mas robusto que se conoce en la actualidad. Según los estudios publicados (Amparo 2001), no existe ningún mecanismo comprobado para intervenir el algoritmo *AES-Rijndael*, ya que el ataque más eficiente es el de búsqueda exhaustiva de claves, que se reduce al ataque de las claves.

Por lo que la fortaleza del sistema depende en parte da la clave y en cada una de sus transformaciones, ya que de acuerdo con sus características algebraicas se pueden proponer los siguientes cambios, respetando las características esenciales del algoritmo AES propuesto por los autores (Joan 2002), para poder tener mayor seguridad en el cifrado y descifrado de datos.

4.4. Implementación del algoritmo *AES-Rijndael* en diferentes tecnologías.

El propósito de *AES-Rijndael* es que sea tomado como un sistema de seguridad estándar, es decir que la mayoría de sus aplicaciones se puedan comunicar de manera segura.

En la actualidad hay una gran variedad de aplicaciones que soportan AES y esto permite una mejor comunicación de manera segura. Las aplicaciones que ya soportan AES están *SSL, IPsec, WinZip* (Pino 2003), entre otras. Y se planea usarlo en nuevas generaciones de *GSM*. Sin embargo, es posible dada su construcción implementar *AES* de una manera propia, esto con objeto de tener un algoritmo dedicado a cierta aplicación y así aprovechar su diseño.

Por ejemplo, en sistemas cerrados que no necesitan ser compatibles con el exterior se puede modificar AES o cambiar algunos parámetros y, obtener un nuevo algoritmo tan seguro como el propuesto por los autores.

Las primeras ideas son muy simples, como la propuesta al FIPS(*Federal Information Processing Standards*), donde se describe AES y se determinan que tipo de parámetros se deben de usar para la denominación AES. Por lo tanto, el cambio o modificación de alguno de ellos, en principio, nos dará una nueva versión de AES. Estos cambios deben hacerse de manera muy cuidadosa para no perder las principales características de seguridad que tiene AES (Pino 2003).

4.5. Ventajas y limitaciones del AES.

4.5.1. Ventajas

AES-Rijndael se puede ser ejecutado a velocidades elevadas para cifradores de bloque, y así como en *Smart Cards*, empleando un mínimo de memoria RAM y usando un número reducido de rondas. Las operaciones de cada ronda se pueden paralelizar, lo cual supone un importante avance en aplicaciones del tipo software y hardware específico.

Como el cifrador no hace uso de operaciones aritméticas, no existe ningún prejuicio con respecto a algun tipo de arquitectura de procesadores. Su simplicidad en el diseño permite que el cifrador sea, por decirlo de alguna manera, autónomo e independiente. Ya que no hace uso de ningún otro componente criptográfico.

El cifrador no basa su seguridad, o parte de ella, en operaciones aritméticas imposibles o secretas. Y el diseño hermético del cifrador no permite la existencia de puertas traseras ocultas.

Respecto a la longitud variable del bloque, tanto de 192 como de 256 bits permiten la construcción de una función tipo hash (Elad 2002). Usando *AES-Rijndael* como función de compresión y con la característica de que el número de colisiones provocadas sería mínimo. Hoy en día, la longitud de bloque de 128 bits es una buena opción y cuenta con una gran seguridad.

4.5.2. Limitaciones.

Las limitaciones del cifrador están relacionadas con su inversa: el descifrador *AES-Rijndael* es más difícil de implementar en *Smart Cards*, que el propio cifrador, debido a que requiere una mayor cantidad de código y un número superior de rondas. A pesar de ello, comparado con otros cifradores, la función inversa es realmente rápida. En implementaciones de software, el cifrador y su inversa emplean diferente código y diferentes tablas. En implementaciones hardware, el descifrador solamente puede aprovechar una parte de la circuitería implementada para el cifrador.

Capítulo 5

Conclusiones

Hoy en día muchas personas manejan el concepto de seguridad de una manera muy relativa. Sin embargo, en los últimos 20 años se han podido resumir varios puntos básicos para que un algoritmo sea seguro (Pino 2003).

Hay quienes dicen que aún el diseño de un algoritmo de cifrado es más ingeniería que ciencia. En términos generales un algoritmo es seguro si está diseñado para soportar todo tipo de ataques conocidos hasta el momento (apéndice D) y, da la suficiente evidencia de su fortaleza. Aunque es claro que siempre existe la posibilidad de que el algoritmo pueda ser vulnerado por recientes y novedosos ataques, es decir, es imposible diseñar un algoritmo inmune a ataques no conocidos .

La eficiencia del algoritmo depende del diseño y de su implementación en las plataforma donde se ejecute, entre otras cosas. Obviamente las mejores eficiencias alcanzadas para cualquier algoritmo se realizan en la implementación de hardware dedicados.

Se tiene entonces que el campo de la eficiencia esta dividida en dos, el software y el hardware.

La eficiencia en software del *AES-Rijndael*: es el código de partida, el cual ya se trató (capítulo 2) y es descrito por (Joan 2002) quienes muestran el funcionamiento de las tranformaciones básicas del AES. Este código no tiene por objetivo ser el más eficiente, sin embargo analizando cada una de sus partes podemos saber cuales de ellas pueden ser optimizadas.

La operación mas costosa en tiempo es la de calcular los inversos multiplicativos del campo $GF(8^2)$, en este caso la operación es pre calculada y substituida por un consulta a una matriz $S-Box$, con esta misma técnica varios cálculos del algoritmo

son pre calculados y entonces se evitan los cálculos reemplazandolas por consultas a tablas. C.1 y su inversas C.2.

Como se ha mencionado el AES es un algoritmo de cifrado confiable, ya que aún cuando su estructura es sencilla tiene un alto nivel de seguridad debido a su estructura algebraica. Si sumamos a esto la seguridad en los movimientos o sustitución de los parámetros resulta un algoritmo mas robusto a un criptoanálisis.

Los autores describen en su libro (Joan 2002) que puede haber variantes de este algoritmo, los cuales tienen una fuerte resistencia a los más conocidos ataque (Jakobsen 1997), ya que como se menciona en el capítulo 2 las transformaciones puede ser modificadas siempre y cuando se respeten los parámetro propuestos.

A lo largo de esta tesis se han planteado alternativas que dan variantes al algoritmo y no lo debilitan. También se han consultado diferentes publicaciones en las cuales se señala que dichas alteraciones no afectan al algoritmo (Pais 2003), (Jose de Jesus 2005) y (Muñoz 2004) , ya que el algoritmo *AES-Rijndael* es del tipo modular y permite la posibilidad de que si en un futuro se encontrase una S-Box más afín a nuestras necesidades, ya sea por motivos de seguridad u otro cualquiera, se podría cambiar esta matriz por una nueva versión, sin tener que modificar en absoluto el resto del algoritmo.

El número de rondas se ha determinado en función del máximo de rondas para el cual se pueden prevenir los ataques shortcut (Massey 1991) con un considerable margen de seguridad. Un ataque shortcut (apéndice D) es más eficiente que uno por fuerza bruta.

Para ejecuciones del algoritmo *AES-Rijndael* se emplea una longitud de bloque y de clave de 128 bits, ya que no ha sido encontrado ningún ataque shortcut para implementaciones con más de 6 rondas, que debilite a este. Para conseguir un margen de seguridad, se han añadido cuatro rondas más, por lo que para claves y bloques de 128 bits se emplean 10 rondas.

Dado que con dos rondas el cifrado de *AES-Rijndael* proporcionan una alta difusión en el sentido de que cada bit de un estado depende de los valores de todos los bits de las dos rondas anteriores, o lo que es lo mismo, un cambio en el valor de un bit de un estado afecta a todos los bits de los estados de las dos siguientes rondas.

La adición de cuatro rondas puede ser vista como una operación de alta difusión al principio y al final del cifrado. La alta difusión de cada una de las rondas de *AES-Rijndael* depende de la estructura uniforme que opera en todos los bits de un estado.

En los llamados cifradores *Feistel*, (Douglas 2000) las operaciones de una ronda afecta únicamente a la mitad de los bits del estado, por lo que la propiedad de alta difusión es alcanzada, en teoría, tras tres rondas (Deamen 1997).

Otro aspecto importante es que este algoritmo soporta distintos tamaño de bloque , para el proceso de cifrado y descifrado, *AES-Rijndael* puede trabajar sobre tres tamaños de bloque diferentes 128, 192 y 256 bits (Joan 2002), sólo que al tener un bloque más grande se tiene que procesar mayor información y se ve afectada la velocidad. Por lo que, cuanto mayor sea el tamaño del bloque, se tendra una alta difusión de datos.

Esto son solo unos aspectos por los cuales el algoritmo *AES-Rijndael* es confiable, ya que como se mencionó, en los capítulos 2 y 3, si modificamos la secuencia propuesta en 3.1 podemos tener una variante del algoritmo *AES-Rijndael* sin perder fortaleza. Este trabajo propone que al cambiar cualquiera de su parámetros en sus transformaciones el algoritmo no se ve afectado en su fortaleza.

En la actualidad existen diversos algoritmos de cifrado con las características del *AES-Rijndael*, como los algoritmos DES y TDES, mencionados en capitulo 1, ya que estos trabajan a nivel de bits y por bloques, sólo que el tamaño de la clave es de 56 bits y los bloques de 64 bits de texto.

Durante algún tiempo se garantizó el secreto de la información, pero ha sido relegado a un segundo plano por el avance imparable de las potentes computadoras, lo que ha motivado que hoy en día, se pueda romper el secreto de un cifrado DES en un tiempo razonable empleando simplemente una computadora (Amparo 2001).

Al día de hoy, el valor de 56 bits de clave (DES) se ha quedado obsoleto frente a los nuevos algoritmos de cifrado, que emplean claves de mayor tamaño, con valores de 128 hasta 256 bits (Muñoz 2004), si lo comparamos con el algoritmo *AES-Rijndael* el cual maneja valores de 128, 256 y 512 bits, frente a la clave fija de 56 bits empleada por el DES. Además, esta clave de cifrado que aparece en el algoritmo DES, no es usada en su totalidad. De los 56 bits de la clave, ocho de ellos son los llamados bits de paridad, que no intervienen en el proceso de cifrado.

En la ejecución de cualquier algoritmo de cifrado, se identifican dos parámetros, directamente apreciables, pero a su vez mutuamente excluyentes, velocidad y fortaleza.

Estas dos cualidades son obligatorias en un buen algoritmo de cifrado a pesar de que por desgracia, el aumento de una de ellas implica la disminución de la otra. Cuanto mayor sea el tamaño de la clave principal de cifrado, más difícil será de romper el secreto del cifrado, aunque esto implica que el proceso de cifrado de la información precisa de un mayor número de ciclos de procesamiento, ya que el número de bits y el número de las operaciones que se deben procesar es superior.

