AES - Advanced Encryption Standard

VERSIÓN 2005, PRINCIPIANTES

José de Jesús Angel Angel
jjangel@computacion.cs.cinvestav.mx
jjaa_@hotmail.com

AES es el nuevo estándar de cifrado simétrico dispuesto por el NIST, después de un periodo de competencia entre 15 algoritmos sometidos. El 2 de Octubre de 2000 fue designado el algoritmo Rijndael como AES, el estándar reemplazó de TDES, para ser usado en los próximos 20 años. Este reporte describe de forma detallada el algoritmo y algunas de sus características.

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). Rijndael fue elegido después de pasar un periodo de análisis durante aproximadamente 3 años, Rijndael fue elegido como la mejor opción dentro de 15 candidatos, sus principales características fueron su fácil diseño, su versatilidad en ser implementado en diferentes dispositivos, así como ser inmune a los ataques conocidos hasta la fecha, soportar bloques de datos de 128 bits y claves de 128, 192, y 256 bits. La idea básica general es tener un estándar que mejore el "performance" de TDES y sea resistente a los ataques conocidos.

La descripción de AES que llevaremos toma el siguiente orden:

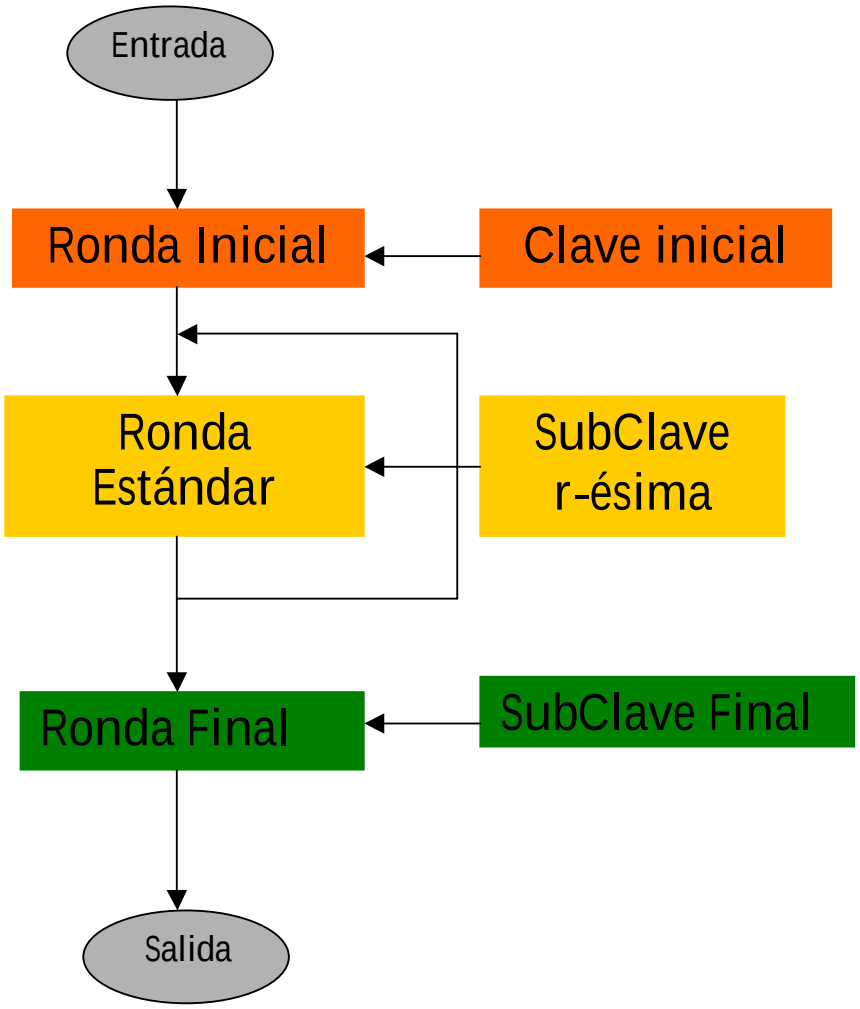
1.- Introducción.

AES (Advanced Encryption Standar) es el nuevo estándar de criptografía simétrica adoptado en el FIPS 197[34](Federal Information Processing Standards). En este reporte estamos comprometidos a poder dar de la manera más simple la descripción total del algoritmo, y dar algunas características de importancia.

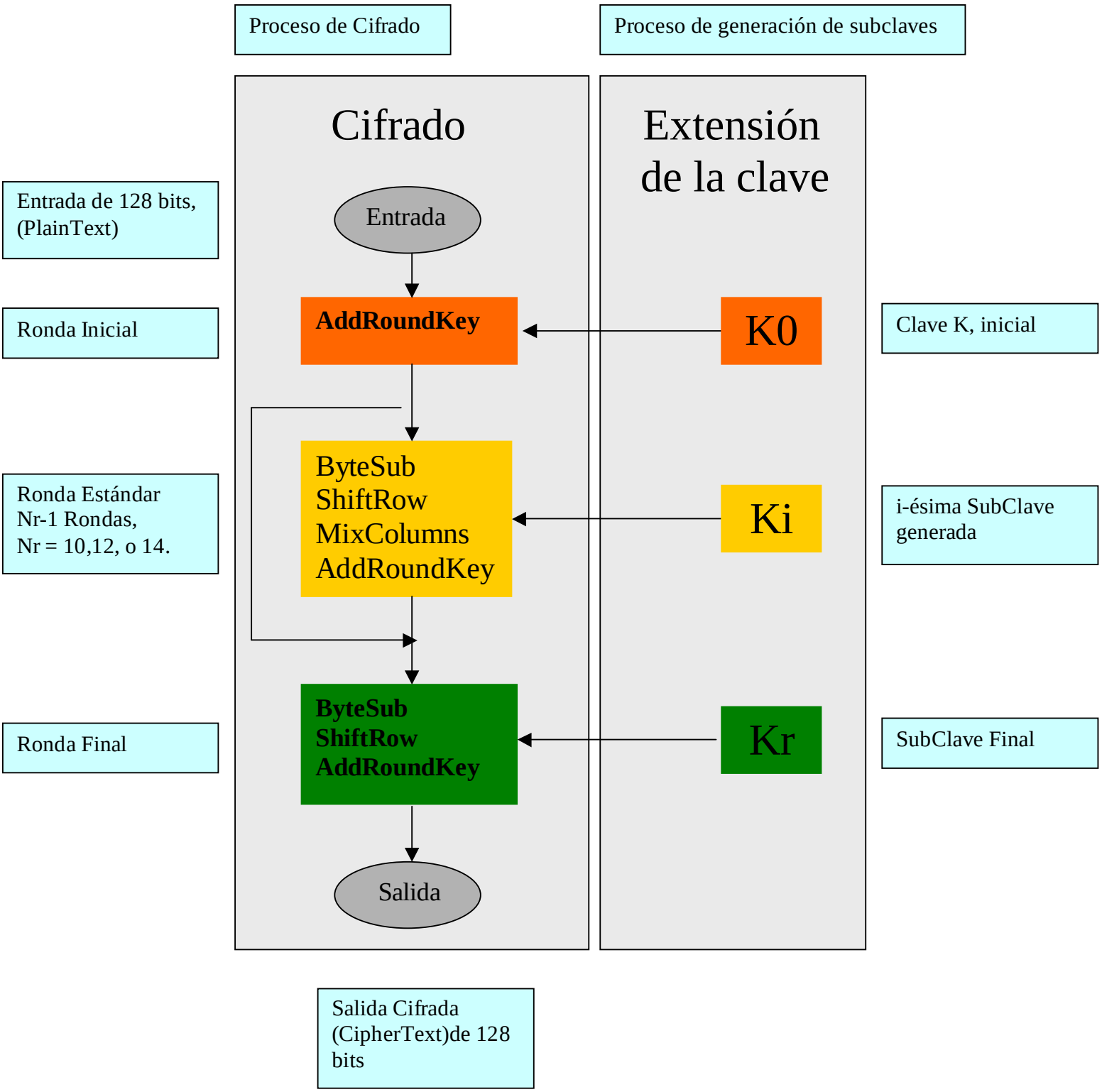
Desde 1977 que apareció la primera versión del estándar FIPS 46, asume como estándar el algoritmo DES (Data Encryption Standar), y sus posteriores reafirmaciones en 1983, 1988, 1993, y 1999. Casi siempre había visto opiniones controversiales de DES, sin embargo nunca fue dado un ataque que derivara por completo la clave secreta partiendo de la información pública, pero su corta longitud de clave lo comprometía poco a poco. La última reafirmación de DES en octubre de 1999 realmente fue suplantado por TDES, que es una versión múltiple de DES, designado como TDEA (Triple Data Encryption Algorithm). De hecho, ya se tenían planes de encontrar un reemplazo definitivo a DES. A pesar de un número grande de algoritmos que en la época estaban presentes como: IDEA, RC5, skipjack, 3-way, FEAL, LOKI, SAFER, SHARK,... NIST decidió convocar a un concurso que tuvo como principales objetivos obtener un algoritmo simétrico que garantice su seguridad para los próximos 20 años a partir del año 2000. La convocatoria apareció el 2 de enero de 1997, se admitieron 15 algoritmos, en agosto de 1998 en la primera conferencia AES se discutieron los algoritmos sometidos y posteriormente en la segunda conferencia AES en marzo de 1999, se realizaron los últimos comentarios. Para que en agosto de 1999 se comunicaran los 5 finalistas: MARS, RC6, Rijndael, Serpent y Twofish. En abril del 2000 se llevó a cabo la tercera conferencia AES, recibiendo los últimos análisis, para que finalmente el 2 de octubre del año 2000 se diera a conocer el ganador y se dispuso al Algoritmo RIJNDAEL como AES [46]. Esto llego a ser asumido oficial en noviembre 26 el 2001 en el FIPS 197. A partir de esa fecha hay conferencias especiales para analizar la situación actual de AES, la última se llevada a cabo en mayo del 2004.

El algoritmo 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 a permitido que AES sea adaptado poco a poco, desde los protocolos más usados como SSL, hasta las aplicaciones más especializadas, como VoIP.

La descripción de AES es simple si se cuentan 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 la siguiente figura:



De manera un poco más detallada el algoritmo AES llega a ser:



Entonces la descripción de AES consiste de dos partes, en describir el proceso de “Cifrado”, y el proceso de “Generación de las subclaves” o “Extensión de la clave K”. El bloque de cifrado tiene una longitud de 128 bits, la longitud de la clave K varia de 128, 192 y 256 bits, en cada caso AES tiene 10,12, y 14 rondas respectivamente.

El proceso de cifrado consiste esencialmente en la descripción de las 4 transformaciones básicas de AES: ByteSub, ShiftRow, MixColumns, y AddRoundKey. Es importante mencionar que el caso de Rijndael las funciones o transformaciones básicas son ligeramente diferentes en el proceso de descifrado, sin embargo es poco el esfuerzo necesario para poder comprender todo. Los autores del algoritmo escribieron un libro que se ha tomado como referencia oficial donde se describe con mayor profundidad varios aspectos aquí expuestos[46].

2.- Antecedentes Matemáticos.

En esta sección daremos los elementos matemáticos necesarios para poder entender AES. Los elementos que describimos en esta sección serán los siguientes:

2.1 El campo finito $GF(2^8)$.
2.2 Construcción del campo finito $GF(2^8)$.
2.3 El anillo $GF(2^8)[x]/(x^4+1)$.

2.1 El campo finito $GF(2^8)$

Esta sección es muy importante para poder entender el funcionamiento de AES, ya que AES trabaja como elementos básicos a los bytes (conjunto de 8 bits), AES ve a los bytes como elementos del campo finito $GF(2^8)$, nuestro propósito es explicar que significa hacer esto.

Comenzamos recordando lo siguiente:

Los conjuntos de números más conocidos son: los números enteros $\mathbf{Z}$, los números racionales $\hat{a}$, los números reales $ß$, y los números complejos $á$. Los números enteros $\mathbf{Z}$, son los números enteros positivos y negativos $\mathbf{Z} = \{\ldots -2, -1, 0, 1, 2, \ldots\}$, los números racionales se construyen anexando los inversos multiplicativos que no hay en $\mathbf{Z}$, así la definición de los números racionales queda como:

$$\hat{a} = \left\{ \frac{a}{b} \mid a, b \in \mathbf{Z},\ b \neq 0 \right\}.$$

Los racionales tienen una excelente propiedad algebraica, llamada estructura de campo, concretamente quiere decir que tanto con la operación suma, como la operación producto, tiene las siguientes propiedades:

$a+b=b+a$	conmutatividad	$a\times b=b\times a$
$a+(b+c)=(a+b)+c$	asociatividad	$a\times(b\times c)=(a\times b)\times c$
$a+0=a$	existe el neutro	$a\times 1=a$
$a+(-a)=0$	existe el inverso	$a\times\frac{1}{a}=1$

Además de una propiedad más que une las dos operaciones y es la propiedad distributiva $a\times(b+c)=a\times b+a\times c$.

De los conocimientos básicos sabemos que tanto el conjunto de los números Reales **ß**, como los complejos **á**, también cumplen estas propiedades. Esto quiero decir que **â**, **ß** y **á** son campos.

Otros conjuntos de números diferentes a los anteriores que forman un campo, estudiados en matemáticas son, por ejemplo el conjunto de números de la forma $\left\{a,b\in \mathbf{ß} \mid a+b\sqrt{-D}\right\}$ para D adecuados, también son un campo, de hecho podemos observar que si D =1, entonces tenemos a los complejos.

En la práctica se puede observar que dotar a un conjunto de la suma y verificar sus propiedades no es muy difícil y en muchos casos lo más complicado es poder mostrar que los elementos diferentes de cero tienen inverso multiplicativo dado el producto, de hecho en la criptografía esto es una propiedad muy recurrente.

Los ejemplos anteriores son usados en una amplia gama de aplicaciones, particularmente en la criptografía, sin embargo en nuestro caso, el caso de AES, lo que nos ocupa es el conocer el campos finitos $GF(2^8)$. Por lo tanto enseguida daremos los elementos para poder entender como surgen los campos finitos, para esto demos las siguientes definiciones preliminares.

Recordemos como se define el conjunto $\mathbf{Z}_n$ que es el conjunto de residuos módulo n, es decir $\mathbf{Z}_n=\{[a]\mid 0\le a<n\}$, y $[a]=\{b\mid \exists\, s\in \mathbf{Z},\ b=sn+a\}$, hagamos algunos

Ejemplos:

$\mathbf{Z}_2=\{[0],[1]\}$, en este caso [0] son todos los números enteros donde existe s, tal que b = s2, es decir la división entre 2 deja un residuo cero, otra manera de decirlo son todos los números pares. [1] son los números b tales que

existe un s y b = s2+1, es decir los números que dejan como residuo el 1, ó otra forma, son los números impares.

$\mathbf{Z}_3 = \{[0],[1],[2]\}$, [0] son todos los números b tales que existe un s, y b = 3s, es decir los múltiplos de 3, o dejan residuo 0 al dividirlo por 3.
[1], son los números b tales que dejan residuo 1 al dividirlos por 3.
[2], son los números b tales que dejan residuo 2 al dividirlos por 4.

$\mathbf{Z}_4 = \{[0],[1],[2],[3]\}$, [0] son los números b tales que existe un s, y b=4s.
[1], son los números b que dejan residuo 1 al dividirlos por 4.
[2], son los números b que dejan residuo 2 al dividirlos por 4.
[3], son los números b que dejan residuo 3 al dividirlos por 4.

Por razones de simplificación omitiremos la notación [_], dando por entendido que en $\mathbf{Z}_n$, se trabaja con clases y no con números simples.

Otra manera de ver los elementos de $\mathbf{Z}_n$, es colocarlos de la siguiente forma, por ejemplo para $\mathbf{Z}_5$

$$\begin{array}{c} \ldots \\ -5\ -4\ -3\ -2\ -1 \\ 0\ \ 1\ \ 2\ \ 3\ \ 4 \\ 5\ \ 6\ \ 7\ \ 8\ \ 9 \\ \ldots \end{array}$$

De el anterior ejemplo podemos inferir que –1 en $\mathbf{Z}_5$ es 4, en general –1 en $\mathbf{Z}_n$ es n-1, -2=n-2,...

De manera natural podemos dotar a $\mathbf{Z}_n$ de las operaciones de suma y producto, [a]+[b]=[a+b], [a][b]=[ab].

Ahora podemos preguntarnos cuándo $\mathbf{Z}_n$ es campo, sin mucho problema podemos chequear para qué n's $\mathbf{Z}_n$ es campo, y se deriva de los siguientes hechos:

1) Es fácil ver que $(\mathbf{Z}_n, +)$ satisfacen las primeras propiedades de campo, es decir que $(\mathbf{Z}_n, +)$ sea un grupo abeliano.

2) es fácil ver que ($\mathbf{Z}_n$, x) es asociativo, que es conmutativo, que existe la identidad, sin embargo no siempre existen los inversos multiplicativos.

En el caso de que existan los inversos multiplicativos en $\mathbf{Z}_n$, entonces $\mathbf{Z}_n$ será un campo, por lo que nos toca investigar cuándo existen los inversos y esto se deduce del siguiente hecho:

Teorema: (Algoritmo de Euclides) sean a, n, números enteros, entonces siempre existen números enteros q,r tales que $n = qa+r$.

El anterior teorema puede ser mostrado de manera inmediata, y es usado para mostrar el siguiente algoritmo:

Algoritmo extendido de Euclides: dados dos números enteros a, n, y d su máximo común divisor, entonces siempre existen s,t tales que $d = sa+tn$.

Si a es primo relativo a n, entonces su máximo común divisor es el 1, entonces del Teorema extendido de Euclides, existen números s,t tales que $1 = sa+tn$, al aplicar la clase de equivalencia modulo n, tenemos [1] = [s][a], ya que $[n] = 0$, es decir que. $[s]=[a]^{-1}$

Lo anterior significa que un número a en $\mathbf{Z}_n$ tiene inverso multiplicativo si y sólo si a es primo relativo a n.