Ahora bien, el algoritmo debe ser elegido considerando estos dos parámetros. Se debe emplear el máximo tamaño posible de clave en el proceso en el cual el tiempo no es un factor importante pero sí la seguridad.

Un ejemplo de dicho procesos puede ser el cifrado de los archivos personales que un usuario realiza al finalizar la jornada laboral o el cifrado de un correo electrónico.

El reto al que se enfrenta el criptoanálisis a la hora de realizar un ataque contra este tipo de procesos, es que la longitud de la clave que impide ataques por fuerza bruta o, sea cual sea la estrategia de ataque empleada, necesita muchos ciclos de computación, o lo que es lo mismo, tiempo.

En este sentido, AES permite adaptar la velocidad de ejecución del algoritmo a las necesidades específicas, mejorando la velocidad mediante el uso de una clave de 128 bits, también se puede seleccionar una fortaleza extrema mediante una clave de 256 bits ó empleando un valor intermedio de 192 bits.

En cuanto a la generación de subclaves, se encuentran similitudes entre AES y DES, ya que ambos algoritmos se engloban dentro de la categoría de cifradores producto (Bruce 1996). Esto es, generan las subclaves a partir de una clave principal mediante una serie de funciones concretas. Los cifradores de esta categoría son más seguros que los cifradores en cascada, que emplean una pequeña clave para cada ronda.

El hecho de poseer tres tamaños distintos de bloque aporta una mayor seguridad al algoritmo debido a que un criptoanálisis tendrá que enfrentarse, al problema de que a mayor tamaño de bloque aumenta la difusión de la información, y al problema que le supone el hecho de hallar cuál de los tres tamaños de bloque posible ha sido empleado en el proceso de cifrado.

AES-Rijndael puede combinar cualquiera de sus tres posibles tamaños de clave con cualquiera de los tres tamaños de bloque, es decir que existen nueve posibilidades distintas de combinación clave-bloque (Jose 1998). Esto provoca que un ataque tenga que hallar la clave y trabajar con los tres tamaños posibles de bloque, por lo que el trabajo que esto conlleva provoca un incremento en el tiempo empleado para romper el secreto de la información, por lo que, aumenta el tiempo para descifrarlo.

DES al poseer únicamente un tamaño de bloque posible, elimina esta variable de la ecuación, por tanto el tiempo necesario para romper el secreto es menor, ya que solamente posee una combinación clave-bloque posible y para descubrir la clave secreta no se tendrá que tomar el factor bloque en cuenta, ya que es una constante de 56 bits.

En el caso del algoritmo DES, el número de rondas de la red *Feistel* es igual a 16, por lo que cada bit se trata, únicamente, 8 veces en la red *Feistel* (Bruce 1996).

Para *AES-Rijndael* un bit es tratado tantas veces como rondas tenga especificadas, para tener una seguridad (Amparo 2001). El número de rondas de la red *Feistel* del algoritmo DES es inalterable, no se puede variar el número de rondas de la red *Feistel* sin modificar la estructura interna del algoritmo, por lo que una modificación de este tipo afecta directamente al diseño del algoritmo y podría comprometer la fortaleza del mismo.

El impedimento a la modificación del número de rondas de la red *Feistel* viene dado por el método que emplea DES para la generación de subclaves (Alfred 2004). En este algoritmo, una vez eliminados los bits de paridad y reducida la clave principal a 48 bits, divide a ésta en dos mitades.

Para calcular una subclave, se toman las dos mitades de la subclave anterior y se rotan uno o dos bits a la izquierda según los valores dados (Jose de Jesus 2005). Por lo que se tienen únicamente variaciones para obtener 16 subclaves. Si se quiere ampliar el número de rondas, se tendrá que ampliar la tabla mediante la adición de nuevas columnas con valores nuevos, y se necesitaría un nuevo análisis para valorar el efecto de las variaciones en la seguridad del algoritmo que provocarían estos cambios.

En AES el número de rondas del algoritmo N_r depende de los valores de N_k y N_b, que a su vez dependen, respectivamente, del tamaño de la clave y del tamaño de bloque seleccionado por el usuario. Cuanto mayor sea el tamaño de bloque y el tamaño de clave, mayor ha de ser el número de rondas del algoritmo.

Los diseñadores del algoritmo han seleccionado valores comprendidos entre 10 y 14 para el número de rondas (sección 2.7). Una de las principales críticas al diseño de AES fue el relacionado con el reducido número de rondas de su red interna, aunque este parámetro se puede variar sin dificultad, únicamente modificando los parámetros de acuerdo a la apartado 2.6. Se podría emplear un número mayor de rondas, para casos de extrema necesidad de privacidad, simplemente variando el valor correspondiente en 2.6 se especifica esto más a detalle.

Como se puede apreciar, AES vuelve a ofrecer un entorno totalmente configurable y adaptable al ámbito de uso (Pais 2003).

Los diseñadores se limitan a sugerirnos unos valores adecuados, aunque el diseño del algoritmo permite modificaciones de estos, de manera inmediata, simplemente cambiando el valor de una constante.

Lo anterior supone que, una vez tomada la decisión de modificar el algoritmo, el sistema podrá volver a operar en cuestión de minutos, el tiempo que lleve volver a compilar el algoritmo, por lo que la interrupción del sistema será mínima, y en el caso de redes corporativas, este proceso será ejecutado por el administrador del sistema de forma totalmente transparente al usuario.

El equipo de diseño de AES ha previsto la posibilidad de su trabajo en paralelo, aprovechando así las posibilidades de cálculo de máquinas que posean varios procesadores, mejorando así el tiempo de cifrado y descifrado del algoritmo y, facilitando las implementaciones en software y hardware específicos.

Considerando que los algoritmos de cifrado mencionados en este proyecto son susceptibles de sufrir ataques existentes hoy en día, an el caso del algoritmo DES, se ha demostrado que con la potencia computacional actual, es viable un ataque por fuerza bruta , empleando una computadora, debido a que su tamaño de clave reduce el espacio de claves a 56 bits.

En el caso del algoritmo AES el espacio de claves es notablemente mayor, ya que el tamaño de la misma abarcar desde 2128 hasta 2256 combinaciones posibles (Jose de Jesus 2005), y aunado a las variaciones que se puedan realizar, se tendrá una mayor fortaleza, lo cual hace inviable un ataque por fuerza bruta, ya que se tardaría cientos de años en descubrir la clave de cifrado.

Ahora bien, este nuevo método multiplica infinitamente la potencia computacional disponible, por lo que en un futuro se podría disponer de un ataque viable sobre internet.

Hoy por hoy, el espacio de claves es demasiado grande, por lo que el número de usuarios trabajando al mismo tiempo es tan grande que convierte este tipo de ataque en una utopía.

Apéndice A

Representación de un Byte en el campo $GF(2^8)$

El campo $GF(2^8)$ es un campo finito en el que los elementos (en nuestro caso bytes) serán representados como polinomios de grado 7 y con coeficientes binarios, esto es, en $0, 1$. Un byte b se compone de 8 bits que representamos como $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$ donde b_7 representa el bit de mayor peso y b_0 al de menor. Así podemos representar el byte como un polinomio cuyos coeficientes son los b_j con $j = 0.,7$ y donde estos b_j pueden tomar los valores 0 ó 1, esto es.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 \tag{A.1}$$

Por ejemplo, un byte que represente el valor hexadecimal 57 que su representación en binario es 01010111 tendrá su correspondiente polinomio:

$$0x^7 + 1x^6 + 0x^5 + 1x^4 + 0x^3 + 1x^2 + 1x^1 + 1x^0 = x^6 + x^4 + x^2 + x + 1$$

Otro ejemplo será el que represente el valor hexadecimal 83 que en binario es 10000011 y su correspondiente polinomio es:

$$1x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0 = x^7 + x + 1$$

A.1. Construcción del campo finito $GF(2^8)$

Existe un polinomio $p(x) \in GF(2^8)$ irreducibles de grado 8, los cuales servirán para la construcción de $GF(2^8)$.

Binario	Hex	Polinomio	Orden
100011011	11b	$x^8+x^4+x^3+x+1$	51
100011101	11d	$x^8+x^4+x^3+x^2+1$	255
100101011	12b	$x^8+x^5+x^3+x+1$	255
100111001	12d	$x^8+x^5+x^3+x^2+1$	255
100111111	139	$x^8+x^5+x^4+x^3+1$	17
101001101	13f	$x^8+x^5+x^4+x^3+x^2+x+1$	85
101011111	14d	$x^8+x^6+x^3+x^2+1$	255
101100011	15f	$x^8+x^6+x^4+x^3+x^2+x+1$	255

Cuadro A.1: Lista de algunos polinomios irreducibles de grado 8 en $GF(2^8)$

Como sabemos el campo $GF(2^8)$ es el cociente $\mathbb{Z}_2[x]/f(x)$ donde $f(x)$ es cualquier polinomio de tabla A.1, en el caso de AES es indiferente que polinomio se elija, sin embargo para ser compatibles con *AES-Rijndael* se toma al primer polinomio irreducible, es decir $m(x) = x^8+x^4+x^3+x+1$.

Como recordamos $\mathbb{Z}_2[x]/f(x)$ es el conjunto de residuos modulo $m(x)$, es decir el conjunto de polinomios de grado menor a 8. Las operaciones de campo a partir de el polinomio $m(x)$ se obtienen modulo $m(x)$, es decir, es el residuo de la división entre $m(x)$.

Particularmente el campo $GF(2^8)$, se construye tomando el primer polinomio irreducible de la lista $m(x) = x^8+x^4+x^3+x+1$ (11b) que tiene orden 51, es decir x no genera a todo el campo, sin embargo en el proceso de las multiplicaciones se toma al polinomio $g(x) = x+1$ como generador. *AES-Rijndael* toma como base al campo $GF(2^8)$, ya que considera a un byte(conjunto de 8 bits) como su palabra básica, haciendo posible así, la implementación en muchas plataformas a partir de los 8 bits.

El campo $GF(2^8)$ entonces se muestra en la siguiente tabla con todos sus elementos, donde el entero es la potencia i de $(1+x)^i$ y el polinomio el resultado de $(1+x)^i \bmod m(x)$:

Por lo que la representación del campo $GF(2^8)$, damos las siguientes dos reglas que son usadas en el código para poder multiplicar dos elementos del campo $GF(2^8)$. La idea es simple, si queremos multiplicar dos elementos digamos $a, b \in GF(2^8)$, entonces existen i, j tales que $a = (1+x)^i$, $b = (1+x)^j$ por lo tanto $ab = (1+x)^{i+j \bmod 255}$, si basta encontrar el elemento que esta el la posición $(i+j) \bmod 255$ para saber el resultado. En términos de notación conocida decimos que $ab =$ AntiLog $((\text{Log}\,[a] + \text{Log}\,[b]) \bmod 255)$.