Se sigue entonces que $\mathbf{Z}_n$ es campo sí y sólo si, n es número primo, ya que en este caso todos los números $0 < a < n$ son primos relativos a n, y por lo tanto tienen inverso multiplicativo.

Ejemplos:

$\mathbf{Z}_2$ es campo, y sus operaciones son las siguientes:

+	0	1
0	0	1
1	1	0

•	0	1
0	0	0
1	0	1

Sabemos también que $\mathbf{Z}_3, \mathbf{Z}_5, \mathbf{Z}_7$ son campos.

Para $\mathbf{Z}_3 = \{0,1,2\}$, el inverso de 2 es el propio 2.

Para $\mathbf{Z}_5 = \{0,1,2,3,4\}$, el inverso de 2 es el 3, el inverso de 4 es el mismo 4.
Para $\mathbf{Z}_7 = \{0,1,2,3,4,5,6\}$, el inverso de 2 es 4, el inverso de 3 es 5, el de 6 es el mismo 6.

También sabemos que $\mathbf{Z}_4, \mathbf{Z}_6$ no son campos ya que:

Para $\mathbf{Z}_4 = \{0,1,2,3\}$, el 2 no tiene inverso multiplicativo.
Para $\mathbf{Z}_6 = \{0,1,2,3,4,5\}$, el 2 tampoco tiene inverso multiplicativo.

En resumen los primeros ejemplos de campos finitos son de la forma $\mathbf{Z}_p$, con p número primo.

En seguida damos un teorema que nos indica que forma tiene todos los campos finitos, y de ahí podremos hablar del campo finito que nos interesa para AES.

Afirmación: todo campo finito tiene p^n elementos, y se puede construir a partir de $\mathbf{Z}_p$.

Observe que la inversa no es cierta, es decir, si existe un conjunto con p^n elementos no necesariamente es campo, como el caso de $\mathbf{Z}_4$.

Veamos ahora algunas definiciones y resultados que nos ayudará a construir un campo finito.

Definición: sea **F** un campo, entonces **F**[x] es el conjunto de polinomios con indeterminada x.

Podemos definir de manera convencional en **F**[x] las operaciones de suma y producto. De hecho **F**[x] es un anillo conmutativo con identidad, es decir, solo le falta que todos sus elementos tengan inverso para ser campo.

Definición: sea f(x) un polinomio en **F**[x], entonces f es irreducible sobre **F**, si f tiene grado positivo y si f=gh, g,h en **F**[x], entonces g o h son constantes.

Ejemplos

Sea $f \in \mathbf{Z}_2[x]$, con $f(x)=x^5+x^3+x+1$ es un polinomio con indeterminadas en x y coeficientes en $\mathbf{Z}_2$.

Sea $f \in \mathbf{Z}_2[x]$, con $f(x)=x^2+1$, entonces f es reducible ya que $f(x)=(x+1)(x+1)=x^2+1$ en $\mathbf{Z}_2$.

Sea $f \in \mathbf{Z}_3[x]$, con $f(x)=x^2+1$, entonces f es irreducible en $\mathbf{Z}_3$, ya que ni el 0, ni el 1 es raíz de f(x).

Observación importante: Sabemos que $\mathbf{Z}_p$ es un campo cuando p es primo, sabemos esto porque todos los elementos tienen inverso. Otra manera de establecer lo mismo es la siguiente: sea $\mathbf{Z}$ el anillo de los números enteros, y $(p\mathbf{Z})$ el conjunto de múltiplos de p, es decir ...,-p,0,p,2p,3p,... entonces construyamos el conjunto cociente $\mathbf{Z}/p\mathbf{Z}$, es decir el conjunto de las clases de equivalencia [a] de $\mathbf{Z}$ módulo $p\mathbf{Z}$, es el siguiente conjunto $[a]=\{b \in \mathbf{Z} | b-a \in p\mathbf{Z}\}$, es decir $b \in [a]$ si y sólo si $b=kp+a$, es decir b tiene residuo a al ser dividido por p.

Lo anterior quiere decir que el campo $\mathbf{Z}_p$ se construyó a partir de tomar el cociente de un anillo $(\mathbf{Z})$ módulo $(p\mathbf{Z})$, que en álgebra se le llama Ideal. Esta observación nos hace pensar que para la construcción de otros campos se sigue el mismo procedimiento, en efecto, en la siguiente importante resultado nos dice como se construyen campos finitos tomando el cociente de un anillo módulo un Ideal generado por un elemento primo, que en el caso de polinomios se denomina irreducible. Claramente el cociente $\mathbf{Z}/n\mathbf{Z}=\mathbf{Z}_n$.

Teorema: sea $f \in \mathbf{F}[x]$, entonces el cociente $\mathbf{F}[x]/(f)$ es campo si y sólo si f es irreducible.
El teorema equivalente para enteros queda así : $\mathbf{Z}/n\mathbf{Z}=\mathbf{Z}_n$ es campo si y sólo si n es primo (lo equivalente a irreducible en polinomios).

El teorema anterior nos dice que basta tomar un polinomio irreducible en $\mathbf{F}$ para poder construir otros campos, particularmente tenemos que si $gr(f)=n$, y $\mathbf{F}=\mathbf{Z}_p$, entonces $\mathbf{F}[x]/(f)$ es un campo finito de p^n elementos. Más aún se puede mostrar que cualquier campo finito tiene p^n, entonces siempre es posible construir un campo finito de la anterior manera. Estos campos se denotan como $\mathbf{F}_{p^n}$ o también como $GF(p^n)$ de "Galois Field".

Finalmente, ya podemos afirma que existe un campo finito de $2^8=256$ elementos que denotaremos también como $GF(2^8)$, que es el campo principal a donde trabaja AES.

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

Una de las cosas más importantes de AES es conocer muy bien el campo $GF(2^8)$, así como sus operaciones y su representación. Sabemos de el anterior punto que es necesario encontrar un polinomio irreducible con coeficientes en $\mathbf{F} = \mathbf{Z}_2$ de grado 8.

Antes de continuar observemos que cualquier polinomio $f \in \mathbf{Z}_2[x]$ se puede ver como una lista de bits, AES toma por convenio el siguiente orden

$$(10001111) \sim x^7+x^3+x^2+x+1$$
$$(11001100) \sim x^7+x^6+x^3+x^2$$
$$(10101010) \sim x^7+x^5+x^3+x$$

Es decir

Si $p(x) \in GF(2^8)$, entonces

$p(x)=b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

corresponde a el byte $(b_7b_6b_5b_4b_3b_2b_1b_0)$

a continuación listamos todos los polinomios irreducibles hasta grado 8, los últimos nos servirán para la construcción de $GF(2^8)$.

Polinomios irreducibles de grado 2

Binario	Hex	Polinomio	orden
111	7	x^2+x+1	3

Polinomios irreducibles de grado 3

Binario	Hex	Polinomio	orden
1011	b	x^3+x+1	7
1101	d	x^3+x^2+1	7

Polinomios irreducibles de grado 4

Binario	Hex	Polinomio	orden
10011	13	x^4+x+1	15
11001	19	x^4+x^3+1	15
11111	1f	$x^4+x^3+x^2+x+1$	5

Polinomios irreducibles de grado 5

Binario	Hex	Polinomio	orden
100101	25	x^5+x^2+1	31
101001	29	x^5+x^3+1	31
101111	2f	$x^5+x^3+x^2+x+1$	31
110111	37	$x^5+x^4+x^2+x+1$	31
111011	3b	$x^5+x^4+x^3+x+1$	31
111101	3d	$x^5+x^4+x^3+x^2+1$	31

Polinomios irreducibles de grado 6

Binario	Hex	Polinomio	orden
1000011	43	x^6+x+1	63
1001001	49	x^6+x^3+1	9
1010111	57	$x^6+x^4+x^2+x+1$	21
1011011	5b	$x^6+x^4+x^3+x+1$	63
1100001	61	x^6+x^5+1	63
1100111	67	$x^6+x^5+x^2+x+1$	63
1101101	6d	$x^6+x^5+x^3+x^2+1$	63
1110011	73	$x^6+x^5+x^4+x+1$	63
1110101	75	$x^6+x^5+x^4+x^2+1$	21

Polinomios irreducibles de grado 7

Lista de polinomios irreducibles en GF(2^8)

Binario	Hex	Polinomio	Orden
10000011	83	x^7+x+1	127
10001001	89	x^7+x^3+1	127
10001111	8f	$x^7+x^3+x^2+x+1$	127
10010001	91	x^7+x^4+1	127
10011101	9d	$x^7+x^4+x^3+x^2+1$	127
10100111	a7	$x^7+x^5+x^2+x+1$	127
10101011	ab	$x^7+x^5+x^3+x+1$	127
10111001	b9	$x^7+x^5+x^4+x^3+1$	127
10111111	bf	$x^7+x^5+x^4+x^3+x^2+x+1$	127
11000001	c1	x^7+x^6+1	127
11001011	cb	$x^7+x^6+x^3+x+1$	127
11010011	d3	$x^7+x^6+x^4+x+1$	127
11010101	d5	$x^7+x^6+x^4+x^2+1$	127
11100101	e5	$x^7+x^6+x^5+x^2+1$	127
11101111	ef	$x^7+x^6+x^5+x^3+x^2+x+1$	127
11110001	f1	$x^7+x^6+x^5+x^4+1$	127
11110111	f7	$x^7+x^6+x^5+x^4+x^2+x+1$	127
11111101	fd	$x^7+x^6+x^5+x^4+x^3+x^2+1$	127

Polinomios irreducibles de grado 8

Lista de polinomios irreducibles en 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
100101101	12d	$x^8+x^5+x^3+x^2+1$	255
100111001	139	$x^8+x^5+x^4+x^3+1$	17
100111111	13f	$x^8+x^5+x^4+x^3+x^2+x+1$	85
101001101	14d	$x^8+x^6+x^3+x^2+1$	255
101011111	15f	$x^8+x^6+x^4+x^3+x^2+x+1$	255
101100011	163	$x^8+x^6+x^5+x+1$	255
101100101	165	$x^8+x^6+x^5+x^2+1$	255

Binario	Hex	Polinomio	orden
110110001	1b1	$x^8+x^7+x^5+x^4+1$	51
110111101	1bd	$x^8+x^7+x^5+x^4+x^3+x^2+1$	85
111000011	1c3	$x^8+x^7+x^6+x+1$	255
111001111	1cf	$x^8+x^7+x^6+x^3+x^2+x+1$	255
111010111	1d7	$x^8+x^7+x^6+x^4+x^2+x+1$	17
111011101	1dd	$x^8+x^7+x^6+x^4+x^3+x^2+1$	85
111100111	1e7	$x^8+x^7+x^6+x^5+x^2+x+1$	255
111110011	1f3	$x^8+x^7+x^6+x^5+x^4+x+1$	51
111110101	1f5	$x^8+x^7+x^6+x^5+x^4+x^2+1$	255
111111001	1f9	$x^8+x^7+x^6+x^5+x^4+x^3+1$	85

Binario	Hex	Polinomio	orden
101101001	169	$x^8+x^6+x^5+x^3+1$	255
101110001	171	$x^8+x^6+x^5+x^4+1$	255
101110111	177	$x^8+x^6+x^5+x^4+x^2+x+1$	85
101111011	17b	$x^8+x^6+x^5+x^4+x^3+x+1$	85
110000111	187	$x^8+x^7+x^2+x+1$	255
110001011	18b	$x^8+x^7+x^3+x+1$	85
110001101	18d	$x^8+x^7+x^3+x^2+1$	255
110011111	19f	$x^8+x^7+x^4+x^3+x^2+x+1$	51
110100011	1a3	$x^8+x^7+x^5+x+1$	85
110101001	1a9	$x^8+x^7+x^5+x^3+1$	255

Ahora tenemos todos los elementos para poder construir al campo $GF(2^8)$, usado en AES. Antes se explicaran los datos que se dieron en las listas anteriores, particularmente el orden del polinomio irreducible.

Como sabemos el campo $GF(2^8)$ es el cociente $\mathbf{Z}_2[x]/(f)$ donde f es cualquier polinomio de los arriba escritos, en unas hojas más se mostrara que en el caso especial de AES es indiferente que polinomio se elija, sin embargo para ser compatibles AES toma al primer polinomio irreducible, es decir $m(x)=x^8+x^4+x^3+x+1$.

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

Para ejemplificar veamos la construcción campos de menor grado:

La construcción del campo finito de 2^2 elementos, GF(2^2), para esto tomemos el polinomio $f(x)=x^2+x+1$ con coeficientes en $\mathbf{Z}_2$, este polinomio es irreducible, ahora sea α una raíz, entonces $\alpha^2=1+\alpha$, de aquí observamos que:

$$\alpha^0=1$$
$$\alpha^1=\alpha$$
$$\alpha^2=1+\alpha$$
$$\alpha^3=\alpha(1+\alpha)=\alpha+\alpha^2=1$$

Se dice que α genera a un grupo cíclico de orden 2^2-1=3 elementos. Entonces el campo finito GF(2^2), de 2^2 elementos tiene como tablas de suma y multiplicación a

+	0	1	α	$1+\alpha$
0	0	1	α	$1+\alpha$
1	1	0	$1+\alpha$	α
α	α	$1+\alpha$	0	1
$1+\alpha$	$1+\alpha$	α	1	0

¥	1	α	$1+\alpha$
1	1	α	$1+\alpha$
α	α	$1+\alpha$	1
$1+\alpha$	$1+\alpha$	1	α

Cualquier elemento de GF(2^2), es un polinomio de grado menor a 2, y coeficientes en $\mathbf{Z}_2$, de otra forma un elemento de GF(2^2), es un elemento del cociente $\mathbf{Z}_2[x]/(x^2+x+1)$, o más bien un elemento de ahí es una clase de equivalencia que consiste de todos los polinomios que al dividirlos por f(x) su residuo es un polinomio de grado menor a 2, de la misma manera como los elementos de $\mathbf{Z}_p$ son números menores a p.

Para construir ahora el campo GF(2^3), es necesario tomar un polinomio irreducible y una raíz de ese polinomio con su orden según la lista. En este caso los dos polinomios son de orden 7, es decir, la elección de cualquiera de los dos, la raíz nos generará todo el campo. De la misma manera, cualquier polinomio f(x) con coeficientes en $\mathbf{Z}_2$, pertenece a una clase, y se sabe a qué clase pertenece cuando se obtiene el residuo de f(x) con el polinomio irreducible. Observamos también que todos los residuos son todos los polinomios de grado menor a 3, es decir los polinomios de grado a lo más 2.

En este caso tenemos dos polinomios irreducibles hagamos las operaciones que corresponden a cada caso y veamos algunas de sus diferencias.

$m(x)= x^3+x+1$, entonces

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

$m(x)= x^3+x^2+1$, entonces

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

Para el primer caso la tabla de producto queda de la siguiente manera

•	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
x	x^2	x^2+x	x+1	1	x^2+x+1	x^2+1
x+1	x^2+x	x^2+1	x^2+x+1	x^2	1	x
x^2	x+1	x^2+x+1	x^2+x	x	x^2+1	1
x^2+1	1	x^2	x	x^2+x+1	x+1	x^2+x
x^2+x	x^2+x+1	1	x^2+1	x+1	x	x^2
x^2+x+1	x^2+1	x^2+x	1	x^2+x	x^2	x+1

Para el segundo caso tenemos

•	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
x	x^2	x^2+x	x^2+1	x^2+x+1	1	x+1
x+1	x^2+x	x^2+1	1	x	x^2+x+1	x^2
x^2	x^2+1	1	x^2+x+1	x+1	x	x^2+x
x^2+1	x^2+x+1	x	x+1	x^2+x	x^2	1
x^2+x	1	x^2+x+1	x	x^2	x+1	x^2+1
x^2+x+1	x+1	x^2	x^2+x	1	x^2+1	x^2+x

Observamos lo siguiente:

1. Tanto en un caso como en otro los polinomios involucrados sólo cambian de lugar en las dos tablas de multiplicación.
2. Sin embargo hay dos cosa que considerar: la primera en los dos casos el polinomio es irreducible, por lo tanto x es un generador y así el producto se puede realizar con la formula $x^i x^j = x^{(i+j) \bmod 2^n - 1}$, que pueden ser usadas para el producto siempre y cuando el polinomio sea primitivo. Segundo la diferencia esta en el tiempo de obtener el producto ya que depende del polinomio irreducible al obtener el residuo. Esto se nota más en campos grandes, donde la obtención de un producto consiste en hacer la división por los respectivos

polinomios irreducibles tardando más en el caso de que este polinomio sea más grande, en tales casos es preferible buscar polinomios irreducibles con el menor número de términos.

3 Existen otras formas de representar este tipo de campos que no son el propósito de este reporte, por ejemplo usando bases normales.

Hay que tener cuidado solo en el caso donde el polinomio es irreducible y su grado (el dato de la derecha en la tabla de polinomios) no alcanza a generar todo el grupo multiplicativo, en estos casos debemos elegir otro generador, el siguiente teorema dice que siempre podemos encontrar un generador, más aún hay suficientes de estos.

Teorema: Todo campo finito tiene un generador. Si g es un generador de $\mathbf{F}_q^*$, entonces g^i es también un generador si y sólo si $(i, q-1)=1$, es decir hay un total de $\varphi(q-1)$ generadores diferentes.

El caso de $\mathbf{F}_{2^4}$ tenemos un polinomio irreducible no primitivo (1f), es decir, que x no genera al grupo cíclico multiplicativo. Dependiendo del caso que tengamos podemos elegir un polinomio primitivo o no.

Nota importante: para el caso de AES no existe diferencia en seguridad si el polinomio es o no primitivo, más aún, se verá en la sección 8 que no es importante la elección del polinomio.

En los siguientes casos la construcción del campo es similar a los anteriores.

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.

Nota importante: AES 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 $(1+x)^i \bmod m(x)$:

Lista de todos los elementos del campo GF(2^8) usando el polinomio irreducible $m(x) = x^8 + x^4 + x^3 + x + 1$.

1 1+x
2 $1 + x^2$
3 $1 + x + x^2 + x^3$
4 $1 + x^4$
5 $1 + x + x^4 + x^5$
6 $1 + x^2 + x^4 + x^6$
7 $1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$
8 $x + x^3 + x^4$
9 $x + x^2 + x^3 + x^5$
10 $x + x^4 + x^5 + x^6$
11 $x + x^2 + x^4 + x^7$
12 $1 + x^5 + x^7$
13 $x^3 + x^4 + x^5 + x^6 + x^7$
14 $1 + x + x^4$
15 $1 + x^2 + x^4 + x^5$
16 $1 + x + x^2 + x^3 + x^4 + x^6$
17 $1 + x^5 + x^6 + x^7$
18 $x^3 + x^4 + x^5$
19 $x^3 + x^6$
20 $x^3 + x^4 + x^6 + x^7$
21 $1 + x + x^4 + x^5 + x^6$
22 $1 + x^2 + x^4 + x^7$
23 $x^2 + x^5 + x^7$
24 $1 + x + x^2 + x^4 + x^5 + x^6 + x^7$
25 x
26 $x + x^2$
27 $x + x^3$
28 $x + x^2 + x^3 + x^4$
29 $x + x^5$
30 $x + x^2 + x^5 + x^6$
31 $x + x^3 + x^5 + x^7$
32 $1 + x^2 + x^5 + x^6 + x^7$
33 $x^2 + x^4 + x^5$
34 $x^2 + x^3 + x^4 + x^6$
35 $x^2 + x^5 + x^6 + x^7$
36 $1 + x + x^2 + x^4 + x^5$
37 $1 + x^3 + x^4 + x^6$
38 $1 + x + x^3 + x^5 + x^6 + x^7$
39 $x + x^2 + x^5$
40 $x + x^3 + x^5 + x^6$
41 $x + x^2 + x^3 + x^4 + x^5 + x^7$
42 $1 + x^3 + x^4 + x^6 + x^7$
43 $x^4 + x^5 + x^6$
44 $x^4 + x^7$
45 $1 + x + x^3 + x^5 + x^7$
46 $x + x^2 + x^5 + x^6 + x^7$
47 $1 + x^4 + x^5$
48 $1 + x + x^4 + x^6$
49 $1 + x^2 + x^4 + x^5 + x^6 + x^7$
50 x^2
51 $x^2 + x^3$
52 $x^2 + x^4$
53 $x^2 + x^3 + x^4 + x^5$
54 $x^2 + x^6$
55 $x^2 + x^3 + x^6 + x^7$
56 $1 + x + x^2 + x^3 + x^6$
57 $1 + x^4 + x^6 + x^7$
58 $x^3 + x^5 + x^6$
59 $x^3 + x^4 + x^5 + x^7$
60 $1 + x + x^4 + x^6 + x^7$
61 $x + x^2 + x^3 + x^5 + x^6$
62 $x + x^4 + x^5 + x^7$
63 $1 + x^2 + x^3 + x^6 + x^7$
64 $x^2 + x^3 + x^6$
65 $x^2 + x^4 + x^6 + x^7$
66 $1 + x + x^2 + x^5 + x^6$
67 $1 + x^3 + x^5 + x^7$
68 $x^5 + x^6 + x^7$
69 $1 + x + x^3 + x^4 + x^5$
70 $1 + x^2 + x^3 + x^6$
71 $1 + x + x^2 + x^4 + x^6 + x^7$
72 $x + x^5 + x^6$
73 $x + x^2 + x^5 + x^7$
74 $1 + x^4 + x^5 + x^6 + x^7$
75 x^3
76 $x^3 + x^4$
77 $x^3 + x^5$
78 $x^3 + x^4 + x^5 + x^6$
79 $x^3 + x^7$
80 $1 + x + x^7$
81 $x + x^2 + x^3 + x^4 + x^7$
82 $1 + x^3 + x^4 + x^5 + x^7$
83 $x^4 + x^6 + x^7$
84 $1 + x + x^3 + x^5 + x^6$
85 $1 + x^2 + x^3 + x^4 + x^5 + x^7$
86 $x^2 + x^3 + x^4 + x^6 + x^7$
87 $1 + x + x^2 + x^3 + x^4 + x^5 + x^6$
88 $1 + x^7$

89 $x^3 \text{ H } x^4 \text{ H } x^7$
90 $1 \text{ H } x \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
91 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^6 \text{ H } x^7$
92 $1 \text{ H } x^3 \text{ H } x^6$
93 $1 \text{ H } x \text{ H } x^3 \text{ H } x^4 \text{ H } x^6 \text{ H } x^7$
94 $x \text{ H } x^2 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
95 $x \text{ H } x^3 \text{ H } x^4 \text{ H } x^7$
96 $1 \text{ H } x^2 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
97 $x^2 \text{ H } x^6 \text{ H } x^7$
98 $1 \text{ H } x \text{ H } x^2 \text{ H } x^4 \text{ H } x^6$
99 $1 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
100 x^4
101 $x^4 \text{ H } x^5$
102 $x^4 \text{ H } x^6$
103 $x^4 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
104 $1 \text{ H } x \text{ H } x^3$
105 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^4$
106 $1 \text{ H } x \text{ H } x^2 \text{ H } x^5$
107 $1 \text{ H } x^3 \text{ H } x^5 \text{ H } x^6$
108 $1 \text{ H } x \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
109 $x \text{ H } x^2 \text{ H } x^4 \text{ H } x^6 \text{ H } x^7$
110 $1 \text{ H } x^5 \text{ H } x^6$
111 $1 \text{ H } x \text{ H } x^5 \text{ H } x^7$
112 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
113 $1 \text{ H } x^3 \text{ H } x^4$
114 $1 \text{ H } x \text{ H } x^3 \text{ H } x^5$
115 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
116 $1 \text{ H } x \text{ H } x^2 \text{ H } x^7$
117 $x \text{ H } x^4 \text{ H } x^7$
118 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^7$
119 $x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
120 $1 \text{ H } x \text{ H } x^2 \text{ H } x^3 \text{ H } x^5$
121 $1 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
122 $1 \text{ H } x \text{ H } x^4 \text{ H } x^7$
123 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^7$
124 $1 \text{ H } x^3 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
125 x^5
126 $x^5 \text{ H } x^6$
127 $x^5 \text{ H } x^7$
128 $1 \text{ H } x \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
129 $x \text{ H } x^2 \text{ H } x^4$
130 $x \text{ H } x^3 \text{ H } x^4 \text{ H } x^5$
131 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^6$
132 $x \text{ H } x^4 \text{ H } x^6 \text{ H } x^7$
133 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^6$
134 $1 \text{ H } x \text{ H } x^2 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
135 $x \text{ H } x^6 \text{ H } x^7$
136 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^6$
137 $1 \text{ H } x \text{ H } x^2 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
138 $x \text{ H } x^4 \text{ H } x^5$
139 $x \text{ H } x^2 \text{ H } x^4 \text{ H } x^6$
140 $x \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
141 $1 \text{ H } x^2 \text{ H } x^4$
142 $1 \text{ H } x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5$
143 $1 \text{ H } x^6$
144 $1 \text{ H } x \text{ H } x^6 \text{ H } x^7$
145 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^6$
146 $x \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
147 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5$
148 $1 \text{ H } x \text{ H } x^2 \text{ H } x^6$
149 $1 \text{ H } x^3 \text{ H } x^6 \text{ H } x^7$
150 x^6
151 $x^6 \text{ H } x^7$
152 $1 \text{ H } x \text{ H } x^3 \text{ H } x^4 \text{ H } x^6$
153 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
154 $x^2 \text{ H } x^3 \text{ H } x^5$
155 $x^2 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
156 $x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^7$
157 $1 \text{ H } x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
158 $x \text{ H } x^3 \text{ H } x^4 \text{ H } x^6 \text{ H } x^7$
159 $1 \text{ H } x^2 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
160 $1 \text{ H } x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^7$
161 $x \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
162 $1 \text{ H } x^2 \text{ H } x^4 \text{ H } x^6 \text{ H } x^7$
163 $x^2 \text{ H } x^5 \text{ H } x^6$
164 $x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^7$
165 $1 \text{ H } x \text{ H } x^2 \text{ H } x^3 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
166 $x \text{ H } x^3 \text{ H } x^5$
167 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
168 $x \text{ H } x^7$
169 $1 \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^7$
170 $x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
171 $1 \text{ H } x \text{ H } x^2 \text{ H } x^3 \text{ H } x^4 \text{ H } x^6 \text{ H } x^7$
172 $x \text{ H } x^3 \text{ H } x^4 \text{ H } x^5 \text{ H } x^6$
173 $x \text{ H } x^2 \text{ H } x^3 \text{ H } x^7$
174 $1 \text{ H } x^3 \text{ H } x^7$
175 x^7
176 $1 \text{ H } x \text{ H } x^3 \text{ H } x^4 \text{ H } x^7$
177 $x \text{ H } x^2 \text{ H } x^4 \text{ H } x^5 \text{ H } x^7$
178 $1 \text{ H } x^6 \text{ H } x^7$
179 $x^3 \text{ H } x^4 \text{ H } x^6$
180 $x^3 \text{ H } x^5 \text{ H } x^6 \text{ H } x^7$
181 $1 \text{ H } x \text{ H } x^5$
182 $1 \text{ H } x^2 \text{ H } x^5 \text{ H } x^6$

183 $1 + x + x^2 + x^3 + x^5 + x^7$
184 $x + x^3 + x^5 + x^6 + x^7$
185 $1 + x^2 + x^5$
186 $1 + x + x^2 + x^3 + x^5 + x^6$
187 $1 + x^4 + x^5 + x^7$
188 $x^3 + x^6 + x^7$
189 $1 + x + x^6$
190 $1 + x^2 + x^6 + x^7$
191 $x^2 + x^4 + x^6$
192 $x^2 + x^3 + x^4 + x^5 + x^6 + x^7$
193 $1 + x + x^2 + x^3 + x^4$
194 $1 + x^5$
195 $1 + x + x^5 + x^6$
196 $1 + x^2 + x^5 + x^7$
197 $x^2 + x^4 + x^5 + x^6 + x^7$
198 $1 + x + x^2$
199 $1 + x^3$
200 $1 + x + x^3 + x^4$
201 $1 + x^2 + x^3 + x^5$
202 $1 + x + x^2 + x^4 + x^5 + x^6$
203 $1 + x^3 + x^4 + x^7$
204 $x^4 + x^5 + x^7$
205 $1 + x + x^3 + x^6 + x^7$
206 $x + x^2 + x^6$
207 $x + x^3 + x^6 + x^7$
208 $1 + x^2 + x^6$
209 $1 + x + x^2 + x^3 + x^6 + x^7$
210 $x + x^3 + x^6$
211 $x + x^2 + x^3 + x^4 + x^6 + x^7$
212 $1 + x^3 + x^4 + x^5 + x^6$
213 $1 + x + x^3 + x^7$
214 $x + x^2 + x^7$
215 $1 + x^4 + x^7$
216 $x^3 + x^5 + x^7$
217 $1 + x + x^5 + x^6 + x^7$
218 $x + x^2 + x^3 + x^4 + x^5$
219 $x + x^6$
220 $x + x^2 + x^6 + x^7$
221 $1 + x^4 + x^6$
222 $1 + x + x^4 + x^5 + x^6 + x^7$
223 $x + x^2 + x^3$
224 $x + x^4$
225 $x + x^2 + x^4 + x^5$
226 $x + x^3 + x^4 + x^6$
227 $x + x^2 + x^3 + x^5 + x^6 + x^7$
228 $1 + x^3 + x^5$
229 $1 + x + x^3 + x^4 + x^5 + x^6$
230 $1 + x^2 + x^3 + x^7$
231 $x^2 + x^3 + x^7$
232 $1 + x + x^2 + x^3 + x^7$
233 $x + x^3 + x^7$
234 $1 + x^2 + x^7$
235 $x^2 + x^4 + x^7$
236 $1 + x + x^2 + x^5 + x^7$
237 $x + x^4 + x^5 + x^6 + x^7$
238 $1 + x^2 + x^3$
239 $1 + x + x^2 + x^4$
240 $1 + x^3 + x^4 + x^5$
241 $1 + x + x^3 + x^6$
242 $1 + x^2 + x^3 + x^4 + x^6 + x^7$
243 $x^2 + x^3 + x^4 + x^5 + x^6$
244 $x^2 + x^7$
245 $1 + x + x^2 + x^4 + x^7$
246 $x + x^5 + x^7$
247 $1 + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$
248 $x^2 + x^3 + x^4$
249 $x^2 + x^5$
250 $x^2 + x^3 + x^5 + x^6$
251 $x^2 + x^4 + x^5 + x^7$
252 $1 + x + x^2 + x^6 + x^7$
253 $x + x^4 + x^6$
254 $x + x^2 + x^4 + x^5 + x^6 + x^7$
255 1
256 $1+x$

Para terminar con 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 $ab=AntiLog((Log[a]+Log[b]) \bmod 255)$, próximamente veremos en el código que usa esta fórmula para calcular los productos.

2.1 El anillo $GF(2^8)[x]/(x^4+1)$.

Otra 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 módulo (x^4+1) para obtener un polinomio de grado 3. El objetivo de lo anterior es poder definir multiplicación entre 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 no tenemos problema.

Sea a(x) un polinomio tal que $a_0+a_1x+a_2x^2+a_3x^3 \in Gf(2^8)[x]/(x^4+1)$, donde $a_i \in GF(2^8)$, y b(x) otro polinomio igual, entonces a(x)b(x)=c(x) donde c(x) tiene la misma forma, particularmente:

$$\begin{aligned} a(x)b(x) &= \left(a_0+a_1x+a_2x^2+a_3x^3\right)\left(b_0+b_1x+b_2x^2+b_3x^3\right) \\ &= a_0b_0+(a_1b_0+a_0b_1)x+(a_2b_0+a_1b_1+a_0b_2)x^2 \\ &\quad + (a_3b_0+a_2b_1+a_1b_2+a_0b_3)x^3 \\ &\quad + (a_3b_1+a_2b_2+a_1b_3)x^4 \\ &\quad +(a_3b_2+a_2b_3)x^5+ a_3b_3x^6 \end{aligned}$$

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

$$\begin{aligned} a(x)b(x)=&(a_0b_0+a_3b_1+a_2b_2+a_1b_3) \\ &+(a_1b_0+a_0b_1+a_3b_2+a_2b_3)x \\ &+(a_2b_0+a_1b_1+a_0b_2+a_3b_3)x^2 \\ &+(a_3b_0+a_2b_1+a_1b_2+a_0b_3)x^3 \\ =&\ c_0+c_1x+c_2x^2+c_3x^3 \end{aligned}$$

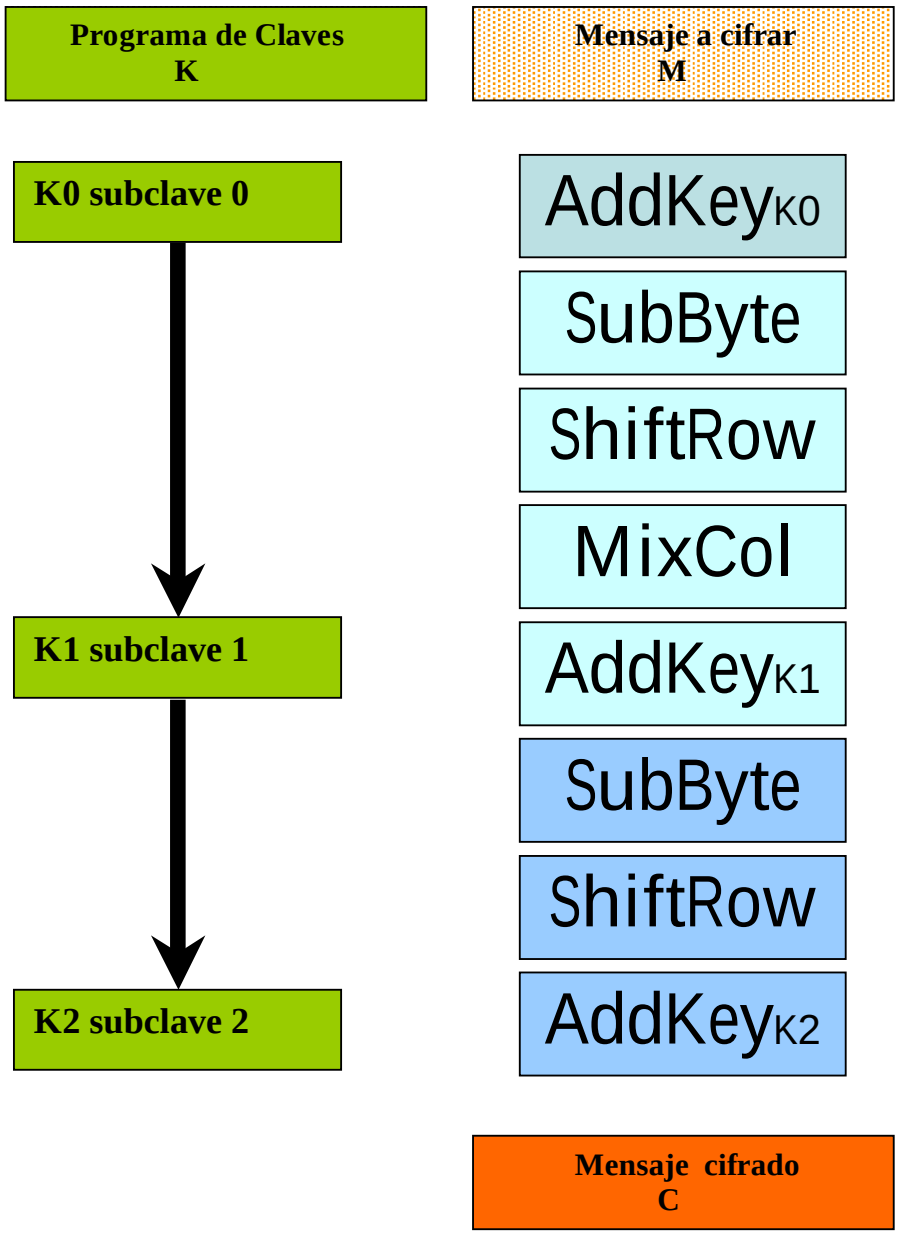
Finalmente de manera matricial el anterior producto puede ser visto como:

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

3.- Baby AES.