A.2. Suma de polinomios en el campo $GF(2^8)$.

La suma de dos elementos del campo $GF(2^8)$ es la suma de dos polinomios, por lo que el resultado sera otro polinomio. La suma de los coeficientes se corresponde con una suma modulo 2 termino a termino. Se puede comprobar que esta suma se corresponde con una operación $EXOR$ (denotada por $\oplus$.) entre los coeficientes de los polinomios, la cual se representa en la Tabla A.2.

A	B	XOR ($\oplus$)
0	0	0
0	1	1
1	0	1
1	1	0

Cuadro A.2: Función XOR

Por ejemplo, se puede efectuar la suma de los elementos del apartado anterior:

$$(57 + 83) = (57 \oplus 83) = 01010111 \oplus 10000011 = 11010100$$

Y en terminos de polinomios.

$$(57 + 83) = (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

Podemos comprobar que el conjunto de los polinomios de grado menor o igual que 7 y con coeficientes pertenecientes a Z_2 forman un grupo conmutativo con la suma, es decir, es una operación interna, que cumple la propiedad asociativa, conmutativa, tiene elemento neutro y tiene simétrico. Debido a la existencia de simétrico podemos referirnos a la operación resta, ya que se puede definir la resta de a y b, donde a y b son polinomios, como la suma de a con el simétrico de b.

A.3. Multiplicación de polinomios en el campo $GF(2^8)$.

Al referirnos a la multiplicación empleada en el algoritmo Rijndael nos estaremos refiriendo realmente a la multiplicación de dos elementos del conjunto $GF(2^8)$, es decir polinomios de grado menor o igual que 7 y con coeficientes en Z_2 pero cuyo resultado se expresa modulo m(x) donde $m(x) = x^8+x^4+x^3+x+1$. Nótese que m(x) se puede representar en binario con el valor 100011011 y se puede comprobar que es un polinomio irreducible. El propósito de realizar la multiplicación módulo m(x) es

con el fin de que el resultado obtenido en la operación siga siendo un polinomio de grado menor que 8, por lo que la operación seguiría siendo a nivel de byte.

Por ejemplo, la operación multiplicación de los valores hexadecimales 57 y 83 sería:

$$(57 \cdot 83) = (x^6 + x^4 + x^2 + x + 1) \cdot (x^7 + x + 1) \tag{A.2}$$

Tomando el polinomio irreducible $m(x)$

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{A.3}$$

Realizando la multiplicación de la ecuación A.2.

$$(57 \cdot 83) = x^{13} + x^{11} + x^9 + x^8 + 2x^7 + x^6 + x^5 + x^4 + x^3 + 2x^2 + 2x + 1 \tag{A.4}$$

Como estamos trabajando sobre los polinomios en el campo $GF(2^8)$, los valores diferentes de $[0, 1]$, serán anulados. Por lo que el polinomio A.4 queda de la siguiente forma.

$$(x^6 + x^4 + x^2 + x + 1) \cdot (x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \tag{A.5}$$

Como se puede apreciar este polinomio es de grado mayor que 8 por lo que no pertenece a $GF(2^8)$ y así la operación no se realiza a nivel de byte. Para remediar esto y conseguir que la multiplicación siga siendo una operación interna en $GF(2^8)$ expresamos el resultado obtenido modulo $m(x)$, esto es

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod m(x) \tag{A.6}$$

Por lo que la ecuación A.6 queda al sustituir $m(x)$:

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \; mod \; x^8 + x^4 + x^3 + x + 1 \tag{A.7}$$

Dando como resultado

$$\begin{aligned} (57 \cdot 83) \; mod \; m(x) &= x^7 + x^6 + 1 \\ &= 11000001 \\ &= c1 \end{aligned} \tag{A.8}$$

Otra forma de calcularlo será razonando que resultado de la multiplicación es reducido para cada valor de x que este fuera del cuerpo de 8 bits.

$$\begin{aligned} m(x) &= x^8 + x^4 + x^3 + x + 1 \\ x^8 &= x^4 + x^3 + x + 1 \end{aligned} \tag{A.9}$$

$$(57 \cdot 83)\ mod\ 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \tag{A.10}$$

Dado que los elementos x^{13}, x^{11}, x^9, no esta dentro del campo se tiene que reducir el polinomio y esto es obteniendo el residuo de la siguiente manera y recordando la ecuación A.9 : Para x^{13} se tiene.

$$\begin{aligned} x^{13} &= x^5 * x^8 \\ &= x^5 * (x^4 + x^3 + x + 1) \\ &= x^9 + x^8 + x^6 + x^5 \\ &= x * x^8 + x^8 + x^6 + x^5 \\ &= x * (x^4 + x^3 + x + 1) + (x^4 + x^3 + x + 1) + x^6 + x^5 \\ &= x^5 + x^4 + x^2 + x + (x^4 + x^3 + x + 1) + x^6 + x^5 \\ x^{13} &= x^6 + x^3 + x^2 + 1 \end{aligned} \tag{A.11}$$

Para x^{11} se tiene.

$$\begin{aligned} x^{11} &= x^3 * x^8 \\ &= x^3 * (x^4 + x^3 + x + 1) \\ &= x^7 + x^6 + x^4 + x^3 \end{aligned} \tag{A.12}$$

Para x^9 se tiene.

$$\begin{aligned} x^9 &= x * x^8 \\ &= x * (x^4 + x^3 + x + 1) \\ &= x^5 + x^4 + x^2 + x \end{aligned} \tag{A.13}$$

Sustituyendo A.9, A.11, A.12, A.13 en A.10 queda:

$$\begin{aligned}(57 \cdot 83)\ mod\ 2 = (x^6 + x^3 + x^2 + 1) \\ +(x^7 + x^6 + x^4 + x^3) \\ +(x^5 + x^4 + x^2 + x) \\ +(x^4 + x^3 + x + 1) \\ +x^6 + x^5 + x^4 \\ +x^3 + 1\ mod\ 2\end{aligned} \tag{A.14}$$

Lo que da por resultado.

$$\begin{aligned}(57 \cdot 83)\ mod\ 2 &= x^7 + x^6 + 1 \\ &= 11000001 \\ &= c1\end{aligned} \tag{A.15}$$

Como se puede observar el polinomio resultante es de grado menor a 8. La multiplicación de polinomios es asociativa y su elemento neutro es 01, para cualquier polinomio binario $b(x)$ de grado menor a 8 y se puede aplicar el algoritmo extendido de Euclides para calcular el polinomio inverso de $b(x)$, a lo que se denominar polinomio inverso $a(x)$esto es:

$$a(x) \cdot b(x)\ mod\ = 1$$

ó

$$b(x)^{-1} = a(x)\ mod\ m(x)$$

es decir que $a(x)$ es la inversa multiplicativa de de $b(x)$. Por lo que el polinomio cumple con el polinomio extendido de Euclides se puede escribir como:

$$b(x)a(x) + m(x)c(x) = 1$$

Apéndice B

Representación de palabras en el campo $GF(2^8)$

Recordemos que denotamos como palabra a un conjunto de bytes. En este sentido una palabra formada por cuatro bytes se puede representar como un polinomio de grado menor o igual que tres.

$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$
$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$

La operación suma de polinomios se realiza mediante unas operaciones XOR byte a byte, al igual que se hacía anteriormente al representar un byte mediante un polinomio de grado menor que 7. Esta operación es interna en $GF(2^8)$ ya que la suma de dos polinomios de grado menor que 4 nos dará como resultado otro polinomio de grado menor que 4.
En cuanto a la operación multiplicación, nos encontramos de nuevo con el problema del apartado anterior. La multiplicación puede no ser una operación interna en $GF(2^8)$, por lo que el producto de dos palabras de 4 bytes puede no ser representable por una palabra de 4 bytes, es decir, mediante un polinomio de grado menor que 4.

$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$
$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$
$a(x) \cdot b(x) = c(x) = c_6x^6 + c_5x^5 + c_4 + c_3x^3c_2x^2 + c_1x + c_0$
donde
$c_0 = a_0 \cdot b_0$
$c_1 = a_1 \cdot b_0 \otimes a_0 \cdot b_1$
$c_2 = a_2 \cdot b_0 \otimes a_1 \cdot b_1 \otimes a_0 \cdot b_2$
$c_3 = a_3 \cdot b_0 \otimes a_2 \cdot b_1 \otimes a_1 \cdot b_2 \otimes a_0 \cdot b_3$
$c_4 = a_3 \cdot b_1 \otimes a_2 \cdot b_2 \otimes a_1 \cdot b_3$

$c_5 = a_3 \cdot b_2 \otimes a_2 \cdot b_3$
$c_6 = a_3 \cdot b_3$

Para solucionar este contratiempo se ha propuesto una solución en la cual el resultado de la operación multiplicación se expresa módulo un polinomio de grado 4. Los autores del algoritmo *AES-Rijndael* han elegido el polinomio $M(x) = x^4 + 1$ para tal fin. En este caso $M(x)$ no es un polinomio irreducible como se le había exigido al anterior $m(x)$. Esto va a provocar que algunas de las multiplicaciones que realicemos puedan dar un resultado que no tenga inverso. En el caso del algoritmo Rijndael este caso no se va a presentar nunca ya que siempre se multiplica por polinomios que poseen inverso. Por lo que queda:

$(a(x) \cdot b(x)) \bmod (x^4 + 1)$
donde

$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

Y para hallar d_3, d_2, d_1, d_0 aplicamos una sencilla regla:
$x^j = \bmod (x^4 + 1) = x^{j \bmod 4}$

Así obtenemos los siguientes valores:

$d_0 = a_0 \cdot b_0 \otimes a_3 \cdot b_1 \otimes a_2 \cdot b_2 \otimes a_1 \cdot b_3$
$d_1 = a_1 \cdot b_0 \otimes a_0 \cdot b_1 \otimes a_3 \cdot b_2 \otimes a_2 \cdot b_3$
$d_2 = a_2 \cdot b_0 \otimes a_1 \cdot b_1 \otimes a_0 \cdot b_2 \otimes a_3 \cdot b_3$
$d_0 = a_0 \cdot b_0 \otimes a_3 \cdot b_1 \otimes a_2 \cdot b_2 \otimes a_1 \cdot b_3$

Otra forma de expresar esta operación es mediante el uso de matrices.