Como una primera vista al algoritmo AES veamos esta versión simplificada que llamamos Baby-AES, tiene la misma estructura que AES, y nos servirá para poder entender enseguida con mayor facilidad la descripción completa de AES.
El algoritmo Baby-AES opera sobre un texto de 16 bits, y genera un texto cifrado de 16 bits, con una clave de 16 bits. Baby-AES consiste en dos procedimientos, el de cifrado dónde se aplican 4 funciones básicas tantas veces como se desee y el proceso de la derivación de las subclaves llamado programa de claves, la idea de hacer una versión simplificada ha sido usada como un primer vistazo al algoritmo en [66], pero también una manera de criptoanálizarlo, es decir, una intentar un ataque en la versión simplificada para después extenderlo a la versión completa [116].

De manera gráfica Baby-AES puede ilustrarse de la siguiente manera:



Se ve que el algoritmo Baby-AES consiste de 8 aplicaciones de funciones, de las cuales son realmente 4, SubByte, ShiftRow, MixCol, y AddKey, dónde se aplican casi dos veces, eliminando en la segunda ronda a MixCol, y agregando una aplicación más de AddKey antes de la primera ronda, con la subclave K0. Esto se justifica en el proceso de descifrado.

Entonces la descripción de Baby-AES consiste en la descripción de las 4 funciones básicas, con el programa de claves que nos generara K0, K1, K2 a partir de la clave inicial K.

Todas las funciones se aplican como ya se dijo a un plaintexto de 16 bits que puede ser visto como un arreglo de 4 medios-bytes, donde cada medio-byte consiste de 4 bits cada uno.

La variable a la que se aplican las anteriores funciones es la entrada de 16 bits ($b_0b_1b_2b_3b_4b_5b_6b_7b_8b_9b_{10}b_{11}b_{12}b_{13}b_{14}b_{15}$), que estará dividida en 4 partes $P_0P_1P_2P_3$ de 4 bits cada una, conocida como estado P y toma la siguiente forma matricial:

$$\begin{array}{|c|c|}\hline P_0 & P_2 \\ \hline P_1 & P_3 \\ \hline \end{array} = \begin{array}{|c|c|}\hline b_0b_1b_2b_3 & b_8b_9b_{10}b_{11} \\ \hline b_4b_5b_6b_7 & b_{12}b_{13}b_{14}b_{15} \\ \hline \end{array}$$

Donde cada conjunto de 4 bits (½ byte) se le asigna el polinomio $b_0 + b_1x + b_2x^2 + b_3x^3$ elemento de $GF(16) = GF(2)[x]/(x^4 + x + 1)$.

De la misma manera la clave inicial K consiste de 16 bits y se representa por la matriz siguiente:

$$\begin{array}{|c|c|}\hline k_0 & k_2 \\ \hline k_1 & k_3 \\ \hline \end{array}$$

SubByte

Esta función reemplaza cada ½-byte de 4 bits, por otro ½-byte según la S-box de la siguiente manera:

La S-Box se obtiene por dos etapas, la primera es considerando cada ½-byte como elemento del campo finito $GF(16) = GF(2)[x]/(x^4 + x + 1)$, es decir un campo de 16 elementos. Entonces se le asigna a cada elemento su inverso multiplicativo (al 0000 se le asigna el mismo 0000).

El segundo paso es aplicar un mapeo lineal que tiene la siguiente forma:

Si tenemos como salida de la inversión al ½-byte $b_0b_1b_2b_3$ se le asigna el polinomio $N(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, y considere los polinomios constantes

$$a(x) = x^3 + x^2 + 1, \text{ y } b(x) = x^3 + 1 \text{ en } GF(2)[x]/(x^4 + 1)$$

Entonces el segundo paso es:

$$N(x) \sim a(x)N(x) + b(x) \text{ modulo } x^4 + 1$$

Los resultados de estos dos pasos los tenemos en la siguiente tabla:

x^i	$x^i \bmod (x^4 + x + 1)$	*bits*	$\left(x^i\right)^{-1}$	$((x^3 + x^2 + 1)\left(x^i\right)^{-1} + (x^3 + 1)) \bmod (x^4 + 1)$
x	x	0010	1001	1010
x^2	x^2	0100	1101	1101
x^3	x^3	1000	1111	0110
x^4	$x+1$	0011	1110	1011
x^5	$x^2 + x$	0110	0111	1000
x^6	$x^3 + x^2$	1100	1010	1100
x^7	$x^3 + x + 1$	1011	0101	0011
x^8	$x^2 + 1$	0101	1011	0001
x^9	$x^3 + x$	1010	1000	0111
x^{10}	$x^2 + x + 1$	0111	0110	0101
x^{11}	$x^3 + x^2 + x$	1110	0011	1111
x^{12}	$x^3 + x^2 + x + 1$	1111	1000	0111
x^{13}	$x^3 + x^2 + 1$	1101	0100	1110
x^{14}	$x^3 + 1$	1001	0010	0010
x^{15}	1	0001	0001	0100

Obviamente el elemento 0000 tiene como correspondencia el elemento 1001.

Si queremos ver a la aplicación $N(x) \sim a(x)N(x)+b(x)$ modulo x^4+1 de manera matricial, veamos primero la representación del producto de dos polinomios

$$\begin{aligned} a(x)n(x) &= \left(a_0 + a_1x + a_2x^2 + a_3x^3\right)\left(n_0 + n_1x + n_2x^2 + n_3x^3\right) \\ &= a_0n_0 + (a_1n_0 + a_0n_1)x + (a_2n_0 + a_1n_1 + a_0n_2)x^2 \\ &\quad + (a_3n_0 + a_2n_1 + a_1n_2 + a_0n_3)x^3 \\ &\quad + (a_3n_1 + a_2n_2 + a_1n_3)x^4 \\ &\quad + (a_3n_2 + a_2n_3)x^5 + a_3n_3x^6 \end{aligned}$$

Si ahora aplicamos el modulo x^4+1

$$\begin{aligned} a(x)n(x) &= (a_0n_0 + a_3n_1 + a_2n_2 + a_1n_3) \\ &\quad + (a_1n_0 + a_0n_1 + a_3n_2 + a_2n_3)x \\ &\quad + (a_2n_0 + a_1n_1 + a_0n_2 + a_3n_3)x^2 \\ &\quad + (a_3n_0 + a_2n_1 + a_1n_2 + a_0n_3)x^3 \\ &= d_0 + d_1x + d_2x^2 + d_3x^3 \end{aligned}$$

Que en forma matricial queda como

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix}$$

Entonces la aplicación $L(x) = a(x)N(x)+c(x)$ modulo x^4+1 la podemos representar como:

$$\begin{bmatrix} l_0 \\ l_1 \\ l_2 \\ l_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ n_2 \\ n_3 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Por ejemplo $0101 \sim 0011$

$$\begin{matrix} 0011 & x^3+x^2+1 & & 0101 & x^3+1 \end{matrix}$$
$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Otro ejemplo $0011 \sim 1111$

$$\begin{matrix} 1111 & x^3+x^2+1 & & 0011 & x^3+1 \end{matrix}$$
$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

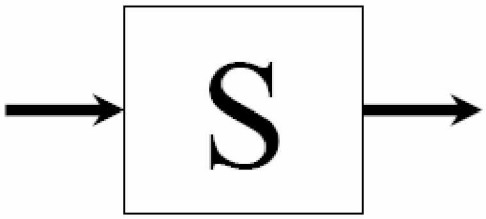
En términos de aplicación a la variable estado de componentes ½-bytes la función SubBytes se ve como:

P_0	P_2	$S(P_0)$	$S(P_2)$
P_1	P_3	$S(P_1)$	$S(P_3)$

ó

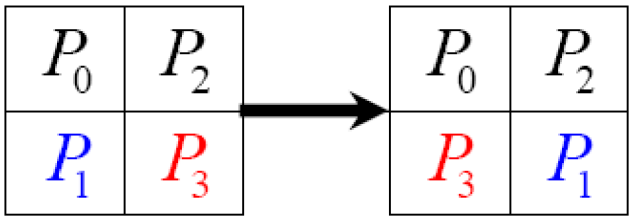
$b_0b_1b_2b_3$	$b_8b_9b_{10}b_{11}$	$S-box(b_0b_1b_2b_3)$	$S-box(b_8b_9b_{10}b_{11})$
$b_4b_5b_6b_7$	$b_{12}b_{13}b_{14}b_{15}$	$S-box(b_4b_5b_6b_7)$	$S-box(b_{12}b_{13}b_{14}b_{15})$

o resumiendo como:

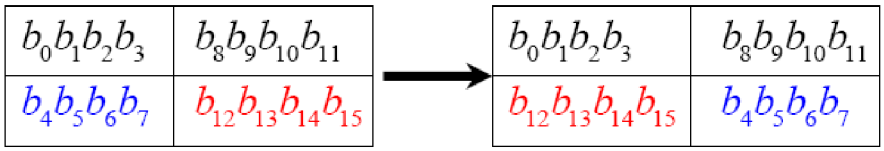


ShiftRow

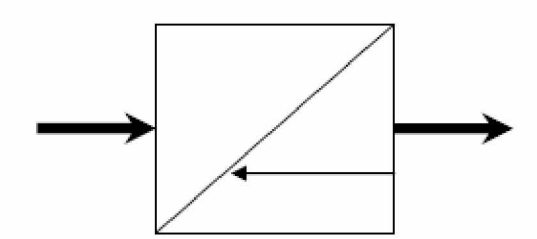
Esta aplicación es muy simple y simplemente aplica un shift a la segunda fila de la matriz estado, de la siguiente manera:



o en representación de bits:



Y que resumimos gráficamente como:



MixCol

La función MixCol se aplica a cada columna, las columnas se ven como un elemento de $GF(2^4)[x]/(x^2+1)$, es decir como polinomio de grado 1 con coeficientes en el campo GF(2⁴)[x] (polinomios de grado 3). Es decir son elementos de la forma $(a_0+a_1x)\in GF(2^4)[x]/(x^2+1)$ donde $a_1, a_2 \in GF(2^4)$.

La función MisCol multiplica a cada columna del estado por un polinomio constante c(x)=3+2x de $GF(2^4)[x]/(x^2+1)$. Para poder ver más explícitamente la aplicación de MixCol primero veamos que forma tiene el producto de dos polinomios en $GF(2^4)[x]/(x^2+1)$.

Si $(a_0+a_1x),\ (b_0+b_1x)$ son dos polinomios, entonces

$$(a_0+a_1x)\cdot(b_0+b_1x)=a_0b_0+(a_0b_1+a_1b_0)x+(a_1b_1)x^2$$

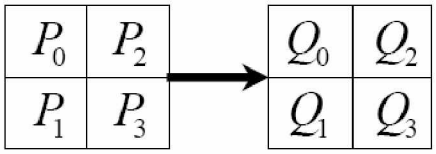
como $x^2+1=0,$ tenemos que:

$$(a_0+a_1x)\cdot(b_0+b_1x)=(a_0b_0+a_1b_1)+(a_0b_1+a_1b_0)x$$
$$=c_0+c_1x$$

Que en forma matricial se ve como:

$$\begin{bmatrix}c_0\\c_1\end{bmatrix}=\begin{bmatrix}a_0 & a_1\\a_1 & a_0\end{bmatrix}\begin{bmatrix}b_0\\b_1\end{bmatrix}$$

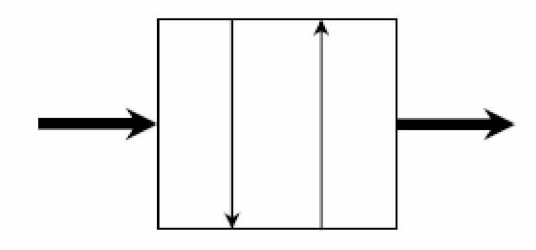
Entonces la función MixCol queda como



Que para cada columna y en forma matricial significa

$$\begin{bmatrix}Q_0\\Q_1\end{bmatrix}=\begin{bmatrix}0011 & 0010\\0010 & 0011\end{bmatrix}\begin{bmatrix}P_0\\P_1\end{bmatrix}$$
$$\begin{bmatrix}Q_2\\Q_3\end{bmatrix}=\begin{bmatrix}0011 & 0010\\0010 & 0011\end{bmatrix}\begin{bmatrix}P_2\\P_3\end{bmatrix}$$

Donde 0011=3, y 0010=2 elementos del campo GF(16).
Finalmente lo resumimos como:

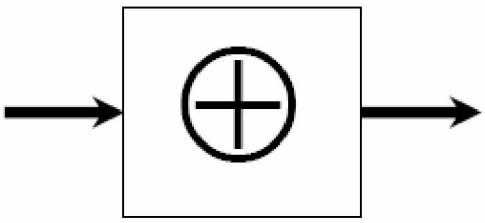


AddKey

La función AddKey simplemente efectúa un Xor entrada con entrada del estado con la clave correspondiente.

$$\begin{array}{|c|c|}\hline P_0 & P_2 \\ \hline P_1 & P_3 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline K_0 & K_2 \\ \hline K_1 & K_3 \\ \hline \end{array} = \begin{array}{|c|c|}\hline P_0 \oplus K_0 & P_2 \oplus K_2 \\ \hline P_1 \oplus K_1 & P_3 \oplus K_3 \\ \hline \end{array}$$

Que lo simplificamos como

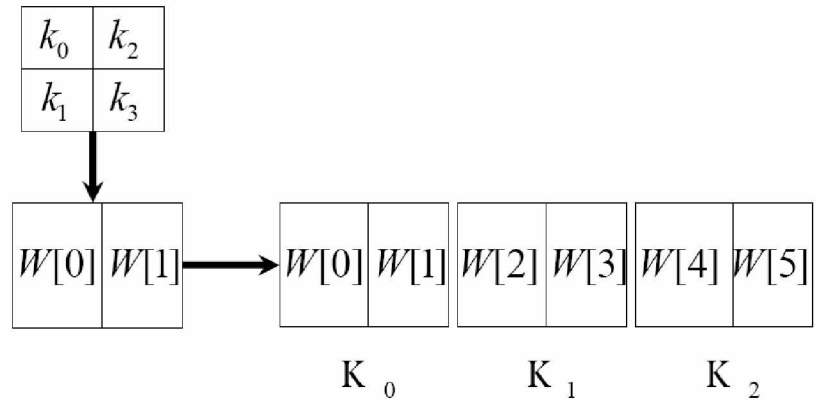


Programa de Claves

En esta sección debemos de mostrar como se obtienen las claves K_0, K_1, y K_2 a partir de K.

El programa de claves consiste en obtener el arreglo W[], de 6 bytes, donde W[0],W[1] son los dos bytes de la clave $K=K_0$, para obtener K_1, y K_2 se

extiende el arreglo W a los bytes W[2],W[3],W[4], y W[5]. Donde K_1=W[2]W[3], y K_2=W[4]W[5].



Por lo tanto es suficiente saber como se construyen las extensiones de W[2],W[3],W[4],W[5] a partir de W[0],W[1].

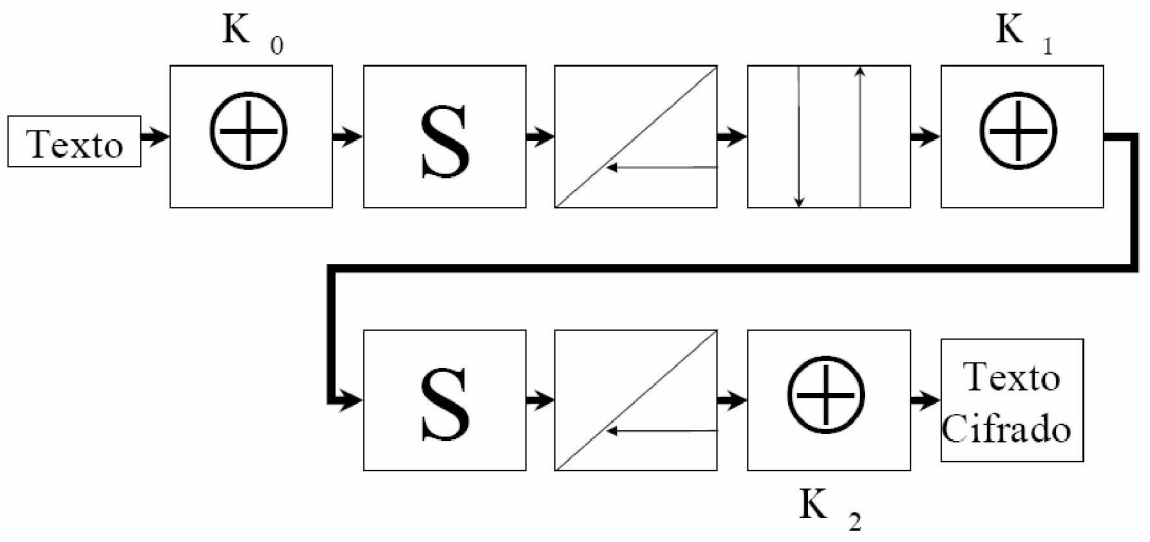
Para la extensión de W primero definamos los siguientes elementos:

Sea $RC[i] = x^{i+2} \in GF(2^4)$, N_0N_1 es la concatenación de dos ½-bytes, entonces sea $RCON[i] = RC[i]0000$ es un byte, y $Rot(N_0N_1) = N_1N_0$, así como $Sub(N_0N_1) = S-Box(N_0)S-Box(N_1)$.

Finalmente como todos son bytes definimos las extensiones de la clave de la siguiente manera:

$$W[0] = K \oplus (11110000)$$
$$W[1] = K \oplus (00001111)$$
$$W[2] = W[0] \oplus RCON[1] \oplus Sub(Rot(W[1]))$$
$$W[3] = W[1] \oplus W[2]$$
$$W[4] = W[2] \oplus RCON[2] \oplus Sub(Rot(W[3]))$$
$$W[5] = W[3] \oplus W[4]$$

Finalmente la descripción final gráfica de Baby-AES queda como sigue:



4.- Descripción de AES.

En esta sección nos dedicamos a dar la descripción detallada de AES, podemos adelantar que con los elementos que se cuentan hasta ahora será muy sencilla de descripción. Primero recordemos que AES trabaja con bloques de datos de 128 bits y longitudes de claves de 128, 192, 256 bit según el FIPS 197[34]. Además AES tiene 10, 12 o 14 vueltas respectivamente, cada vuelta de AES consiste en la aplicación de una ronda estándar, que consiste de 4 transformaciones básicas, la última ronda es especial y consiste de 3 operaciones básicas, añadiendo siempre una ronda inicial. Por otro lado tenemos el programa de claves o extensión de la clave. Por lo anterior nuestra descripción consiste en describir las transformaciones básicas AddRoundKey, SubByte, ShiftRows, MixColumns, y por último de Key Schedule.

Recordemos que AES interpreta a el bloque de entrada de 128 bits, como una matriz 4x4 de entradas de bytes, si el bloque es de 192 bits se agregan 2 columnas más, si lo es de 256 se agregan 4 columnas más.

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

esta asociada al bloque de 128 bits.

$$a_{00}a_{10}a_{20}a_{30}a_{01}a_{11}a_{21}a_{31}a_{02}a_{12}a_{22}a_{32}a_{03}a_{13}a_{23}a_{33}$$

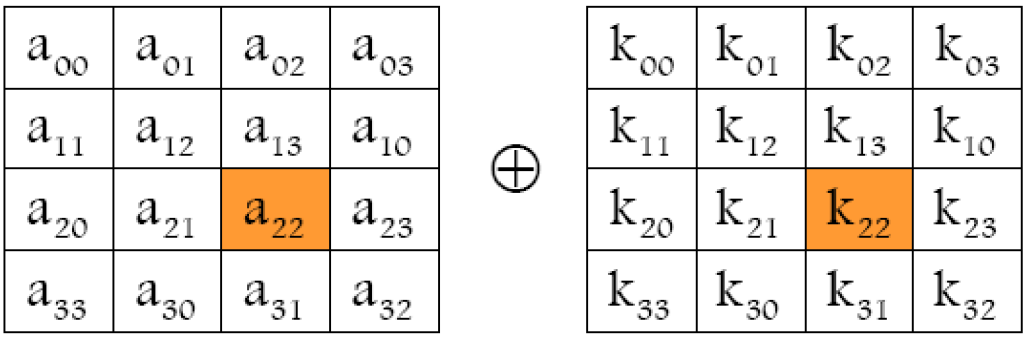
es decir, tomando los primeros 4 bytes conforman la primera columna, los segundos 4 bytes son la segunda columna, etc.

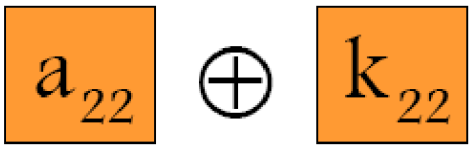
La regla se aplica a los bloques de 192 bits y 256 bits, obteniendo matrices de 6, y 8 columnas respectivamente.

La matriz es la entrada del algoritmo AES, y va cambiando en cada una de las rondas, la denotaremos como $[a_{ij}]$, y la llamaremos matriz estado.

AddRoundKey

Esta transformación toma una matriz y simplemente hace un xor byte a byte con la correspondiente matriz de claves dependiendo de la ronda en que nos encontremos.



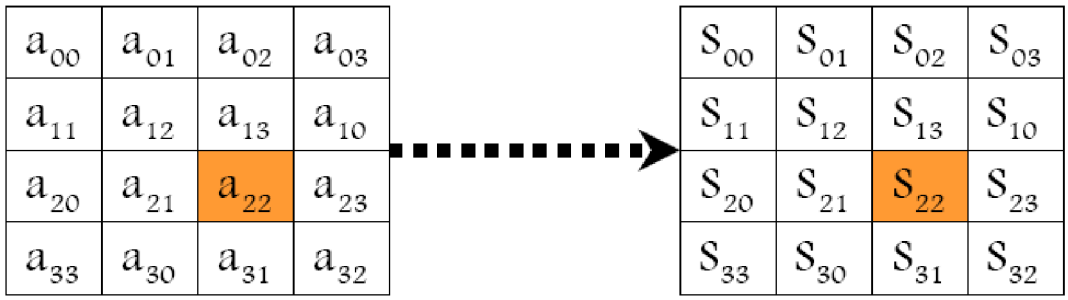


Toma la matriz $[a_{ij}]$ y $[k_{ij}]$ y da como resultado la matriz $[a_{ij} \oplus k_{ij}]$ es decir xorea entrada por entrada, la matriz del bloque con la matriz de la subclave correspondiente.

La matriz estado tiene 4 columnas (FIPS 197) y toma de la extensión de la clave también 4 columnas. El programa de claves genera las necesarias matices de claves para todas las rondas.

ByteSub

En este caso a cada elemento de la matriz estado (byte) se le substituye por otro byte, que depende del primero.



Se substituye la matriz $[a_{ij}]$ por $[s_{ij}]$, donde s_{ij} es el resultado de aplicar dos funciones a a_{ij}, 1) su inverso multiplicativo $a_{ij} \sim a_{ij}^{-1} \in GF(2^8)$, 2) posteriormente una transformación lineal.

1) Todo byte, conjunto de 8 bits, puede verse como un elemento del campo finito $GF(2^8)$, del que ya se hablo bastante, como todo elemento tiene inverso multiplicativo, entonces esta primera función de bytesub, asocia el inverso multiplicativo en $GF(2^8)$, es decir $a_{ij} \sim a_{ij}^{-1} \in GF(2^8)$, al elemento cero se le asocia el mismo cero.

En la siguiente tabla mostramos todos los inversos multiplicativos en representación hexadecimal, por ejemplo el inverso de 88 es 9b. Por definición al 00 le asociamos el mismo 00.

	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	4f	29	c0	b0	e1	e5	c7
1	74	b4	aa	4b	99	2b	60	5f	58	3f	fd	cc	ff	40	ee	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

2) Ahora la transformación lineal que sigue al inverso multiplicativo se aplica bit por bit y sigue la siguiente regla

$$\text{Transformacion lineal en } GF(2^8) \rightarrow GF(2^8)$$

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

$$0 \le i<8,\ c = 01100011 = 63_x$$

En términos de bits queda como

$$\begin{aligned}
b_0' &= b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus c_0 \\
b_1' &= b_1 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_0 \oplus c_1 \\
b_2' &= b_2 \oplus b_6 \oplus b_7 \oplus b_0 \oplus b_1 \oplus c_2 \\
b_3' &= b_3 \oplus b_7 \oplus b_0 \oplus b_1 \oplus b_2 \oplus c_3 \\
b_4' &= b_4 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus c_4 \\
b_5' &= b_5 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus c_5 \\
b_6' &= b_6 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus c_6 \\
b_7' &= b_7 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus c_7
\end{aligned}$$

o en representación matricial

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

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

Por ejemplo la aplicación de ByteSub de 7b sigue los siguientes pasos

$$7b = 01111011 = x^6 + x^5 + x^4 + x^3 + x + 1$$

$$7b^{-1} = x^2 + x = 00000110 = 06$$

que se obtiene de la tabla de inversos en la intersección de la fila 7 y la columna b.

Aplicando ahora la transformación lineal

$$\begin{bmatrix}1\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}=\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\1\\1\\0\\0\\0\\0\\0\end{bmatrix}+\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}$$

Finalmente tenemos

0010 0001 = 21

Que puede obtenerse de la tabla directamente en la intersección de la fila 7 y la columna b.

Otro ejemplo a7

$a7 = 1010\ 0111 = x^7 + x^5 + x^2 + x + 1$

$a7^{-1} = x^6 + x^3 = 0100\ 1000 = 48$

Se tiene de la tabla de inversos, en la intersección de la fila a y la columna 7, y la transformación lineal

$$\begin{bmatrix}0\\0\\1\\1\\1\\0\\1\\0\end{bmatrix}=\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\\0\\1\\0\end{bmatrix}+\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}$$

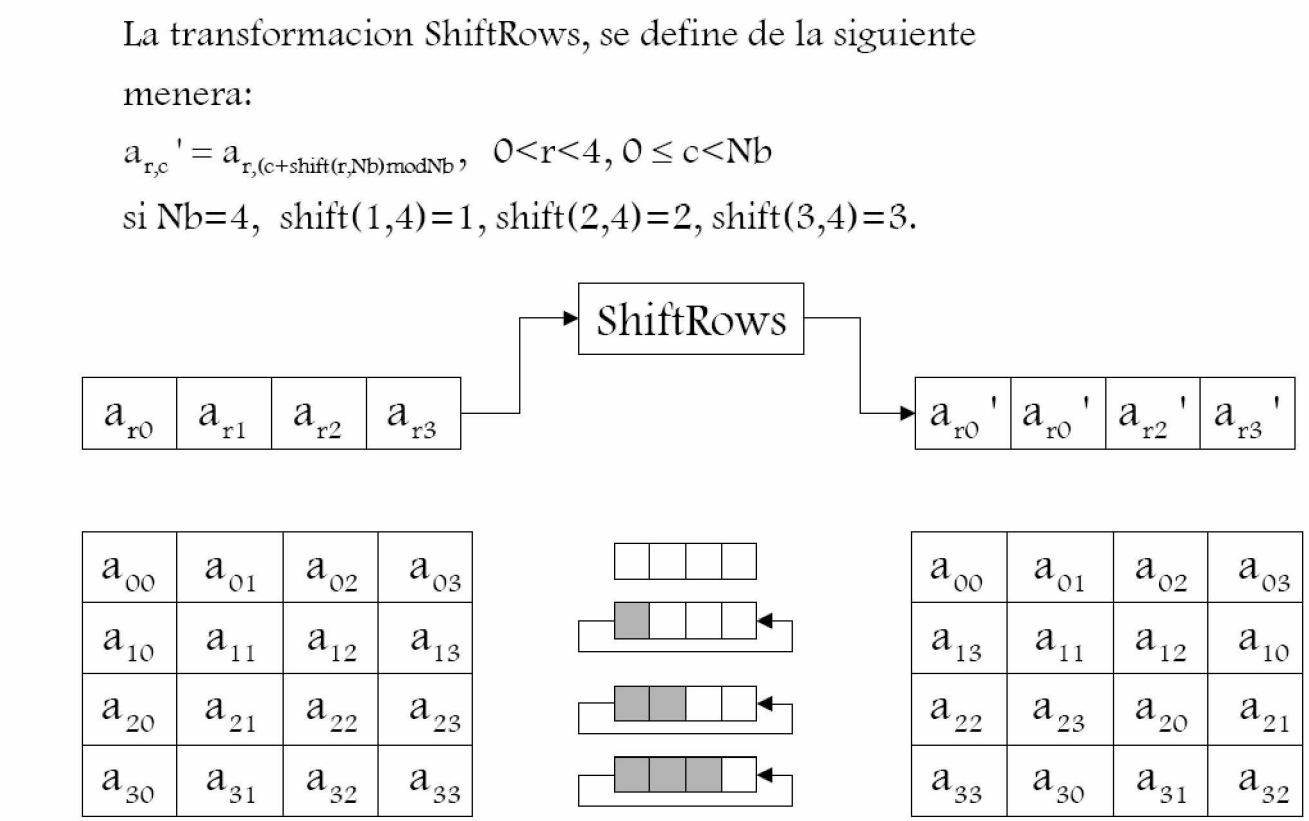
Finalmente tenemos que

01011100 = 5c

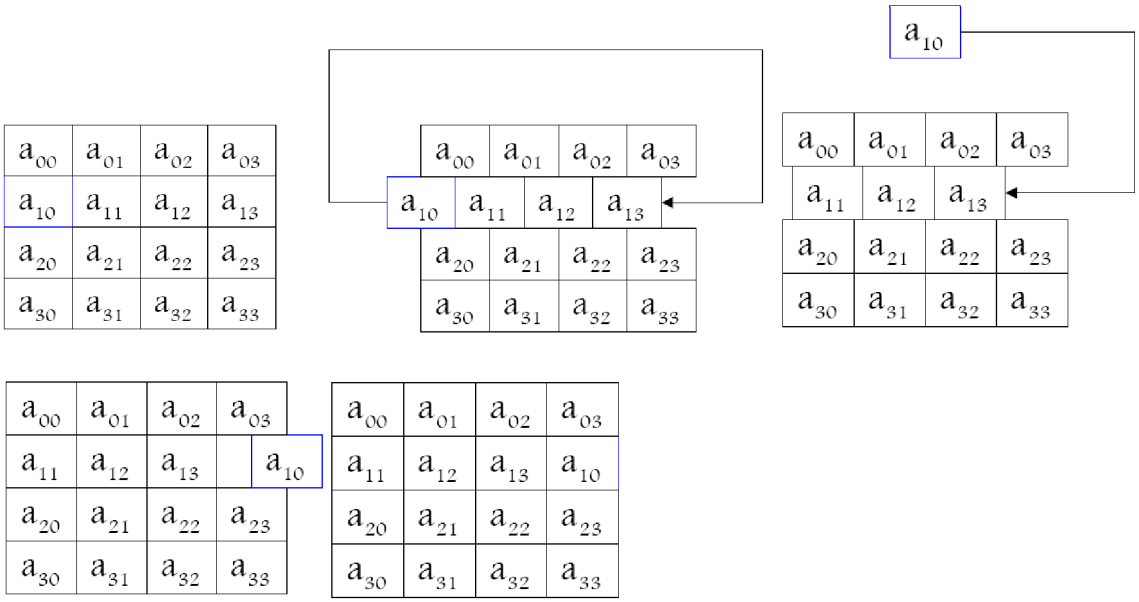
Que es la fila a y columna 7 de S-Box.

ShiftRow

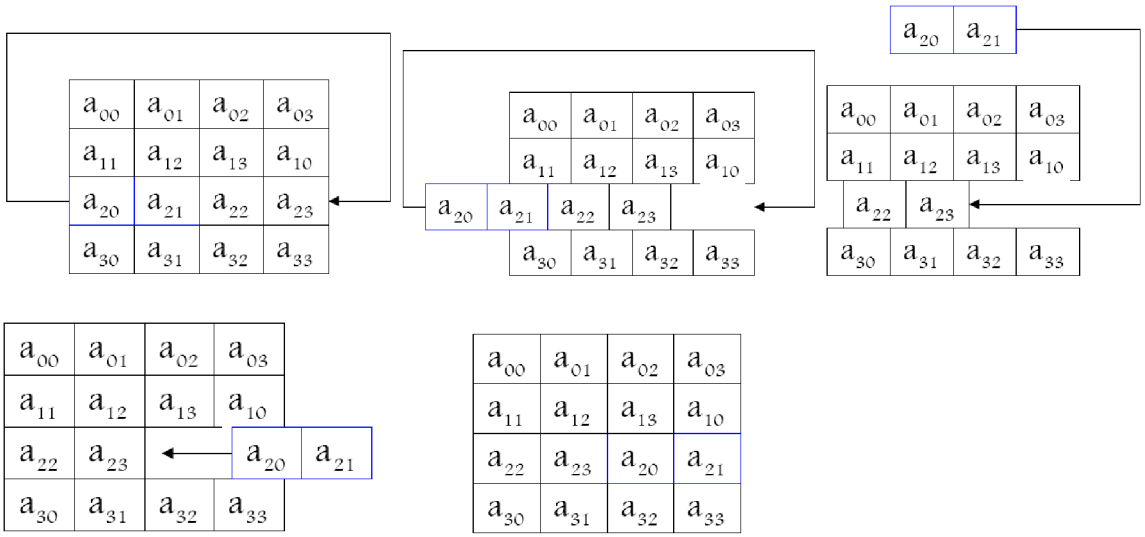
La transformación ShiftRow se aplica a la matriz estado, aplicando corrimientos a sus columnas. Sea aplica a la matriz $\left[a_{ij}\right]$, shifts (corrimientos izquierdos circulares de bytes) a las renglones, de la seguiente manera, recorre 0 bytes a el primer renglón, 1 byte al segundo renglón, 2 bytes al tercer renglón, y 3 bytes recorridos al cuarto renglón.



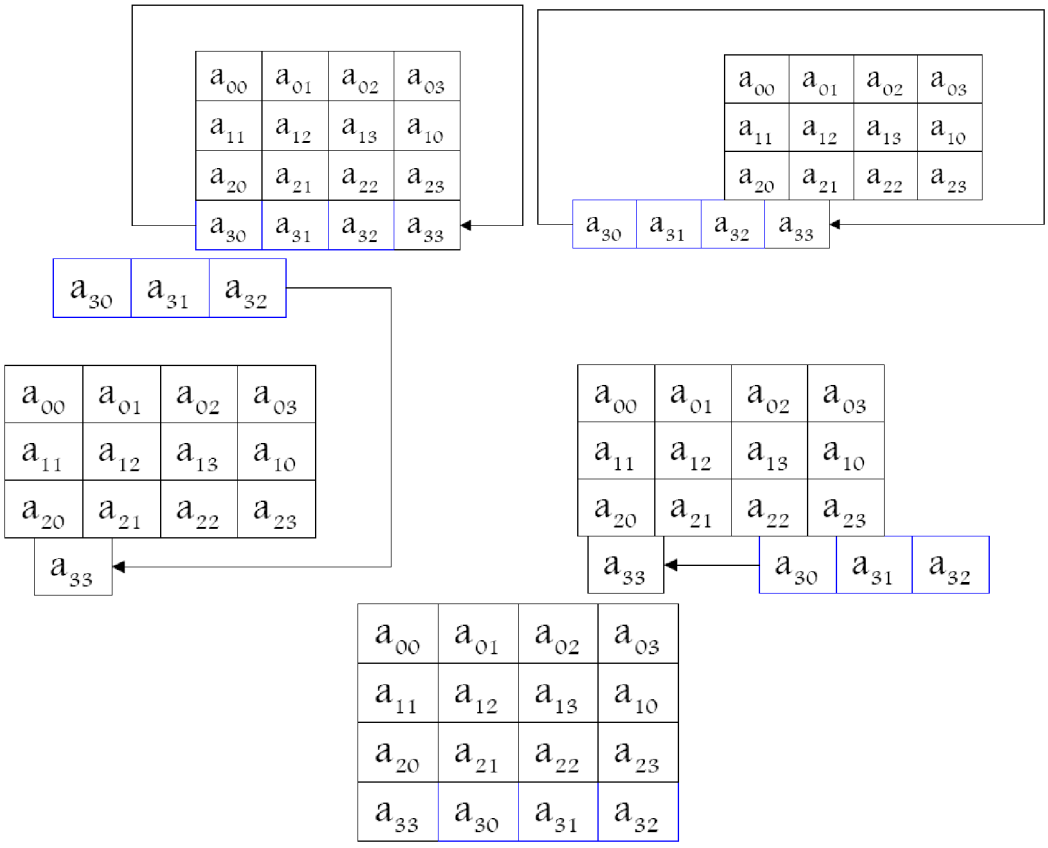
La primera fila queda igual. La segunda fila se recorre un byte, como las figuras siguientes lo muestran.