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Un caso especial de la multiplicación de polinomios es la multiplicación de un polinomio $b(x)$ por el polinomio x.

$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$
$d(x) = x \cdot b(x) = b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

Dividimos $d(x)$ entre $M(x) = x^4 + 1$ para obtener un polinomio de grado menor que cuatro.

$$c(x) = d(x) \bmod (x^4 + 1) = b_2x^3 + b_1x^2 + b_0x + b_3$$

Esta multiplicación también se puede expresar como el producto de dos matrices, de la misma forma que la matriz anterior pero cuyos elementos son sustituidos todos 00, a excepto los a_1,que se sustituyen por el valor 01

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Al igual que en el caso de los polinomios de grado menor que siete, que representaban a los bits de un byte, si aplicamos restrictivamente esta multiplicación obtenemos una manera rápida y sencilla de multiplicar un polinomio por cualquier potencia de x.

B.1. El anillo $GF(2^8)$.

La estructura que usa AES es la del anillo $GF(2^8)[x]/(x^4+1)$, otra de las operaciones básicas de AES es multiplicar un polinomio de tercer grado con coeficientes en $GF(2^8)$, por una constante. Es decir, un elemento de $GF(2^8)[x]/(x^4+1)$, por un polinomio constante, el resultado se reduce modulo (4 x +1) para obtener un polinomio de grado 3. El objetivo de lo anterior es poder definir multiplicación columnas de 4 bytes por un elemento constante también de 4 bytes, en el proceso de descifrado se aplica la operación inversa y aunque el polinomio $(x^4 + 1)$ no es irreducible, en este caso particular la constante si tiene inverso en el anillo $GF(2^8)[x]/(x^4+1)$, por lo tanto la constante tiene inverso.

Sea $a(x)$ un polinomio tal que $a_0+a_1x+a_2+x^2+a_3x^3 \in GF(2^8)[x]/(x^4+1)$, donde $a_i \in GF(2^8)[x]/(x^4+1)$, y $b(x)$ es otro polinomio igual, entonces $a(x)b(x) = c(x)$ donde $c(x)$ tiene la misma forma, particularmente:

$$\begin{aligned} a(x)b(x) &= (a_0 + a_1x + a_2 + x^2 + a_3x^3)(b_0 + b_1x + b_2 + x^2 + b_3x^3) \\ &= a_0b_0 + (a_1b_0 + a_0b_1)x + (a_2b_0 + a_1b_1 + a_0b_2)x^2 \\ &+ (a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)x^3 \\ &+ (a_3b_1 + a_2b_2 + a_1b_3)x^4 \\ &+ (a_3b_2 + a_2b_3)x^5 + a_3b_3x^6 \end{aligned} \tag{B.1}$$

El siguiente paso es aplicar modulo $x^4 + 1$ a todo el polinomio, que es aplicar modulo a cada uno de sus términos, entonces $x^i \bmod x^4 + 1 = x^{\mathrm{i\ mod4}}$, entonces el resultado de $a(x)b(x)$ queda como:

$$\begin{aligned} a(x)b(x) &= (a_0b_0 + +a_3b_1 + a_2b_2 + a_1b_3) \\ &+ (a_1b_0 + a_0b_1 + a_3b_2 + a_3b_3)x \\ &+ (a_2b_0 + a_1b_1 + a_0b_2 + a_3b_3)x^2 \\ &+ (a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)x^3 \\ &= c_0 + c_1x + c_2x^2 + c_3x^3 \end{aligned} \tag{B.2}$$

Finalmente de manera matricial el anterior producto puede ser visto como:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \bullet \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Apéndice C

Transformaciones no lineal S-Box

C.1. Función sustitución (*SubByte*)

La transformación sustitución (*SubByte*) consiste en una sustitución no lineal que se aplique a cada byte de la matriz de Estado (estado intermedio 1) de forma independiente, generando un nuevo byte. Esta transformación consiste en la sustitución de cada byte por el resultado de aplicarle la tabla de sustitución S-box. Esta tabla lógicamente es invertible y se construye mediante dos transformaciones:

1^aTransformación. Cada byte es considerado como elemento en $GF(2^8)$ que genera el polinomio irreducible $m(x) = x^8 + x^4 + x^3 + x + 1$, siendo sustituido por su inversa multiplicativa. El valor cero queda inalterado, ya que tiene inversa.

2^a Transformación .Al resultado de la 1a transformación se le aplica la siguiente transformación afín en $GF(2)$, siendo $x_0, x_1, x_2, x_3, x_4, x_5, x_6$, y x_7 los bits del byte resultante de la 1a transformación , e $y_0, y_1, y_2, y_3, y_4, y_5, y_6$ e y_7 los bits del resultado final de la transformación sustitución (*SubByte*).

Por ejemplo, si el byte al cual se le aplica la función sustitución (*SubByte*) es $A = 11001011$, deberíamos calcular su inversa multiplicativa. Cada byte en *AES-Rijndael* se representa como un polinomio $a(x)$, calcular la inversa multiplicativa consiste en buscar un polinomio $b(x)$ (que es único que multiplicado por $a(x)$ modulo $m(x)$ es igual a 1, es decir:

$$a(x) \bullet b(x) mod x^8 + x^4 + x^3 + x + 1 = 1$$

El polinomio buscado en este caso es $b(x) = x^2$ que tiene una representación binaria de $B = 00000100$.Se dice entonces que B es la inversa en $GF(2^8)$ de A. Una vez que tenemos el resultado de la primera transformación, debemos aplicar la transformación afín definida:

$$y_0 = x_0 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus c_0$$

$$y_1 = x_1 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_0 \oplus c_1$$
$$y_2 = x_2 \oplus x_6 \oplus x_7 \oplus x_0 \oplus x_1 \oplus c_2$$
$$y_3 = x_3 \oplus x_7 \oplus x_0 \oplus x_1 \oplus x_2 \oplus c_3$$
$$y_4 = x_4 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus c_4$$
$$y_5 = x_5 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus c_5$$
$$y_6 = x_6 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus c_6$$
$$y_7 = x_7 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus c_7$$

Siendo la representación matricial. También llamada:Transformación afín en $GF(2)$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{C.1}$$

Ejemplo de un valor concreto. Obtenemos el valor final de $Y = 00011111$. Luego el byte $A = 11001011$ se convierte en Y al aplicar la función sustitución (*SubByte*). Utilizando estas dos transformaciones para todos los valores posibles de entrada (256 valores ya que se trabaja como un byte) se calcula una tabla de sustitución denominada S-Box útil para el proceso de cifrado. Gracias a esta tabla aplicar la función sustitución (*SubByte*) resulta trivial, consiste en dividir el byte de la matriz de Estado en dos partes de 4 bits. Los 4 bits más significativos, denominados por x (toma valores de 0 a 15) actúan de fila en la tabla y los 4 bits menos significativos de columna, denominados por y (toma valores de 0a 15). El valor para esa fila y columna en la tabla es resultado de aplicar $S - BOX$ a un byte. Siguiendo con el ejemplo anterior, si tenemos el byte $A = 11001011$ y le aplicamos la función sustitución (*SubByte*) el resultado será ($x = 1100$[fila c] $y = 1011$[columna b]) $Y = 0x1F$ que en binario equivale a Y = 00011111, valor que es idéntico al calculado previamente.

Tabla de Sustitución $S - BOX$ para un byte genérico xy (hexadecimal).

Para el proceso de descifrado es necesario calcular la función inversa de sustitución (*invSubByte*). Esta función inversa consiste en calcular una tabla inversa a la utilizada en el proceso de cifrador, inversa $S - BOX$.

Finalmente el resultado de *ByteSub* puede resumirse en la siguiente tabla conocida como S-Box AES.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	8d	f6	cb	52	7b	d1	e8	af	29	c0	b0	e1	e5	c7
1	74	b4	aa	4d	99	2b	60	5f	58	3f	fd	cc	ff	40	33	b2
2	3a	6e	5a	f1	55	4d	a8	c9	c1	0a	98	15	30	44	a2	c2
3	2c	45	92	6c	f3	39	66	42	f2	35	20	6f	77	bb	59	19
4	1d	fe	37	67	2d	31	f5	69	a7	64	ab	13	54	25	e9	09
5	ed	5c	05	ca	4c	24	87	bf	18	3e	22	f0	51	ec	61	17
6	16	5e	af	d3	49	a6	36	43	f4	47	91	df	33	93	21	3b
7	79	b7	97	85	10	b5	ba	3c	b6	70	d0	06	a1	fa	81	82
8	83	7e	7f	80	96	73	be	56	9b	9e	95	d9	f7	02	b9	a4
9	de	6a	32	6d	d8	8a	84	72	2a	14	9f	88	f9	dc	89	9a
A	fb	7c	2e	c3	8f	b8	65	48	26	c8	12	4a	ce	e7	d2	62
B	0c	e0	1f	ef	11	75	78	71	a5	8e	76	3d	bd	bc	86	57
C	0b	28	2f	a3	da	d4	e4	0f	a9	27	53	04	1b	fc	ac	e6
D	7a	07	ae	63	c5	db	e2	ea	94	8b	c4	d5	9d	f8	90	6b
E	b1	0d	d6	eb	c6	0e	cf	ad	08	4e	d7	e3	5d	50	1e	b3
F	5b	23	38	34	68	46	03	8c	dd	9c	7d	a0	cd	1a	41	1c

Cuadro C.1: Matriz Inversa $S - BOX$

	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	3d	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

Cuadro C.2: Matriz S Box AES.

Apéndice D

Criptoanálisis

El aspecto más importante de un algoritmo es su seguridad, sin embargo, la seguridad no es algo fácil de manejar, en esta sección solo se describen aspectos muy generales de la seguridad de AES, un estudio más serio queda fuera del alcance de este documento. El propósito de esta sección es simplemente como complemento a la descripción general de AES. Hoy en día muchas personas manejan el concepto de seguridad de una manera muy relativa, el lenguaje usado es particular y no se tiene en general conceptos bien definidos. Sin embargo en los últimos 30 años se han podido resumir varios puntos básicos para que un algoritmo simétrico sea seguro. Hay quienes dicen que aún el diseño de un algoritmo criptográfico es más ingeniería que ciencia. En términos generales un algoritmo es seguro si éste está diseñado para soportar todos los ataques conocidos hasta el momento y da la suficiente evidencia de su fortaleza. Aunque es claro que siempre existe la posibilidad de que el algoritmo pueda ser roto por recientes y novedosos ataques, es decir, es imposible diseñar un algoritmo inmune a ataques no conocidos .