La tercera columna mueve dos bytes circularmente.

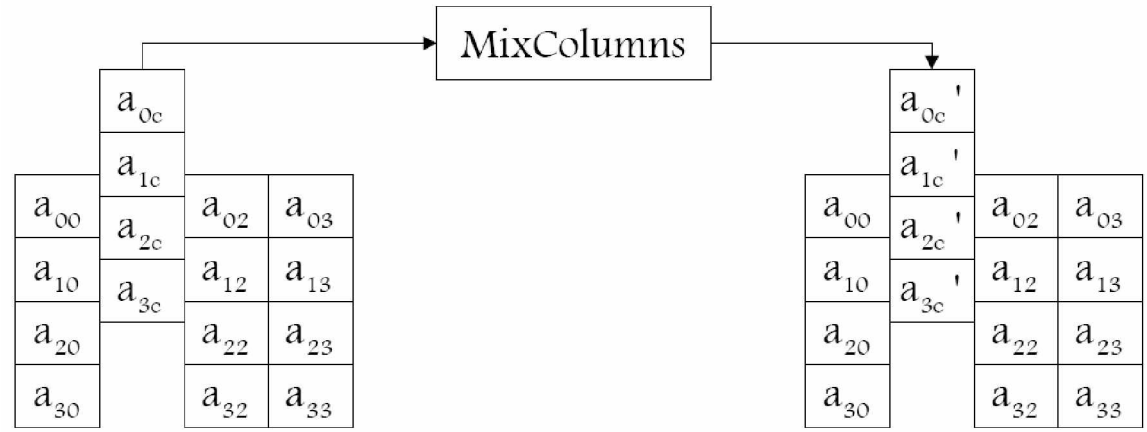


Finalmente la cuarta fila recorre 3 bytes circularmente.



MixColumns

MixColumns: a cada columna de la matriz $[a_{ij}]$ la multiplica por una columna constante en $GF(2^8)[x]/(x^4+1)$.



MixColumns toma cada columna A, y la manda a otra columna A', que se obtiene al multiplicar A por un polinomio constante $c(x) \in GF(2^8)[x]/(x^4+1)$, $c(x) = 03\ x^3 + 01\ x^2 + 01\ x + 02$, entonces $A' = A \cdot c(x)$, como se vio en los antecedentes, este producto se puede representar por la siguiente matriz.

$$\begin{bmatrix} a_0' \\ a_1' \\ a_2' \\ a_3' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Key Schedule.

Aunque Rijndael esta diseñado para manejar muchos casos de longitudes de claves y de bloques, finalmente AES definido en el estándar determina solo permitir los casos de bloques de 128 bits, y longitudes de claves de 128, 192 y 256. Por otra parte la longitud de 128 bits, garantiza la seguridad del sistema hasta después del año 2030, por lo que en este reporte consideraremos el caso solo de claves 128 bits, aunque los algoritmos pueden ser extendidos fácilmente a los casos restantes.

El algoritmo de extensión de la clave es el siguiente:

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

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}	.....

```
KeyExpansion(byte Key[4*Nk] word[Nb*(Nr+1)])
{
  For (i=0; i<Nk; i++)
       W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
  For (i=Nk; i<Nb*(Nr+1); i++)
     {  temp =W[i-1];
        if(i%Nk == 0)  temp=SubByte(RotByte(temp))^Rcon[i/Nk];
        W[i]=W[i-Nk]^temp;
     }
```

En nuestro caso Nb=Nk=4, y Nr=10

La parte verde significa que

```
For (i=0; i<4; i++)
    W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
W[0] = (key[0],key[1],key[2],key[3]);
W[1] = (key[4],key[5],key[6],key[7]);
W[2] = (key[8],key[9],key[10],key[11]);
W[3] = (key[12],key[13],key[14],key[15]);
```

key[0]	key[4]	key[8]	key[12]
key[1]	key[5]	key[9]	key[13]
key[2]	key[6]	key[10]	key[14]
key[3]	key[7]	key[11]	key[15]

Es decir las primeras 4 columnas de la extensión son la clave original.

Tomaremos como ejemplo el descrito en el FIPS 197 como vector de prueba para el caso 128.

Cipher Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

for $Nk = 4$, which results in

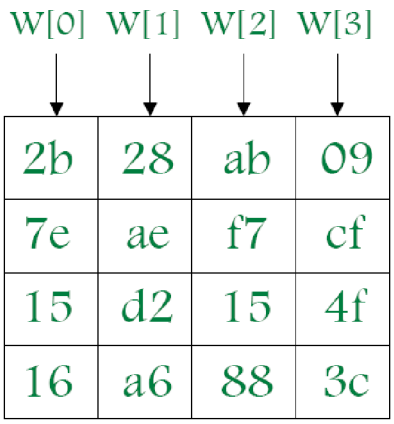
$w0$ = **2b7e1516** $w1$ = **28aed2a6** $w2$ = **abf71588** $w3$ = **09cf4f3c**

Quedando como

```
unsigned char  clave[4][4]={0x2b,0x28,0xab,0x09,
                           0x7e,0xae,0xf7,0xcf,
                           0x15,0xd2,0x15,0x4f,
                           0x16,0xa6,0x88,0x3c};
```

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

o



Ahora veremos con detalle lo que describe la parte de rojo. Para nuestro caso, de bloques de 4 columanas, claves de 4 columans, y por lo tanto 10 rondas.

```
KeyExpansion(byte Key[4*Nk] word[Nb*(Nr+1)])
{
  For (i=0; i<Nk; i++)
       W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
  For (i=Nk; i<Nb*(Nr+1); i++)
    {  temp =W[i-1];
       if(i%Nk == 0)  temp=SubByte(RotByte(temp))^Rcon[i/Nk];
       W[i]=W[i-Nk]^temp;
    }
```

Particularmente:

$N_k = 4;$ $N_b*(N_r+1)=4(10+1)=44;$

```
For (i=4; i<44; i++)
  {  temp =W[i-1];
     if(i%Nk == 0)  temp=SubByte(RotByte(temp))^Rcon[i/Nk];
     W[i]=W[i-Nk]^temp;
  }
```

Observemos que si i es múltiplo de 4, entonces W[i] sigue un procedimiento especial, en otro caso simplemente W[i] es W[i-1] xor W[i-4].

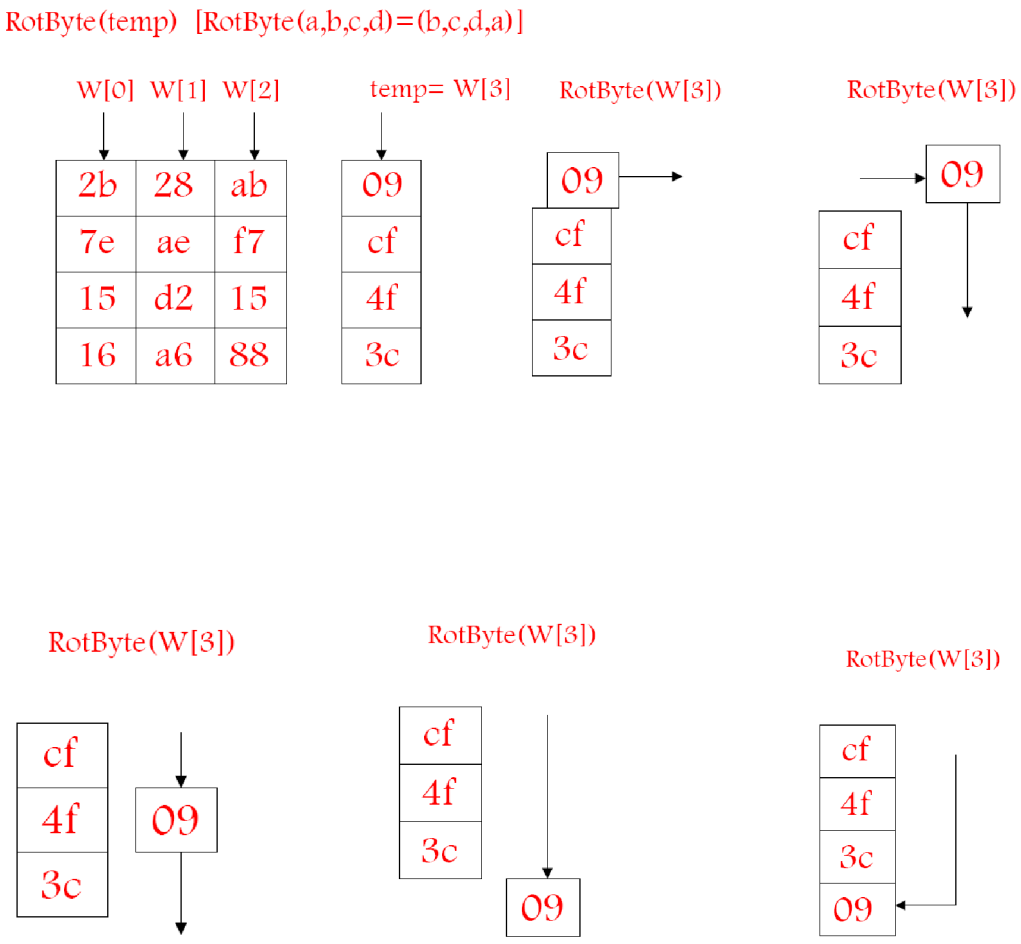
En el caso de i=4, tenemos lo siguiente:

```
{  temp =W[3];
   if(4%4 == 0)  temp=SubByte(RotByte(temp))^Rcon[4/4];
   W[4]=W[0]^temp;
}
```

Entonces el calculo de W[4], es el siguiente:

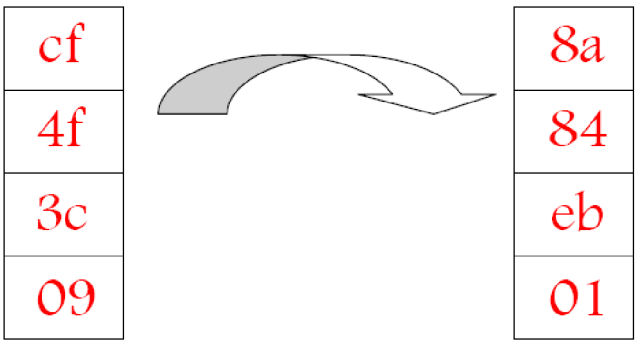
1) temp= RotByte(temp)
2) temp = SubByte(temp)
3) temp = temp xor Rcon[4/4]

Rotbyte es una rotación circular de el byte más arriba de la columna.



SubByte es la substitución de byte, de acuerdo a la tabla ya conocida, de todos los bytes de la columna.

SubByte(RotByte(W[3]));



Finalmente se aplica un xor

temp=SubByte(RotByte(temp))^Rcon[4/4]

Donde, Rcon se obtiene de acuerdo a la siguiente forma:

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

$$RC[1]=1;$$

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

Es decir, desde el punto de vista de polinomios:

$RC[1] = 1$ $\qquad$ $RC[6] = x^5 = 20$

$RC[2] = x = 2$ $\qquad$ $RC[7] = x^6 = 40$

$RC[3] = x^2 = 4$ $\qquad$ $RC[8] = x^7 = 80$

$RC[4] = x^3 = 8$ $\qquad$ $RC[9] = x^8 = x^4 + x^3 + x + 1 = 1b$

$RC[5] = x^4 = 10$ $\qquad$ $RC[10] = x^9 = x^5 + x^4 + x^2 + x = 36$

Así el xor queda de la siguiente manera:.

temp = SubByte(RotByte(temp))^Rcon[1]

8b		8a		01
84	=	84	xor	00
eb		eb		00
01		01		00

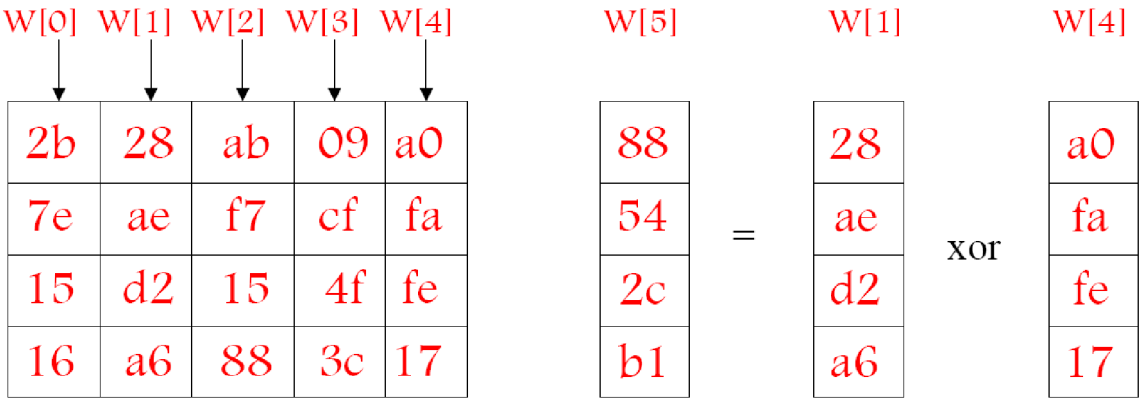
Finalmente para terminar con esté cálculo, tenemos un xor con la columna W[i-4].

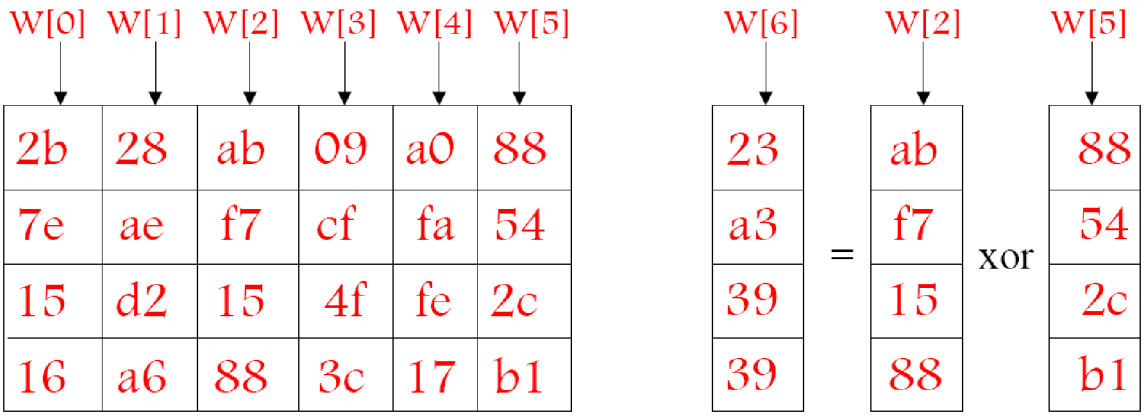
```
i=4;
{  temp =W[3];
   if(4%4 == 0)  temp=SubByte(RotByte(temp))^Rcon[4/4];
   W[4]=W[0]^temp;
}
W[4]=W[0]^temp;
```

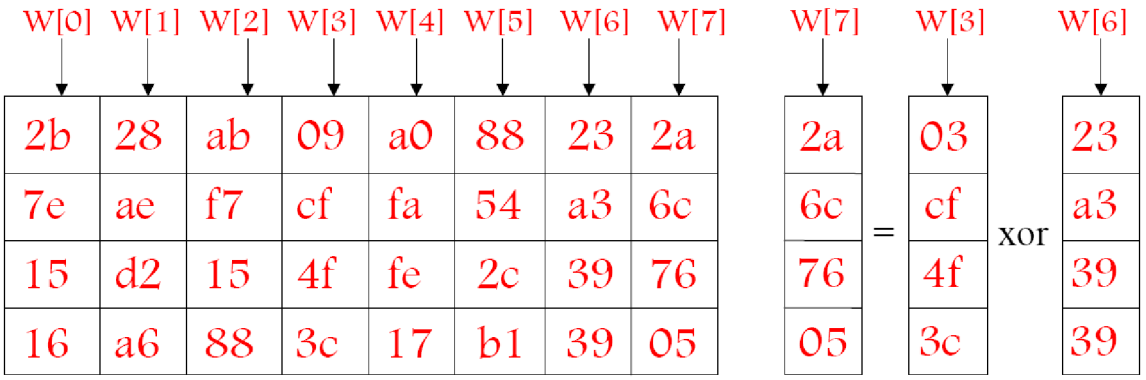
a0		2b		8b
fa	=	7e	xor	84
fe		15		eb
17		16		01

Hemos terminado con la extensión de la primera columna de la calve, las siguientes tres columnas i=5,6,7 no son múltiplo de 4, por lo que su cálculo se reduce a un simple xon con la columna W[i-4].

```
For (i=5,6,7)
  {  temp =W[i-1];
     W[i]=W[i-4]^temp;
  }
```







La siguiente columna i=8, es múltiplo de 4, por lo que hay que seguir la misma regla que la columna 4, es decir:

```
{  temp = W[7];
   If  (8%2 == 0) temp = SubByte(RotByte(temp))^Rcon[8/2];
       W[8] = W[4]^temp;
}
```

Así sucesivamente hasta llenar las 44 columnas de que consiste este caso el programa de clave.