En el caso concreto de un algoritmo criptográficos simétrico existen varios comportamientos que nos dirán si un algoritmo puede considerarse seguro. Desde el punto de vista metodológico el algoritmo debió de haberse sometido en un tiempo considerable al análisis y estudio de la comunidad científica, el algoritmo debe de dar evidencia de haber pasado la mayoría de los análisis que pudieran existir, el algoritmo debe ser reconocido por un organismo dedicado(por ejemplo por el *NIST, NIESSIE, IEEE etc.*)(Jakobsen 1997). En términos más técnicos un algoritmo simétrico debe de dar evidencia de ser inmune a los ataques más potentes y conocidos, en este caso al menos a el análisis lineal y el análisis diferencial. Enseguida hacemos notar algunos puntos muy generales y características que AES tiene para poder evitar ataques como el lineal y diferencia. Las características más mencionadas que debe de contar un algoritmo simétrico son:

1.- La no linealidad entre las entradas y las salidas, o la correlación de las entradas con las salidas.

2.- La propagación de las diferencias de los bits, o el medir la probabilidad de que tanto se confunden los bits.

Particularmente las anteriores características son las más requeridas en un algoritmo simétrico, en el diseño las dos se mezclan y se miden en cada una de las rondas que consiste el algoritmo, es decir, en términos muy generales y básicos si un algoritmo tiene esas dos propiedades y se realizan las suficientes rondas, entonces el algoritmo será inmune a los análisis lineal y diferencial.

Cada uno de los elementos de *AES-Rijndael* fue cuidadosamente elegido para poder cumplir al menos con los dos requerimientos básicos anteriores.

D.1. Texto claro desconocido y clave desconocida.

Si el atacante solo dispone de un bloque cifrado, deberá cifrar con todas las claves posibles todos los bloques en claro posibles para ir comparando el resultado con el bloque cifrado. Para los valores estándar del algoritmo, con claves de 128 bits y tamaño de bloque de 128, se puede demostrar que el algoritmo no se puede invertir. Si se considera que el tamaño es de v bits con una clave de tamaño de n bits, el atacante deberá probar para cada clave 2^v bloques posibles y repetir el proceso para todas las 2^n claves . Actualmente resulta muy complicado estar probando todas estas combinaciones ya que el tiempo seria muy largo para probar todas las combinaciones posibles, esto nos llevaría a tener que combinar todas las posibles parejas clave-texto (Amparo 2001).

D.2. Texto claro desconocido y clave desconocida.

El atacante posee el texto en claro y el texto cifrado. Ante esta situación se puede realizar lo que se conoce como un ataque de fuerza bruta, es decir cifrar el texto claro con todas las posibles combinaciones de 2^v claves hasta producir un resultado que coincida con el texto cifrado. En esta situación el atacante debe aplicar el algoritmo de *AES-Rijndael* al texto en claro para estar seguro de que se ha obtenido la clave correcta.

Para una clave de 128 bits de longitud se necesitaría ejecutar 2^{127} veces el algoritmo de *Rijndael* sobre el texto en claro y comprobarlo con el texto cifrado (Amparo 2001).

D.3. Criptoanálisis Diferencial

El criptoanálisis diferencial es un método de ataque basado en un conjunto de pares de texto en claro-criptograma que se emplean para determinar el valor de los bits de la clave principal. La información sobre la clave se obtiene en base a deducciones procedentes de bloques de texto cifrado sobre bloques de texto en claro con una diferencia de bits específica entre ellos. La información sobre la clave se obtiene basándose en el hecho de que para que haya exactamente una cantidad de bits determinada de diferencia entre dichos bloques a la salida de una vuelta dada. Se imponen una serie de restricciones en las vueltas del algoritmo que median entre el punto en el que nos encontremos y el principio del algoritmo y se hace un supuesto sobre la clave. Acto seguido se comprueba la veracidad del supuesto y se continúa el proceso. El número de claves sugeridas por un par de bytes de texto en claro criptograma no es una constante, es decir, el número de claves sugeridas puede variar considerablemente según el par seleccionado, ya que las restricciones no tienen por qué ser las mismas. Probando con varios pares distintos obtendremos distintas posibles claves.

El ataque tendrá éxito si la clave que aparezca con más frecuencia coincide con el valor de la clave principal. Este tipo de ataque se ve favorecido por la estructura de la red Feistel del algoritmo DES, ya que la mitad izquierda del bloque de texto a la entrada de una vuelta determinada coincide con el bloque derecho a la entrada de la vuelta anterior (Douglas 2000) por lo que los patrones de diferencias sólo afectan a parte de las ocho S-Boxes del algoritmo DES, por lo que es susceptible de que un ataque mediante este método tenga éxito en un tiempo razonable. El diseño del algoritmo DES no estaba preparado para soportar este tipo de ataques, ya que fue desarrollado por *Eli Biham* y *Adi Shamir* en 1991 (Massey 1991) , casi quince años después del desarrollo de DES. Eli *Biham* y *Adi Shamir* demostraron que dado un sistema de cifrado DES con la clave pre cargada(Alfred 2004), se podía llegar a deducir la clave principal de cifrado mediante unas deducciones estadísticas basadas en los resultados de hacer pasar un total de 247 mensajes concretos a través de las S-Boxes de DES. A pesar de que este método de ataque es capaz de comprometer la seguridad del algoritmo DES, no logra romperlo con la facilidad que cabría suponer de un algoritmo diseñado con anterioridad a la creación del ataque, es más, muchos de los algoritmos modernos de cifrado son menos resistentes que el DES ante dicho ataque. Este hecho, junto a que los motivos que llevaron a la elección de las S-Boxes del DES han sido declarados como secreto nacional por las autoridades norteamericanas, han llevado a creer que durante el desarrollo de DES se había previsto un ataque de este tipo, aunque no se había llegado a concretar. Obviamente esta afirmación jamás ha sido demostrada y no pasan de ser meros rumores dentro de la comunidad científica. En este punto, reflejar la ventaja de la juventud del algoritmo *AES-Rijndael*, por lo que sus diseñadores ya han previsto la fortaleza del algoritmo

frente a este ataque.

D.4. Criptoanálisis Lineal

El criptoanálisis lineal es un método de ataque basado en pares de bytes de texto en claro criptograma empleados para obtener información sobre la clave principal de cifrado. Para una variante del algoritmo DES de 8 vueltas, el criptoanálisis se puede aplicar únicamente al criptograma. El criptoanálisis lineal basa su efectividad sobre los cifradores de bloque en tratar de encontrar expresiones de carácter lineal del siguiente tipo:

$$P[i_1, i_2, ..., i_n] = C[j_1, j_2, j_3....., j_n] + K[k_1, k_2, k_3, ..., k_n]$$

donde los vectores P, C y K representan el texto en claro, criptograma y clave respectivamente. Otra posible expresión lineal es ser la siguiente:

$$I[i_1, i_2, ..., i_n] = I_{15}[j_1, j_2, j_3....., j_n] + K[k_1, k_2, k_3, ..., k_n]$$

Esta segunda expresión no contiene referencias a bits de textos en claro o a textos cifrados, sino que trabaja con los estados intermedios del algoritmo (I). A partir de esta ecuación y haciendo una suposición sobre un subconjunto de bits de la subclave de la primera y de la última vuelta, se puede hallar el valor de los bits de I_1 e I_{15}, y reiterando este proceso, junto con la combinación de las expresiones anteriores, calcular los estados intermedios restantes, y por consiguiente, la clave de cifrado principal.

$$\begin{aligned} P[i_1, i_2, ..., i_a] + I_{m-1}[j_1, j_2, j_3....., j_b] &= K[k_1, k_2, ..., k_c] \\ I_{m-1}[j_1, j_2, ..., j_b] + I_m[m_1, m_2, ..., m_u] &= K[k_1, k_2, ..., k_d] \\ P[i_1, i_2, ...i_a] + I_m[m_1, m_2, ..., m_u] &= K[k_1, k_2, ..., k_d] \end{aligned} \tag{D.1}$$

En general, un ataque por criptoanálisis lineal contra el algoritmo DES o algún otro método de cifrado basado en él, como puede ser el *TDES*, se aprovecharía de que la mitad izquierda de los bits de salida de una vuelta son los mismos que la mitad derecha de los bits de entrada en esa vuelta, por lo que las ecuaciones anteriores se simplifican notablemente.

$$I_{i-1}[j_1 + 32, j_2 + 32, ..., j_a + 32] = I_i[j_1, j_2, ..., j_a]$$

El ataque lineal más efectivo contra DES se logra mediante la combinación de expresiones lineales sobre una vuelta concreta de la red *Feistel* con expresiones referentes a los bits de salida de alguna S-Box en concreto. En el caso del algoritmo

AES-Rijndael, se ha probado que este tipo de ataque no es efectivo siempre y cuando no se implemente una versión del algoritmo con menos de cuatro rondas en la red interna.

Por otra parte otro tipo de ataques o conceptos de debilidad que han sido propuestos, *Rijndael* los evita, como *Square Attack, Six Round Attack, Herds Attack, Gilbert-Minier Attack, Interpolation attack, Related- Key Attack, Timing Attack, Imposible Differential attack, Collision attack*, últimamente se han propuesto los llamados ataques algebraicos que en general explotan las propiedades algebraicas del algoritmo. Sin embargo por el momento no se ha podido montar un ataque a la versión completa de *AES-Rijndael*, lo que lo hace tan seguro como una búsqueda exhaustiva.

D.5. Ataques por análisis temporal (*Timing Attacks*).

Este ataque fue diseñado y desarrollado por *Paul Kocher*. En *Rijndael*, el uso de polinomios con coeficientes pertenecientes a $GF(2^8)$ favorece la fortaleza del algoritmo frente a ataques por análisis temporal. Este tipo de ataques se basan en el hecho de que una computadora tarda más tiempo en procesar una multiplicación matemática si los coeficientes de dicha operación son distintos de 0 que si alguno de ellos es 0, por lo que un criptoanálisis puede obtener información sobre la clave al realizar un estudio sobre el tiempo que emplea el algoritmo en procesar cada unos de los bits de dicha clave. Como hemos visto, el éxito de un ataque por análisis temporal reside en las multiplicaciones matemáticas. En *AES-Rijndael* sólo se lleva a cabo una multiplicación matemática en cada una de las rondas, ya que el resto son operaciones lógicas. Esta operación se lleva a cabo en la función *MixColumn* (sección 2.5). Este podría ser el único punto débil frente a ataques por análisis temporal, evitándolo por medio de coeficientes pertenecientes a $GF(2^8)$. El mayor coeficiente de multiplicación es 03 (notación hexadecimal). Se puede reducir este 03 a 02 + 01. La multiplicación por 01 es inmediata y aplicando las propiedades del campo $GF(2^8)$ se puede implementar una multiplicación por 02 de la siguiente forma:

•Desplazar los bits del multiplicando 1 posición a la izquierda.