Hasta el momento hemos revisado los pasos mínimos necesarios para poder entender el algoritmo AES, en lo siguiente se darán algunos detalles de la implementación de AES y posteriormente nos daremos algunos otros comentarios interesantes.

5.- Detalles de Implementación del Código AES

En esta sección particularmente detallamos un poco más el código puesto por los autores de Rijndael, el código esta disponible en línea o esta impreso en el libro de los mismos autores[46].

Por razones didácticas, simplifiqué el código en primera instancia para el caso de claves K de 128 bits, y bloques de 128 bits, así como ligeras modificaciones que no tiene mayor propósito que algunos ajustes didácticos.

Este código fue compilado y corrido con Borland c++ v5.01, y cualquier modificación tiene el propósito de hacer más clara la explicación.

```
/* Rijndael code August 01
 * rijndael-alg-ref.c   v2.0   August '99
 * Reference ANSI C code
 * this code is based on the official reference code by
 * Paulo Barreto and  Vincent Rijmen
*/
#include <stdio.h>
#include <conio.h>
#include <string.h>

/*  Definición de funciones */

int rijndaelKeySched (unsigned char k[4][4], unsigned char rk[10+1][4][4]);
int rijndaelEncrypt (unsigned char a[4][4], unsigned char rk[10+1][4][4]);;
int rijndaelDecrypt (unsigned char a[4][4], unsigned char rk[10+1][4][4]);


/*  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,  61, 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,
};
```

```
/*  Definición de los AntiLogaritmos(potencias) de todos los elementos de GF(2^8) base 3 */

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

/*  Definición de la S-caja */
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,
};

/*  Definición de la S-caja inversa (para el descifrado) */

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

/*  Definición de los valores de la función rcon */

unsigned long rcon[30] = {
  0x01,0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,
0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, };

/*  Definición de los valores corrimiento correspondiente a cada caso
    de longitud del bloque o de cifrado y descifrado */
static unsigned char shifts[3][4][2] = {
   0, 0,
   1, 3,
   2, 2,
   3, 1,

   0, 0,
```

```
  1, 5,
  2, 4,
  3, 3,

  0, 0,
  1, 7,
  3, 5,
  4, 4
};

/*  Función que multiplica dos elementos del campo finito GF(2^8)
    usando las tablas  Logtable[] y Alogtable[] */

unsigned char mul(unsigned char a, unsigned char b) {

        if (a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
        else return 0;
}

/* Función que efectúa la transformación  AddRoundKey */

void KeyAddition(unsigned char a[4][4], unsigned char rk[4][4], unsigned char BC) {

        int i, j;


        for(i = 0; i < 4; i++)
                for(j = 0; j < BC; j++) a[i][j] ^= rk[i][j];
}

/* Función que efectúa la transformación ShiftRows */

void ShiftRow(unsigned char a[4][4], unsigned char d, unsigned char BC) {

        unsigned char tmp[4];
        int i, j;

        for(i = 1; i < 4; i++) {
                for(j = 0; j < BC; j++) tmp[j] = a[i][(j + shifts[((BC - 4) >> 1)][i][d]) % BC];
                for(j = 0; j < BC; j++) a[i][j] = tmp[j];
        }
}

/* Función que efectúa la transformación SubBytes */

void Substitution(unsigned char a[4][4], unsigned char box[256], unsigned char BC) {

        int i, j;

        for(i = 0; i < 4; i++)
                for(j = 0; j < BC; j++) a[i][j] = box[a[i][j]] ;
}

/* Función que efectúa la transformación MixColumns */

void MixColumn(unsigned char a[4][4], unsigned char BC) {

        unsigned char b[4][4];
        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];
}
```

```
/* Función que efectúa la transformación MixColumns para el descifrado */

void InvMixColumn(unsigned char a[4][4], unsigned char BC) {

        unsigned char b[4][4];
        int i, j;

        for(j = 0; j < BC; j++)
        for(i = 0; i < 4; i++)
                b[i][j] = mul(0xe,a[i][j])
                        ^ mul(0xb,a[(i + 1) % 4][j])
                        ^ mul(0xd,a[(i + 2) % 4][j])
                        ^ mul(0x9,a[(i + 3) % 4][j]);
        for(i = 0; i < 4; i++)
                for(j = 0; j < BC; j++) a[i][j] = b[i][j];
}

/* Función que genera la extención de la clave K*/

int rijndaelKeySched (unsigned char k[4][4],  unsigned char W[10+1][4][4]) {
        /* Calculate the necessary round keys
         * The number of calculations depends on keyBits and blockBits
         */
        int KC, BC, ROUNDS, s;
        int i, j, t, rconpointer = 0;
        unsigned char tk[4][4];

   KC = 4;
   BC = 4;
         ROUNDS = 10;

        for(j = 0; j < KC; j++)
                for(i = 0; i < 4; i++)
                        tk[i][j] = k[i][j];
        t = 0;
        /* copy values into round key array */
        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) { /* while not enough round key material calculated */
                    for(i = 0; i < 4; i++) tk[i][0] ^= S[tk[(i+1)%4][KC-1]];

                      tk[0][0] ^= rcon[rconpointer++];

                            for(j = 1; j < KC; j++)
                                    for(i = 0; i < 4; i++)  tk[i][j] ^= tk[i][j-1];

        /******* copias la subclave  r esima en el arreglo W  **********/
        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;
}

/* Procedimiento para cifrar */

int rijndaelEncrypt (unsigned char a[4][4],  unsigned char rk[10+1][4][4])
{
        /* Encryption of one block.
         */
        int r, BC, ROUNDS,i,j,k;
  int B[8],X[8],Y[8],XE;

        BC = 4;
  ROUNDS = 10;

        KeyAddition(a,rk[0],BC);

        for(r = 1; r < ROUNDS; r++) {

        Substitution(a,S,BC);
        ShiftRow(a,0,BC);
  MixColumn(a,BC);
        KeyAddition(a,rk[r],BC);
```

```
  gotoxy(6,5);
  printf("CT[");printf("%d",r);printf("] =   ");
  for(i = 0; i < 4; i++)
  for(j = 0; j < 4; j++) printf("%02X ",a[j][i]);

 getchar();
                }

                Substitution(a,S,BC);
                ShiftRow(a,0,BC);
                KeyAddition(a,rk[ROUNDS],BC);

  printf("\n\n\n");
  printf("Cipher Text CT = ");
  for(i = 0; i < 4; i++)
  for(j = 0; j < 4; j++) printf("%02X ",a[j][i]);

  gotoxy(60,24); printf("Press Enter....");
  getchar();
  clrscr();

                return 0;
}

/*  Procedimiento para descifrar */

int rijndaelDecrypt (unsigned char a[4][4], unsigned char rk[10+1][4][4])
{
                int r, BC, ROUNDS;

  BC = 4;
  ROUNDS = 10;
                KeyAddition(a,rk[ROUNDS],BC);
                Substitution(a,Si,BC);
                ShiftRow(a,1,BC);
                /* ROUNDS-1 ordinary rounds
                 */
                for(r = ROUNDS-1; r > 0; r--) {
                                KeyAddition(a,rk[r],BC);
                                InvMixColumn(a,BC);
                                Substitution(a,Si,BC);
                                ShiftRow(a,1,BC);
                }
                KeyAddition(a,rk[0],BC);
                return 0;
}
/**********************/

/* Ejemplo para cifrar y descifrar a A[][] con la clave clave[][] */

int  main ()  {
/**********************/
unsigned char  clave[4][4]={0x00,0x04,0x08,0x0c,
                        0x01,0x05,0x09,0x0d,
                        0x02,0x06,0x0a,0x0e,
                        0x03,0x07,0x0b,0x0f};
unsigned char  A[4][4]={0x00,0x04,0x08,0x0c,
                    0x01,0x05,0x09,0x0d,
                    0x02,0x06,0x0a,0x0e,
                    0x03,0x07,0x0b,0x0f};
/******************/
/*
unsigned char  clave[4][4]={0x2b,0x28,0xab,0x09,
                        0x7e,0xae,0xf7,0xcf,
                        0x15,0xd2,0x15,0x4f,
                        0x16,0xa6,0x88,0x3c};
unsigned char  A[4][4]={0x32,0x88,0x31,0xe0,
                    0x43,0x5a,0x31,0x37,
                    0xf6,0x30,0x98,0x07,
                    0xa8,0x8d,0xa2,0x34};
/******************/
unsigned char  W[10+1][4][4];
int s,i,j;

memset(W,0,sizeof(W));

rijndaelKeySched (clave,W);
```

```
rijndaelEncrypt (A,W);

printf("    Cipher Text  ");
for(i = 0; i < 4; i++)
for(j = 0; j < 4; j++) printf("%02X ",A[j][i]);

printf("\n\n\n");
printf("  In the decipher procces, we apply the inverse transformations.\n");
printf("  \n\n");


rijndaelDecrypt (A,W);
printf("\n");
printf(" Decipher Text  ");
for(i = 0; i < 4; i++)
for(j = 0; j < 4; j++) printf("%02X ",A[j][i]);
gotoxy(50,24); printf("Press Enter to End");
getchar();
return 0;
}
```

A continuación explicaremos paso a paso éste código, tratando de hacer algunos comentarios donde sea necesario.

1.- Procedimiento de Cifrado

```
 /* Procedimiento para cifrar */

int rijndaelEncrypt (unsigned char a[4][4],  unsigned char rk[10+1][4][4])
{

int r, BC, ROUNDS,i,j,k;
int B[8],X[8],Y[8],XE;
BC = 4;
ROUNDS = 10;

        KeyAddition(a,rk[0],BC);

        for(r = 1; r < ROUNDS; r++) {
        Substitution(a,S,BC);
        ShiftRow(a,0,BC);
        MixColumn(a,BC);
        KeyAddition(a,rk[r],BC);

          for(i = 0; i < 4; i++)
          for(j = 0; j < 4; j++) printf("%02X ",a[j][i]);
        }

        Substitution(a,S,BC);
        ShiftRow(a,0,BC);
        KeyAddition(a,rk[ROUNDS],BC);

          for(i = 0; i < 4; i++)
          for(j = 0; j < 4; j++) printf("%02X ",a[j][i]);
        return 0;
}
```

En a[4][4] esta el texto a cifrar como matriz de 16 bytes, en rk[10+1][4][4] Tenemos la expansión de la clave K.

En este caso , BC=4 ya que solo tenemos 4 columnas de texto, además tenemos 4 columnas de clave, por lo tanto ROUNDS=10

Ronda Inicial

Ronda Normal, 9 en éste caso

Sólo imprime los valores intermedios de texto cifrado.

Ronda Final

Imprime el texto cifrado final.

2.- Función KeyAddition

```
void KeyAddition(unsigned char a[4][4], unsigned char rk[4][4], unsigned char BC) {
        int i, j;
        for(i = 0; i < 4; i++)
                for(j = 0; j < BC; j++) a[i][j] ^= rk[i][j];
}
```

Esta función es la más sencilla, y recibe la matriz a[4][4] y rk[4][4], BC=4, entonces regresa los valores de a[4][4] actualizados, simplemente haciendo una operación xor, byte por byte, a[i][j] ^= rk[i][j].

3.- Función Substitution

```
void Substitution(unsigned char a[4][4], unsigned char box[256], unsigned char BC) {

        int i, j;

        for(i = 0; i < 4; i++)
                for(j = 0; j < BC; j++) a[i][j] = box[a[i][j]] ;
}
```

Esta función es también simple, y sólo reemplaza cada byte a[i][j] por el correspondiente box(a[i][j]), es decir:

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

```
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,
};
```

Vista así la tabla, observamos que en la primera fila están colocados los elementos de 0 al 15, es decir en hexadecimal del 00 al 0f, en la segunda del 16 al 31, en hexadecimal 10,..,1f, en la tercera 30,..,3f, en la cuarta 40,..,4f, en la i-ésima i0,..,if, en la última, f0,..,ff, respectivamente, entonces el valor de S correspondiente a a[i][j] = xy en hexadecimal, es el valor que está en la fila x y la columna y.

Por ejemplo:

Lo importante de esta parte es conocer la manera de que se construye la S-box. La descripción de AES dice que la transformación SubByte(a[i][j]) es igual al 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, si representamos a $a[i][j]=a_{ij}$ $T(a_{ij})=A a_{ij}+b$, donde A es una matriz 8x8 de bits, y b es una constante, particularmente b=0x63.

Puede referirse a la descripción de AES para más detalles, en esta sección la aplicación de la SubByte se reduce a la substitución de la correspondiente entrada, es decir, SubByte(a[i][j]) es el byte que esta en la fila i y la columna j de la caja.

4.- Función ShiftRow

```
/* Función que efectúa la transformación ShiftRows */

void ShiftRow(unsigned char a[4][4], unsigned char d, unsigned char BC) {

        unsigned char tmp[4];
        int i, j;

        for(i = 1; i < 4; i++) {
                for(j = 0; j < BC; j++) tmp[j] = a[i][(j + shifts[((BC - 4) >> 1)][i][d]) % BC];
                for(j = 0; j < BC; j++) a[i][j] = tmp[j];
        }
}
```

Esta transformación se aplica a toda la matriz 4x4, a[4][4], dejando la primera fila igual, la segunda fila aplica un shift izquierdo de bytes circular, a la tercera fila aplica 2 shift izquierdos de bytes circulares, y a la cuarta columna aplica 3 bytes.

5.- Función Mixcolumns

```
/* Función que efectúa la transformación MixColumns */

void MixColumn(unsigned char a[4][4], unsigned char BC) {

        unsigned char b[4][4];
        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];
}
```

En esta función se multiplica cada columna de entrada por una columna constante, como observamos al multiplicar la columna por el primer renglón lo hacemos en el orden 2,3 1, 1, que corresponden al código

```
 mul(2,a[i][j])
^ mul(3,a[(i + 1) % 4][j])
^ a[(i + 2) % 4][j]
^ a[(i + 3) % 4][j];
```

El xor significa 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 mod 4.

$$\begin{bmatrix} a_0' \\ a_1' \\ a_2' \\ a_3' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

6.- Función mul

Hemos visto 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 $GF(2^8)$.

```
/* Función que multiplica dos elementos del campo finito GF(2^8)
   usando las tablas Logtable[] y Alogtable[] */

unsigned char mul(unsigned char a, unsigned char b) {

        if (a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
        else return 0;
}
```

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

```
/* 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, };

/* Definición de los AntiLogaritmos(potencias) de todos los elementos de GF(2^8) base 3 */

unsigned char Alogtable[256] = {
  1,  3,  5,  15,  17,  51,  85, 255,  26,  46, 114, 150, 161, 248,  19,  53,  95, 225,  56,  72, 216, 115,
149, 164, 247,  2,  6,  10,  30,  34, 102, 170,  229,  52,  92, 228,  55,  89, 235,  38, 106, 190, 217, 112,
144, 171, 230,  49,  83, 245,  4,  12,  20,  60,  68, 204,  79, 209, 104, 184, 211, 110, 178, 205,  76,
212, 103, 169, 224,  59,  77, 215,  98, 166, 241,  8,  24,  40, 120, 136,  131, 158, 185, 208, 107, 189,
220, 127, 129, 152, 179, 206,  73, 219, 118, 154,  181, 196,  87, 249,  16,  48,  80, 240,  11,  29,  39,
105, 187, 214,  97, 163,  254,  25,  43, 125, 135, 146, 173, 236,  47, 113, 147, 174, 233,  32,  96, 160,
251,  22,  58,  78, 210, 109, 183, 194,  93, 231,  50,  86, 250,  21,  63,  65, 195,  94, 226,  61,  71, 201,
64, 192,  91, 237,  44, 116, 156, 191, 218, 117, 159, 186, 213, 100, 172, 239,  42, 126, 130, 157, 188,
```

223, 122, 142, 137, 128, 155, 182, 193, 88, 232, 35, 101, 175, 234, 37, 111, 177, 200, 67, 197, 84, 252, 31, 33, 99, 165, 244, 7, 9, 27, 45, 119, 153, 176, 203, 70, 202, 69, 207, 74, 222, 121, 139, 134, 145, 168, 227, 62, 66, 198, 81, 243, 14, 18, 54, 90, 238, 41, 123, 141, 140, 143, 138, 133, 148, 167, 242, 13, 23, 57, 75, 221, 124, 132, 151, 162, 253, 28, 36, 108, 180, 199, 82, 246, 1, };

La primera tabla son los elementos en decimal que son potencias del polinomio 1+x,. La tabla Logtable contiene los valores de la potencia de 1+x a la que hay que elevar el elemento (1+x) para que de el valor dado.

Podemos hacer algunas cálculos de esto para quedar convencidos del contenido de la tabla.

potencia	$GF(2^8)$	binario	decimal
i	$(1+x)^i$		$(3)^i \bmod 256$
0	1	1	1
1	$x+1$	11	3
2	x^2+1	101	5
3	x^3+x^2+x+1	1111	15
4	x^4+1	10001	17
5	x^5+x^4+x+1	110011	51
6	$x^6+x^4+x^2+1$	1010101	85
7	$x^7+x^6+x^5+x^4+x^3+x^2+x+1$	11111111	255
8	x^4+x^2+x	10110	26
9	x^4+x^2+x	101110	46
10	$x^6+x^5+x^4+x$	1110010	114

Por otro lado la algunos cálculos de la tabla ALogtable quedan de la siguiente manera:

	i	$3^i = (1+x)^i$
0	0	
1	0	$3^0 = (1+x)^0 = 1$
2	25	$3^{25} = (1+x)^{25} = 2$
3	1	$3^3 = (1+x)^3 = 3$
4	50	$3^{50} = (1+x)^{50} = 4$
5	2	$3^2 = (1+x)^2 = 5$
6	26	$3^{26} = (1+x)^{26} = 6$
7	198	$3^{198} = (1+x)^{198} = 7$
8	75	$3^{75} = (1+x)^{75} = 8$
9	199	$3^{199} = (1+x)^{199} = 9$

Las operaciones se pueden mostrar de la tabla de potencias, generadas por elemento (1+x).