•Si se produce un acarreo, realizar una operación XOR del resultado con 1B. Por lo tanto, se simplifican las multiplicaciones a operaciones lógicas, desplazamientos y sumas. La ventaja de haber seleccionado el campo $GF(2^8)$ como modo de representación es que facilita la modificación del algoritmo, ya que las propiedades de estos coeficientes permiten incluir funciones que contengan multiplicaciones matemáticas sin que por ello se ponga en peligro la fortaleza del cifrado.

Otra medida que se debe adoptar para prevenir estos ataques está en la función *xtime*, que se debe implementar para que se ejecute un número fijo de ciclos, independientemente de su valor. El algoritmo DES también es resistente frente a los ataques por análisis temporal aunque, a diferencia de *AES-Rijndael*, su resistencia no se basa en el modelo de representación de coeficientes. El DES no utiliza multiplicaciones matemáticas. Los dos algoritmos proporcionan una plataforma de cifrado robusta frente a ataques por análisis temporal, aunque, mientras *AES-Rijndael* basa su fortaleza en la calidad del diseño interno, DES basa la suya en las limitaciones a la hora de implementar funciones matemáticas.

D.6. Ataque diferencial (*Impossible Differentials Attack*).

Existe un ataque de este tipo a 5 rondas de AES, requiriendo 2^{29} texto a cifrar elegidos, 2^{30} procesos de cifrados, 2^{42} bytes de memoria, 2^{26} pasos de pre calculo. Estas condiciones fueron mejoradas en para alcanzar un ataque a 6 rondas de AES.

D.7. Número de rondas.

La estructura del el algoritmo AES es conocida públicamente en lo que respecta su estructura algebraica la cual se compone de las funciones mencionadas en el capitulo 2. Este numero de rondas fue determinado por los autores (Joan 2002). Ya que con un numero de rondas mínimo se ofrece una seguridad maxima frente a ataques conocidos.

El número de rondas se ha determinado en función del máximo número de rondas para el cual se pueden prevenir los ataques *shortcut* con un considerable margen de seguridad. Un ataque shortcut es más eficiente que uno por fuerza bruta. Para ejecuciones del algoritmo *Rijndael*se empleando una longitud de bloque y de clave de 128 bits, no ha sido encontrado ningún ataque shortcut para implementaciones con más de 6 rondas Para conseguir un margen de seguridad, se han añadido cuatro rondas más, por lo que para claves y bloques de 128 bits se emplean 10 rondas. Los motivos de esta decisión han sido los siguientes (Muñoz 2004):

Dos rondas de *AES-Rijndael* proporcionan una alta difusión en el sentido de que cada bit de un estado depende de los valores de todos los bits de las dos rondas anteriores, o lo que es lo mismo, un cambio en el valor de un bit de un estado concreto afecta a todos los bits de los estados de las dos siguientes rondas. La adición de cuatro rondas puede ser vista como una operación de alta difusión al principio y al final del cifrado. La alta difusión de cada una de las rondas de Rijndael depende de la estructura uniforme que opera en todos los bits de un estado. En los llamados

cifradores *Feistel*, las operaciones de una vuelta afectan únicamente a la mitad de los bits del estado, por lo que la propiedad de alta difusión es alcanza, en teoría, tras tres rondas, aunque en la práctica se necesitan cuatro o más rondas.

Generalmente los criptoanálisis lineales, los criptoanálisis diferenciales y los ataques con truncated differentials provocan una propagación a través de n rondas con el fin de atacar $n + 1$ ó $n + 2$ rondas. Este es el caso del ataque Square que emplea una estructura de propagación de 4 rondas para atacar 6 rondas. A este respecto, la adicción de cuatro rondas adicionales dobla el número de rondas a través del cual se puede dar una propagación (Deamen 1997).

Para versiones del algoritmo con claves de mayor longitud, el número de rondas aumenta en una unidad por cada 32 bits adicionales en la clave de cifrado principal. Esto se debe a las siguientes razones:

Uno de los objetivos principales es la erradicación de los ataques shortcut, es decir, ataques que son más eficientes que una búsqueda exhaustiva de la clave de cifrado. Cuanto mayor sea la clave, un ataque shortcut tendrá más carga de trabajo, por lo que se vuelve más ineficiente.

Los ataques empleando el conocimiento de parte de la clave principal o los ataques empleando claves relacionadas se basan en el conocimiento de los bits de la clave principal de cifrado o en la habilidad para probar reiteradamente distintas claves de cifrado. Si la clave de cifrado aumenta en tamaño, el rango de posibles claves también aumenta, dificultando así el trabajo del criptoanálisis. No se ha encontrado ningún ataque empleando el conocimiento de parte de la clave principal o el uso de claves relacionadas, que pueda amenazar la seguridad en versiones de *AES-Rijndael* con seis rondas, por lo que éste es un buen margen de seguridad. Para versiones del algoritmo con mayores tamaños de bloque, el número de rondas aumenta en una unidad por cada 32 bits adicionales en el tamaño del bloque. Esto se debe a las siguientes razones:

•Para un tamaño de bloque superior a 128 bits se necesitan 3 rondas para conseguir una alta difusión. El poder de difusión de cada vuelta disminuye al aumentar el tamaño de bloque.

• El aumento en el tamaño de bloque provoca un incremento en el rango de posibles patrones aplicables a la entrada o a la salida de una secuencia de rondas. Esta flexibilidad añadida permite la extensión de un ataque a una o más rondas. Se ha encontrado que dichas extensiones de los ataques en una simple vuelta están muy lejos de provocar cualquier fallo en seguridad. De cualquier forma, se ha adoptado un margen de seguridad.

D.8. Pesos de los conjuntos diferenciales y de los conjuntos lineales.

En (Massey 1991) se expone lo siguiente:

•El coeficiente de propagación de un conjunto diferencial puede ser aproximado mediante el producto de los coeficientes de propagación de sus S-Boxes activas.

•La correlación de un conjunto lineal puede ser aproximada mediante el producto de las correlaciones entrada-salida de las $S - Boxe$ activas.

La estrategia *Wide Trail* puede resumirse en lo siguiente:

•Se escoge una S-Box activa donde el máximo coeficiente de propagación y la máxima correlación entrada-salida sean lo más pequeñas posibles. En el caso del algoritmo *AES-Rijndael* estos valores son 2^6 y 2^3 respectivamente.

•Se construye la *layer* de difusión de forma que no haya conjuntos de múltiples rondas con pocas $S - Box$ activas.

Se ha probado que el mínimo número de $S - Box$ activas en cualquier conjunto lineal o diferencial de 4 rondas es 25. Esto nos da un coeficiente de propagación máximo de 2^{150} para cualquier conjunto diferencial de 4 rondas y un máximo de 2^{75} para la correlación en cualquier conjunto lineal de 4 rondas (Amparo 2001).

Esto se cumple para cualquier longitud de bloque de *AES-Rijndael* independientemente del valor de las subclaves. Señalar que la no-linealidad de la $S - Box$ elegida aleatoriamente entre el conjunto de las $S - Box$ inversibles de 8 bits va a ser menos óptima. Valores usuales para el ratio de propagación máximo son desde 2^5 hasta 2^4 y para la máxima correlación entrada-salida es de 2^2.

D.8.1. Propiedades de simetría a calves débiles.

Se han tomado medidas para evitar la simetría en el comportamiento del algoritmo. Esto se obtiene mediante las constantes de vuelta, que son diferentes para cada una de las rondas. El hecho de que el cifrador y su inversa empleen distintos componentes prácticamente elimina la posibilidad de existencia de claves débiles, como existen en el algoritmo DES. La no-linealidad de la función extensión de las subclaves (*KeyExpansion*) prácticamente elimina la posibilidad de existencia de claves equivalentes.

D.9. Ataque cuadrado (*Square Attack.*)

El mas potente ataque a *AES-Rijndael* a la fecha es el ataque *Square*, es un ataque dirigido a un algoritmo del tipo de *Rijndael* que basa su diseño en estructuras de bytes. Precisamente el primer ataque de este tipo fue hecho a el algoritmo predecesor llamado *Square*. Este ataque puede romper a *AES-Rijndael* de 6 a 7 rondas, que puede ser mejorado para atacar a 9 rondas de AES-259 con 2^{77} texto a cifrar, con 2^{56} claves relacionadas, y 2^{224} procesos de cifrado.

D.10. Ataque por colisiones (*Collision Attack*)

este ataque afecta a todas las versiones de AES, 128, 192 y 256 con 7 rondas.

Los ataques anteriores son de alguna manera el último intento de diseñar un ataque a *AES-Rijndael* de la manera tradicional. Es obvio que la introducción de estructuras algebraicas a AES por un lado deja atrás a los ataques más tradicionales, pero por el otro reta a encontrar nuevos ataques dirigidos a la nueva estructura algebraica. De manera natural los nuevos ataques son llamados ataques algebraicos, y como casi siempre en estos temas existen dos tendencias una de ellas que dice que solo son ideas que pueden no pueden considerarse aún como peligrosas, y la otra que dice que este tipo de ataques promete mucho.

En general este tipo de ataques consiste en dos etapas: la primera de coleccionar datos como los descifrados, los cifrados, las claves, valores intermedios, y expresa todo el algoritmo como ecuaciones que involucran los datos anteriores. La segunda es resolver las ecuaciones, donde la solución deberá ser información de la clave. Precisamente debido al diseño tan formal de *AES-Rijndael*, éste puede ser expresado de diferentes maneras de una forma elegante. Algunas ideas o ataques algebraicos que pueden ser listados.

D.11. Fracciones continuas

Una de las primeras propuestas puede ser vista como una fracción continua como la siguiente:

$$x = K + \sum \frac{c_1}{k^* + \sum \frac{c_2}{k^* + \sum \frac{c_3}{k^* + \sum \frac{c_4}{k^* + \sum \frac{c_5}{k^* + P_*^*}}}}}$$

Donde cada K es un byte que depende de varios bytes de la clave expandida, los C_i son constantes conocidas, y los K^* son exponentes o índices desconocidos, estos

valores depende de la suma que se efectúan. Una combinación de este tipo de de fórmulas describe totalmente al algoritmo AES, con 2^{26} incógnitas, sin embargo no se conoce un método práctico que resuelva este tipo de ecuaciones.

D.12. Ataque *XSL*

Se observo que las $S-box$ de AES pueden ser descritas por ecuaciones cuadráticas booleanas, es decir, si $x_1, x_2, ..., x_8$ son los bits de entrada y $y_1, y_2, ..., y_8$ los bits de salida entonces existe una función de la forma $f(x_1, x_2, ..., x_8, y_1, y_2, ..., y_8) = 0$, donde el grado de f es 2. El ataque se fundamenta que este tipo de ecuaciones sirven para el segundo paso del ataque algebraico y que existe un método que pueda resolverlas. Este problema esta considerado del tipo *Multivariate Quadratic Equations* que se sabe es un problema difícil de resolver. Sin embargo algunas opiniones mencionan que el trabajo de *Courtois* y *Pieprzyk* tiene imperfecciones, particularmente que no cuentan con las ecuaciones lineales suficientes para resolver el sistema. La complejidad estimada para el ataque en el mejor de los escenarios es de 2^{255} lo que equivale a la resistencia de la versión AES-256.

D.13. Ataque por combinaciones (*Embedding*)

Otra idea que fue expuesta es la de *Murphy y Robshaw*, tratando de encajar a AES en otra algoritmo llamado BES(*Big Encryption System*), de la siguiente manera:

$$\text{Rijndael}_k(x) = \phi^{-1}(\text{BES }_{\phi(k)}(\phi(x))$$

Donde K es la clave, x el mensaje y ϕ un mapeo de la estructura de campo de *AES-Rijndael* a la estructura de campo de BES. Como *AES-Rijndael* usa al combinaciones de los campos $GF(2^8)$ y $GF(2^8)$, BES solo usa el campo $GF(2^8)$. Los autores afirma que BES tiene mejor estructura algebraica y pudiera ser mas fácil su criptoanálisis, sin embargo no se puede afirmar que las sus propiedades puedan ser trasladadas a *AES-Rijndael*. Se afirma aquí también que el método *XSL* es puede ser aplicado a BES con ligeras mejorías en los resultados.

D.14. Ataque cifrado dual (*Dual Cipher*)

Otra manera de para atacar, es definir un algoritmo dual a *AES-Rijndael*, esto quiere decir una especie de traslación. Si tenemos los mapeos invertibles f, g, h entonces el cifrado Dual es definido como:

$$\text{Rijndael}_k(x) = f^{-1}(\text{Dual }_{g(k)}(h(x))$$

Por ejemplo, como la función cuadrado es lineal en los campos de característica 2, es natural que los duales inmediatos sean los cuadrados. Aunque se puede mostrar que los duales son equivalentes en seguridad, la idea es poder encontrar un ataque a algún dual para que después sea trasladado al original Rijndael, sin embargo por el momento no se ha encontrado alguna debilidad llamativa.

D.14.1. Claves débiles como en IDEA.

Las llamadas claves débiles son claves que resultan de un cifrador de bloque diseñado con debilidades detectables. El caso más conocido de claves débiles se da en el algoritmo IDEA (Daemen 1995). Estas debilidades son características de cifradores en los cuales las operaciones no lineales dependen del valor actual de la clave. Este no es el caso de *AES-Rijndael*, donde las claves se aplican usando funciones $EXOR$ y todos los elementos no lineales se encuentran agrupados en las S-Box. En *AES-Rijndael* no existen restricciones a la hora de seleccionar una clave.

D.15. El conjunto de posibles cifradores para una longitud de bloque y de clave dadas.

Un cifrador de bloque con una longitud de bloque v tiene un conjunto v de 2^v posibles entradas. Si la longitud de la clave es u, ésta define un conjunto de $U = 2^u$ permutaciones sobre $(0,1)^v$. El número de permutaciones posibles sobre $(0,1)^v$ es $V!$. Por lo tanto, el número posible de cifradores de bloque de dimensiones u y v es:

$$((2^v)!)^{2^n} \tag{D.2}$$

o su equivalente

$$(v!)^u \tag{D.3}$$

Para valores prácticos de las dimensiones (por ejemplo, v y u iguales a 40), el subconjunto de cifradores de bloque con debilidades explotables forman una minoría insignificante dentro del conjunto.

D.15.1. Factor de trabajo y requisitos de memoria para los ataques

Combinando las extensiones anteriores se obtiene como resultado un ataque basado en seis rondas. Aunque no es factible con la tecnología actual, este ataque es más

rápido que una búsqueda exhaustiva de la clave de cifrado, y por tanto relevante. El factor de trabajo y los requisitos de memoria se resumen en la cuadro D.1. Para diferentes tamaños de bloque de *AES-Rijndael* no se han encontrado extensiones de más de siete rondas que sean más rápidas que un ataque por fuerza bruta.

Ataque	Texto en claro	Cifrado	Memoria
Básico (4 rondas)	2^9	2^9	Poca
Extensión al final	2^{11}	2^{40}	Poca
Extensión al principio	2^{32}	2^{40}	2^{32}
Ambas Extensión	2^{32}	2^{72}	2^{32}

Cuadro D.1: Complejidad del ataque *Square*

D.15.2. Factor de trabajo y requisitos de memoria para los ataques

En (Jakobsen 1997), *Jakobsen* y *Knudsen* introducen un nuevo ataque sobre cifradores de bloque. En este nuevo tipo de ataque, el atacante construye polinomios usando pares de bytes de entrada-salida del cifrador. Este ataque es factible si los componentes del cifrador poseen una expresión algebraica compacta de tal forma que se pueda combinar para obtener expresiones de una complejidad manejable. Este ataque se basa en que si los polinomios construidos tienen un grado pequeño, solamente serán necesarios unos pocos pares de entrada-salida del cifrador para obtener los coeficientes del polinomio (dependientes de la clave). La compleja expresión de las S-Boxes con coeficientes pertenecientes a $GF(2^8)$, en combinación con los efectos de las layers de difusión hacen inviables este tipo de ataques para un número considerable de rondas. La expresión para la S-Box viene dada por:

$$63 + 8fx^{127} + b5x^{191} + 01x^{223} + f4x^{239} + 25x^{247} + f9x^{251} + 09x^{253} + 05x^{254} \quad \text{(D.4)}$$

Apéndice E

Sistema Seguridad

Los siguientes datos nos dicen algunos resultados más representativos que se han obtenido a lo largo de el estudio de *Rijndael*. Nos sirven como referencia para poder conocer las velocidades estándares con que se cuentan en nuestros días. El propósito de AES es que sea tomado como un estándar, es decir que la mayoría de aplicaciones puedan comunicarse de manera segura. En la actualidad ya una gran variedad de aplicaciones soportan AES y esto permite una mejor comunicación de manera segura, entre las aplicaciones que ya soportan AES están *SSL, IPsec, WinZip* entre otras y en muchas otras se planea soportarlo próximamente como en la nueva generación de *GSM*, etc.

Sin embargo es posible dada su construcción implementar AES de una manera propia, esto con el objeto de tener un algoritmo dedicado a cierta aplicación y aprovechar su diseño. Por ejemplo en sistemas cerrados que no necesitan ser compatibles con el exterior pueden modificar AES o cambiar algunos parámetros y obtener un nuevo algoritmo tan seguro como AES.

La primera idea es modificar la longitud del bloque de cifrado o la longitud de la clave, la longitud de la clave puede ser reducida por razones de seguridad hasta 80 bits, por razón de diseño del algoritmo lo más simple es tomar en lugar de 4 columnas, solo 3, es decir tomar una longitud de 96 bits. También cambiar el número de rondas, sabemos que AES es seguro con al menos 8 rondas. Otra sugerencia simple es cambiar el orden de las aplicaciones hechas en el proceso AES, es decir el lugar de aplicar la secuencia *ByteSub, ShiftRow, MixColumn, AddRoundKey*, cambiarla por ejemplo por *ByteSub, MixColumn, ShiftRow, AddRoundKey*. Hasta lo que se, esto no alteraría de alguna forma la seguridad de AES, solo cambiaría la salida.

1.- AES tiene los siguientes niveles de seguridad
Claves de 80, 112, 128, 192, y 256 bits.
2.- Usar claves de 80 bits será seguro hasta el año 2010, 112 hasta el año 2020, y 128 posteriormente. Aunque esta seguridad puede reducirse por el modo de operación de los algoritmos en consideración.

3.- Para 80 bits de seguridad AES, es equivalente a 1024 bits de *RSA*, y 163 de *ECDSA*.
4.- Se recomienda estandarizar los niveles de seguridad de todos los algoritmos en una aplicación, por ejemplo al usar AES 80, *RSA* 1024/ECDSA-160, y un *MAC* de 160 bits.
5.- Simple DES queda totalmente descontinuado, *TDES* con dos claves podrá ser soportado hasta el año 2010, *TDES* con 3 claves hasta 2020, de ahí se espera sólo usar AES con 128.
6.- Actualmente se revisan los modos de operación ya conocidos y se estudian *CCM (para el estándar IEEE 802.11)*y CTR como nuevos modos de operación. Así como los MACs[40] con AES son estudiados. Se estudia también un generados de bits aleatorios usando AES.
7.- Se habla de los ataques llamados algebraicos que en nuestros días no son operables, sin embargo se presume pueden ser una línea de criptoanálisis eficiente en el futuro, aunque hoy en día hay más preguntas que respuestas respecto a este tipo de ataques.
8.- Como un punto complementario del estado actual de AES podemos mencionar varias aplicaciones que ya han asumido a AES en sus esquemas, algunas de ellas se describen en los documentos RFC que pueden consultarse al final de la bibliografía. Entre las aplicaciones están *TLS, IPsec, IKE* .

E.1. Sistema Seguridad IPv6

Si tuviéramos que aumentar las principales características de Internet, diríamos, en primer lugar, que es una red totalmente abierta y pública, sin ningún tipo de jerarquía establecida ni de autoridad central. El número de usuarios aumenta día a día de forma espectacular, así como el tipo de operaciones realizadas. Podríamos decir que Internet no es más que una red general formada por la interconexión de multitud de redes. Con las características indicadas, es difícil de imaginar la inseguridad de los datos transmitidos y almacenados en los computadores conectados. En las distintas capas del modelo Internet, se pueden establecer diferentes controles de seguridad atendiendo al tipo de los datos. Por un lado, en el nivel IP hay que establecer un modelo de seguridad que controle los paquetes que circulan por la red; los servicios de seguridad que se puede proporcionar son: autenticación, integridad, confidencialidad y control de acceso. Por otro lado, tenemos que atender la seguridad en las aplicaciones; es decir, la seguridad de los datos de los usuarios. Aquí se pueden proporcionar todos los servicios de seguridad. También es necesario atender la seguridad global del sistema terminal y su entorno local . En este apartado vamos a ver la seguridad en la capa IP . El protocolo utilizado en la actualidad, IPv4, no contemplaba originalmente funciones y mecanismos de seguridad .No ocurre lo

mismo con el futuro protocolo IPv6,que ha sido diseñado prestando una atención especial a estos aspectos .Alguno de los elementos de seguridad introducidos en las definiciones de IPv6 han sido recogidos e implantados en IPv4. La seguridad en las aplicaciones será estudiada en posteriores apartados. La seguridad del sistema queda fuera de los objetivos del libro.

E.2. Certificados X.509 y autoridades de certificación

Para que una clave pública se pueda considerar válida y asociada a un usuario determinado debe tener un certificado que así lo demuestre .El sistema utilizado más difundido es el de certificados de clave pública firmados por una Autoridad de Certificación (AC) según la norma X.509.El esquema general está mostrado en la figura.

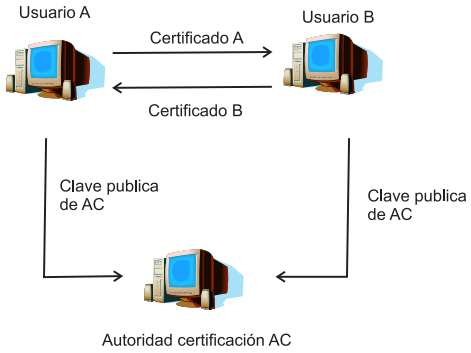

Figura E.1: Sistema de autentificación mediante certificador de clave publica

Para diseñar una AC es necesario definir una política de certificación donde se indique el ámbito de actuación de los certificados emitidos por esa AC y la estructura asociada (niveles de jerarquía, etc.). Asimismo, se debe decidir el modo de verificación de la calve de el usuario. Una vez definida la política de certificación, la implantación de una nueva AC es simple. La AC debe generar su propio par de claves, públicas y privadas, guardar de forma segura la clave privada y certificarse su propia clave pública. Ese certificado debe permanecer en algún lugar de acceso público, como puede ser el directorios.500 o una página Web.

Los actuales navegadores de Internet llevan incorporadas algunas herramientas que permiten una instalación simple y cómoda de los certificados. Como consejos generales, a la hora de implantar una AC se deben tener en cuenta los siguientes puntos:

Nunca se debe conectar a la red el computador donde reside la autoridad de certificación.

No automatizar el proceso de firma de certificados, para lo que es recomendable la presencia física del usuario o, en su defecto, una adecuada documentación.
Restringir el acceso de los usuarios locales a la AC.
Restringir el acceso físico a la AC.
Proteger el equipo de interferencias de radiofrecuencias.
No guardar la clave privada de la AC sin cifrar (no debemos olvidar que es la utilizada para realizar la firma).
Procurar que los usuarios de la AC utilicen claves de longitud adecuada (1024 bits).

E.3. Seguridad en Internet.

La popularización de Internet, que con lleva un aumento en el número de usuarios que desconocen los peligros de la red, los hace potencialmente cómplices de este novedoso ataque el cual consta en romper la seguridad de un texto cifrado, en el cual la ingenuidad de algunos usuarios de la red puede llegar a convertirse, si no se remedia, en el mayor arma de criptoanálisis de la historia, ya que cualquier algoritmo es susceptible de sufrir este ataque.

La idea básica de este ataque se basa en el uso de Internet para llevar a cabo un ataque por fuerza bruta contra un criptograma en un tiempo razonable.

Todo algoritmo es susceptible de sufrir un ataque por fuerza bruta. La manera de prevenir este tipo de ataques es aumentar el tamaño de la clave, con lo cual se incrementa el número total de claves y por tanto el tiempo total para probarlas todas. A pesar de los avances en la potencia computacional, los algoritmos modernos, con claves de un tamaño considerable, consideran que un ataque por fuerza bruta emplearía una media de cientos o incluso miles de años en probar todas las claves y encontrar la clave correcta. Los creadores de este ataque utilizan los siguientes pasos para poder reducir esos cientos de años a un tiempo aceptable.

• Una vez conseguido un criptograma e identificado el algoritmo con el cual se ha llevado a cabo el cifrado, se divide el espacio total de claves en pequeños paquetes de un reducido número de claves de cifrado, que todo usuario doméstico pueda probar en su computador personal sin que le suponga una pérdida excesiva de tiempo.

• Se crea una página web en Internet que permite descargar estos pequeños paquetes y el criptograma que interesa descifrar.

•Se publica el sitio web en foros y portales de Internet bajo la caracterización de un portal de juegos on-line en el que existirá un juego, al estilo de la lotería, que consiste en descifrar el criptograma. La forma de atraer a las víctimas consiste en

ofrecer algún tipo de premio a la persona que consiga descifrarlo. El usuario sólo debe descargar a su computador un paquete de claves y, si empleando alguna de ellas consigue descifrar el criptograma, habrá ganado el premio. Para reclamar el premio se debe devolver el texto en claro e introducir sus datos para recibir el supuesto premio. Así, cuantos más paquetes de claves descargue el usuario, más posibilidades tendrá de ganar, con lo cual, la ambición innata del hombre hace que el total de los paquetes de claves sea probado en un tiempo accesible.

Es ataque puede llevarnos a métodos científicos en el criptoanálisis, pero esta misma sencillez es la que puede llegar a convertirlo en uno de los ataques más peligrosos en un futuro próximo. A nadie se le había ocurrido la idea de emplear un reclamo de un juego on-line con un premio como incentivo para lograr colaboradores en la labor de la búsqueda exhaustiva de la clave. Cualquier algoritmo es susceptible de sufrir este tipo de ataque. Lo único que impide que un ataque por fuerza bruta tenga éxito es el factor tiempo que, debido a este nuevo método de ataque, se ha visto considerablemente reducido.

En el caso del algoritmo DES, ya se ha demostrado que con la potencia computacional actual, es viable un ataque por fuerza bruta normal, empleando una computadora, ya que su tamaño de clave reduce el espacio de claves a 2^{56}(Henk 2000).

$$2^{56} = 7{,}2058e + 016 \text{ claves del algoritmo DES}$$

En el caso del algoritmo *Rijndael*, el espacio de claves es notablemente mayor. El espacio de claves depende del tamaño de la misma y puede abarcar de las 2^{128} a las 2^{192} ó 2^{256} combinaciones posibles, lo cual hace inviable un ataque por fuerza bruta normal, ya que se tardaría cientos de años en descubrir la clave de cifrado. Ahora bien, este nuevo método multiplica infinitamente la potencia computacional disponible, por lo que en un futuro se podría disponer de un ataque viable sobre Internet. Hoy por hoy, el espacio de claves es demasiado grande, por lo que el número de usuarios trabajando al mismo tiempo debería ser tan grande que convierte este tipo de ataque en una utopía.

E.3.1. Seguridad en redes de comunicaciones.

La creciente expansion de las redes de comunicaciones ha hecho necesario el desarrollo de herramientas de seguridad que protejan de los posibles ataques que puedan sufrir tanto los datos transmitidos como el acceso a los elementos de la red. Son principalmente las redes las que debido a sus características y su estructura pueden ser blanco de diferentes ataques, por ejemplo, la ausencia de conexión física directa entre el emisor y el receptor, que no asegura la inmediata transmisión entre

ambos. Los ataques a los sistemas de comunicación se agrupan en cuatro categorías:

•Interrupción: Algún elemento del sistema esta fuera de servicio.

•Intercepción: Alguien no autorizado accede a cierta información o elemento de la red.

•Modificación: Alguien no autorizado, tras haber accedido aún mensaje o elemento de la red altera su contenido.

•Fabricación: Alguien no autorizado, falsificando su identidad inserta información al sistema.

El ataque por intercepción se considera del tipo pasivo, y el resto de tipo activo. Para poder detectar y prevenir estos ataques las redes deben incorporar mecanismos de seguridad siendo estos:

E.3.2. Aplicaciones de seguridad en redes de comunicaciones.

En los sistemas convencionales o simétricos, para el caso de redes de paquetes conmutados, la confidencialidad de la información transmitida se puede conseguir mediante un cifrado simétrico que debe realizarse a nivel de enlace o extremo a extremo como el mostrado en la Figura 4.1. En donde se muestran los elementos de seguridad para el caso de cifrado simétrico en una red de comunicación de paquetes. En el caso de un cifrado a nivel de enlace se necesitan equipos de cifrado en cada nodo de la red, tanto a su entrada como a su salida . El número de elementos cifrados es muy alto como se muestra en la figura, pero permiten asegurar la confiabilidad del tráfico de mensajes. Sin embargo, la seguridad en los nodos de comunicación puede disminuir, ya que el mensaje ha de ser descifrado, para leer la dirección destino que aparece en la cabecera del paquete y la red pública no tiene ningún control sobre los nodos. En el caso de los nodos se tiene una clave que los caracteriza la cual debe ser clara para poder tener la ubicación correcta de estos. En la figura se muestra un esquema de cifrado simétrico en una red convencional. Para el caso de dos computadoras se omiten las nodos por lo que se facilita la transmisión.

Bibliografía:

Joan Daemen and Vincent Rijdael (2002). ***The Desing of Rijdael AES-The avanced Encryption Standard*** .Springer. Germany.

Francisco Pais Suárez (2003). ***Estudio del cifrado Rijndael***. Universidad de curuña. Facultad Informática. España.

Alfonso Muñoz Muñoz (2004). ***Seguridad Europea para EEUU Algoritmo de cifrado criptográfico Rijndael***.Madrid. España.

José de Jesús Ángel Ángel (2005) ***AES The avanced Encryption Standard***. CINVESTAV. México.

Alfred J. Meneses and Paul G. Oorschot. (1997). ***Hand Book Applied Cryptography***. CRC. Florida USA.

Douglas R. Stinson (1995). ***Cryptography Theory and Practice***, CRC. Florida USA.

Henk C.A. van Tilborg (2000). ***Fundamentals of Cryptography A professional Reference and interactive tutorial***, Kluwer. Massachusetts.USA.

Bruce Schneier (1996). ***Applied Cryptography Protocols, Algoritthms and source code in C*** 2da edition. Wiley .USA .

Jesús de Marcelo Ridoa (2002). ***Piratas cibernéticos Seguridad informática e Internet***. Alfaomega. España.

Jose Pastor F. and Miguel A. Sarasa L. (1998). ***Criptografía Digital Fundamentos y aplicaciones*** .Universidad de Zaragoza. España.

Hand Delfs and Helmut Kenebl (2002). ***Introduction to Cryptography principles and Applications***. Springer New York USA.

Michael Welschenbach (2001). ***Cryptography in C and C++.*** Apress. New York USA.

Pino Caballero Gill (2003***). Introducción a la criptografía*** 2da edición. Alfaomega. España.

Amparo S.(2001) ***. Técnicas Criptográficas de Protección de Datos*** 2da edición. Alfaomega. España.