Una pregunta recurrente sobre el generador del campo finito, se refiere a que si se hubiera elegido un polinomio primitivo para la construcción del campo entonces el generador simplemente hubiera sido x, sin embargo como veremos después esto no altera nada la seguridad del algoritmo

7.- Función Keysched

Finalmente revisemos la parte de extensión de la clave. Esta función recibe un clave k, de 4 columnas y regresa un arreglo W de 44 columnas.

```
int rijndaelKeySched (unsigned char k[4][4],  unsigned char W[10+1][4][4]) {
        /* Calculate the necessary round keys
         * The number of calculations depends on keyBits and blockBits
         */
        int KC, BC, ROUNDS, s;
        int i, j, t, rconpointer = 0;
        unsigned char tk[4][4];

   KC = 4;
   BC = 4;
         ROUNDS = 10;

        for(j = 0; j < KC; j++)
                for(i = 0; i < 4; i++)
                        tk[i][j] = k[i][j];
        t = 0;
        /* copy values into round key array */
        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) {  /* while not enough round key material calculated */
                    for(i = 0; i < 4; i++) tk[i][0] ^= S[tk[(i+1)%4][KC-1]];

                      tk[0][0] ^= rcon[rconpointer++];

                        for(j = 1; j < KC; j++)
                                for(i = 0; i < 4; i++)  tk[i][j] ^= tk[i][j-1];

        /******* copias la subclave  r esima en el arreglo W  **********/
        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;
}
```

Se copia el valor de la clave en la variable tk.

Se copia la primera subclave en el arreglo W.

Se generan la siguiente subclave, y se hace esto tanto como sea necesario. Si es una columna modulo 4 se opera un xor que se toma de la tabla rcon[].

Después de generada la subclave se copia en el arreglo W.

6.- Eficiencia de AES

La eficiencia del algoritmo AES es otra de las tan anheladas propiedades que debe de contar un algoritmo criptográfico. La eficiencia además de depender del diseño del algoritmo y de su implementación depende de la plataforma donde se corra entre otras cosas. Obviamente las mejores eficiencias alcanzadas para cualquier algoritmo se alcanzan en la implementación de hardware dedicados. Se tiene entonces que el campo de la eficiencia esta dividida en dos campos uno de ellos es el software y el otro el hardware.

La eficiencia en software en AES: el código de partida de AES es el conocido que muestra el funcionamiento de las funciones básicas de 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 el obtener inversos multiplicativos del campo GF(2^8), en este caso esta operación es precalculada y la operación es substituida por un consulta de S-Box, con esta misma técnica varios cálculos del algoritmo son precalculados y entonces se evitan los cálculos reemplazándolos por consultas de tablas.

Respecto a la velocidad de los algoritmos las manera más simple de medirla es usando Mb/seg que significa MegaBits por segundo es decir, millones de bits por segundo, un algoritmo puede medirse entonces por cuantos millones de bits por segundo procesa. Los tres procesos más comunes en un algoritmo son el cifrado, el descifrado y el programa de claves, estos tres se miden de manera independiente.

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.

Plataforma PIV 3.2GHz, assembly
Autor H. Lipmaa, Lipmaa Web Page
Longitud de clave 128
Velocidad de Cifrado 1537.9.7 Mb/s
Velocidad de Descifrado 1519.9 Mb/s

Plataforma PPro 200 MHz, C
Autor B.Gladman, Gladman Web Page
Longitud de clave 128
Velocidad de Cifrado 70.7 Mb/s
Velocidad de Descifrado 71.5 Mb/s

Plataforma PIV 1.8GHz, C++
Autor C. Devine, Devine Web Page
Longitud de clave 128
Velocidad de Cifrado 646 Mb/s

Plataforma PII 450MHz, MMX Assembly
Autor K. Aoki, H. Lipmaa [17]
Longitud de clave 128
Velocidad de Cifrado 243 Mb/s

Por otra parte los diseños de hardware especialmente para la implementación de Rijndael requiere de un estudio más especializado que esta fuera del alcance de este reporte sin embargo aquí damos algunos de los resultados también representativos de esta otra parte de la eficiencia con AES.

Tecnología ASIC
Autor H. Kuo, I. Verbauwhede, P. Schaumont [60]
Velocidad 2.29 Gb/s, cifrado, 128b.

Tecnología VLSI
Autor H. Kuo, I. Verbauwhede [37]
Velocidad 1.82 Gbits/s

Tecnología FPGA[24]
Autor A.J. Elbirt
Velocidad 2 Gb/s, cifrado, 128b.

Tecnología Smart Cards Z80
Autor F. Sano [7]
Velocidad 25 494 Clock (2 veces más rápido que DES)

Tecnología TMS320C62201 DSP
Autor T.Wollingr, M. Wang, J. Guajardo, C. Paar [14]
Velocidad 112 Mbits/s (200 Mhz)

Desde que se inicio el proceso AES la eficiencia ha estado y estará siendo causa de una enorme investigación. En la bibliografía encontraremos gran parte de las nuevas aportaciones que se hacen en esta área tanto en software como en hardware[54][60][63][74][95][103][110][115].

7.- Seguridad de AES

El aspecto más importante de un algoritmo es su seguridad, sin embargo, la seguridad no es algo fácil de manejar, en este reporte solo se describen aspectos muy generales de la seguridad de AES, un estudio más serio queda fuera del alcance de este documento, y el lector debe de dirigirse directamente a la bibliografía recomendada. El propósito de esta sección es simplemente como complemento a la descripción general de AES, se puede ver más detalladamente en[78].

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 taques 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 (por ejemplo por la IACR), 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.). 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 al 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.

Enseguida trataremos de explicar de la manera más sencilla posible las anteriores dos características.

1) Linealidad

En términos elementales por ejemplo si las entradas de nuestro algoritmo son 1,2,3,4, y 5 y tenemos como salidas 2,4,6,8, y 10, claramente hay una dependencia lineal entre las entradas y las salidas, es decir $f(x)=2g(x)$ donde f,g son las funciones de entrada y salida correspondientemente. El método conocido como correlación es el que se aplica a dos conjuntos de datos A, B y determina que tanto pueden haber uno del otro una dependencia lineal. De tal modo que si existe una relación lineal entre las entradas de las salidas, no es nada difícil conocer la información de los textos originales o de la clave de

cifrado. Claramente la linealidad no es una propiedad que se quiera en un algoritmo criptográfico.
El mecanismo que impide que haya correlación es el alternar las claves, es decir que para cada ronda de nuestro algoritmo se aplique una clave diferente, esto se logra en términos generales implementado un programa de claves que proporciona una clave diferente para cada ronda. Esto permite disminuir la linealidad en la mayoría de las modalidades que pueda pensarse como una debilidad de nuestro algoritmo, y es aprovechada principalmente por el criptoanálisis lineal.

2) Propagación

El otro concepto muy trabajado en una algoritmo simétrico es la propagación, este concepto hay que trabajarlo de la manera más cuidadosa, el que el algoritmo tenga buena "propagación" impide que sea aplicado principalmente el criptoanálisis diferencial que es un ataque del tipo "Chosen Plaintext Attack" y permite derivar información de la clave a partir de conocer las probabilidades de las diferencias de la propagación, por lo que estas probabilidades deben de ser lo más pequeño posible. La propagación se obtiene con el programa de claves, con la propagación de la función no lineal del algoritmo (S-Box), el número de rondas, etc.

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

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 Rijndael, lo que lo hace tan seguro como una búsqueda exhaustiva.

Describamos ligeramente algunos de los últimos ataques, que de alguna manera son versiones modificadas del los ataques diferencial y lineal.

Impossible Differentials Attack: existe un ataque de este tipo a 5 rondas de AES, requiriendo 2^{29} plaintext elegidos, 2^{30} encripciones, 2^{42} bytes de memoria, 2^{26} pasos de precalculo. Estas condiciones fueron mejoradas en [31][33] para alcanzar un ataque a 6 rondas de AES.

Square Attack: el mas potente ataque a 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 al algoritmo predecesor llamado "Square". Este ataque puede romper a Rijndael de 6 a 7 rondas, que puede ser mejorado para atacar a 9 rondas de AES-259 con 2^{77} plaintexts, con 256 claves relacionadas, y 2^{224} encripciones[13].

Collision Attack: este ataque afecta a todas las versiones de AES, 128, 192 y 256 con 7 rondas[29].

Los ataques anteriores son de alguna manera el "último" intento de diseñar un ataque a AES 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 plaintexts, los ciphertexts, 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, éste puede ser expresado de diferentes maneras de una forma elegante.

Algunas "ideas" o "ataques" algebraicas/os pueden ser listados ahora:

Fracciones continuas: una de las primeras propuestas fue hecha en [41] y proponen una sola formula que puede ser vista como una fracción continua como la siguiente:

$$x = K + \sum \frac{C_1}{K^* + \sum \frac{C_2}{K^* + \sum \frac{C_3}{K^* + \sum \frac{C_4}{K^* + \frac{C_5}{K^* + P_*^*}}}}}$$

Donde cada K es un byte que depende de varios bytes de la clave expandida, los C_i son constantes conocidas, y los * 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.

XSL: En [56] los autores observaron que las S-cajas de AES pueden ser descritas por ecuaciones cuadráticas booleaneas, 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 [43] 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.

Embedding: otra idea que fue expuesta es la de Murphy y Robshaw [53], tratando de encajar a AES en otra algoritmo llamado BES(Big Encryption System), de la siguiente manera:

$$Rijndael_K(x) = \phi^{-1}\left(BES_{\phi(K)}(\phi(x))\right)$$

donde K es la clave, x el mensaje y ϕ un mapeo de la estructura de campo de Rijndael a la estructura de campo de BES. Como Rijndael usa al combinaciones de los campos GF(2) y GF(2^8), BES sólo 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 Rijndael. Se afirma aquí también que el método XSL es puede ser aplicado a BES con ligeras mejorías en los resultados.

Dual Cipher: [52]Otra manera de encajamiento es definir un algoritmo dual a AES, esto quiere decir una especie de traslación.
Si tenemos los mapeos invertibles f,g,h entonces el Cipher "Dual" es definido como:

$$Rijndael_K(x) = f^{-1}Dual_{g(K)}(h(x))$$

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.

En [108] podemos encontrar algo sobre los últimos taques de tipo algebraico. En [116] una manera nueva de ataque considerando una versión reducida de AES. En [107] lo último del ataque Boomerang. En [114] combinaciones de ataques. En [97] sobre el ataque "Imposible Differential ". En [93] sobre la linealidad del programa de claves. En [61],[83],[90], y [91] algo sobre "Power Analysis",

8.- Modificaciones de AES

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.

Las primeras ideas son muy simples, como FIPS 197 donde se describe AES determina 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 debe hacerse de manera muy cuidadosa para no perder las principales características de seguridad que tiene AES. Obviamente uno esperaría con estos cambios al menos tener las mismas características del original AES, o eventualmente alguna mejora.

- 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.

Un estudio más completo lo dieron E. Barkan, y E. Biham [52] al mostrar que si cambiamos todas las constantes involucradas en el algoritmo AES no tendremos alteración importante alguna, esto 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 $GF(2^8)$.

Para explicar esto con detalle damos las siguientes definiciones:

Def 1: Dos algoritmos E y E' son "Duales" si son isomorfos, es decir,
si existen funciones invertibles f, g, y h tales que:
para cualquier plaintext P y clave K, $f(E_K(P)) = E'_{g(K)}(h(P))$.

Particularmente si f,g,h son la función cuadrado entonces podemos mostrar que los algoritmos E, E^2 son duales, es decir son equivalentes, particularmente $E^2_{K^2}(P^2) = (E_K(P))^2$ es decir que si usamos AES con los plaintext (que son elementos de un campo finito) al cuadrado, con la clave K al cuadrado y aplicamos E^2 (definición en detalle posterior), entonces el resultado sería el mismo al usar el original AES 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 son:

1) En ByteSub la matriz y la constante de la aplicación lineal.

Para la transformación afín $Ax+b$ la debemos cambiar por $QAQ^{-1}x+b^2$.

2) En MixColumn el polinomio a ser multiplicado.

   En MixColumn el polinomio $03x^3+01x^2+01x+02$ debe ser cambiado por $05x^3+01x^2+01x+04$, en el proceso de descifrado el polinomio $0bx^3+0dx^2+09x+0e$ se cambian por $0dx^3+0bx^2+0ex+09$.

3) En el programa de claves, las constantes Rcon.

   Las constantes de Rcon $(02)^{i-1}$ por las constantes $(03)^{i-1}$.

4) El polinomio irreducible para construir el campo $GF(2^8)$.

   El polinomio irreducible puede ser cambiado por alguno de los 30 que existen y que están listados en los antecedentes matemáticos.

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.

Posteriormente Biryukov, De Cannière, Braeken, y Preneel [79], generalizaron estos cifradores isomorfos hasta 61200, encontrando algunos equivalentes a las S-cajas Esta técnica también puede ser aplicada a otros algoritmos importantes como Camellia, Misty, y Kasumi.

Aunque es buena la idea de usar para un caso especial una diferente manera de AES, su aplicación más importante se ha dado en el criptoanálisis, se cree que con el número tan importante de diferentes maneras de reescribir el algoritmo, quizá en alguna de ellas se pueda derivar un ataque eficiente que después se traslade al original AES. Otra aplicación es poder evitar ataques como el análisis de potencial.

9.- Estado Actual de AES

El estado actual de AES por procedimiento es proporcionada por el NIST, es probable que en cada periodo de tiempo el NIST de un estado principalmente de la seguridad de AES. El último reporte de este estilo fue dado en la conferencia AES4, donde se afirmaron los siguientes puntos[94]:

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[107][108].
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, HMAC[40], IPsec, IKE, S/MIME, SIP, etc. o RFIDs[85].

10.- Bibliografía

Referencias \ AES (Advanced Encryption Standard)

2005

[117] "A Side-Channel Analysis Resistant Description of the AES S-box", Elisabeth Oswald, Stefan, Mangard, Norbert Pramstaller and Vincent Rijmen, to apper in FSE 2005.

[116] "Small Scale Variants of the AES", C. Cid, S. Murphy and M. Robshaw, Proceedings of Fast Software Encryption 2005: 12th International Workshop, Paris.

2004

[115] "A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA", Alireza Hodjat, Ingrid Verbauwhede, IEEE Symposium on Field -Programmable Custom Computing Machines.

[114] "A Collision-Attack on AES Combining Side Channel- and Differential attack", Kai Schramm, Gregor Leander, Patrick Felke, Christof Para, AES4 Horst Görtz Institute

[113] "A High Throughput AES-Based Programmable Crypto Coprocessor for Networking Applications", Alireza Hodjat, Ingrid Verbauwhede, IEEE Micro Magazine, Special Issue on WASP.

[112] "A Masked AES ASIC Implementation",Pramstaller N., Oswald E., Mangard S., Gürkaynak F., and Haene S, in Proceedings of Austrochip 2004, Villach, Austria, Oct. 8, 2004.

[111] "A three rounds property of the AES", Marine Minier, AES4 Horst Görtz Institute.

[110] "A Universal and Efficient AES Co-Processor for Field Programmable Logic Arrays", Pramstaller N., Wolkerstorfer J.,Proceedings of Field Programmable Logic and Application—FPL 2004, LNCS 3203, Springer Verlag, ISBN 3-540-22989-2, pp. 565-574, 2004

[109] "Algebraic Cryptanalysis of AES: an overview", H. Nover, U. of Wisconsin.

[108] "An algebraic interpretation of AES-128", Ilia Toli, Alberto Zanoni, AES04, Horst Görtz Institute.

[107] "The Boomerang attack on 5- and 6-round AES", Alex Biryukov, AES4 Horst Görtz Institute.

[106] "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of AES Rijndael Very Well Suited for Small Embedded Applications", Gaël Rouvroy, François-Xavier Standaert, Jean-Jacques Quisquater, Jean-Didier Legat, ITCC 2004, IEEE Computer Society.

[105] "Computational and Algebraic Aspects of the Advanced Encryption Standard", C. Cid, S. Murphy and M. Robshaw, Proceedings of the Seventh International Workshop on Computer Algebra in Scientific Computing, CASC' 2004, pp. 93-103.
[104] "Cyclic Properties of AES Round Functions", Yvo Desmedt, AES4 Horst Görtz Institute.
[103] "Efficient AES implementations on ASICs and FPGAs", Sandra Dominikus, Stefan Mangard, Norbert Pramstaller, Johannes Wolkerstorfer, AES4 Horst Görtz Institute.
[102] "Elastic AES", Debra L. Cook and Moti Yung and Angelos D. Keromytis, eprint-iacr 2004/141.
[101] "Electromagnetic Side Channels of an FPGA Implementation of AES", Vincent Carlier, Hervé Chabanne, Emmanuelle Dottax and Hervé Pelletier eprint-iacr 2004/145.
[100] "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)", Johannes Blömer, Jean-Pierre Seifert, FC 2004, Proceedings, LNCS Vol. 2742, pp 162-181.
[99] "General Principles of Algebraic Attacks and New Design Criteria for Components of Symmetric Ciphers", Nicolas T. Courtois, AES4 Horst Görtz Institute.
[98] "Gröbner algorithms: Algebraic systems related to AES", Jean-Charles Faugere, AES4 Horst Görtz Institute.
[97] "Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES) ", R. C.-W. Phan, Information Processing Letters, Elsevier Science, Vol. 91, No 1, pp. 33-38, July 2004.
[96] "More dual Rijndaels", Håvard Raddum, AES4 Horst Görtz Institute.
[95] "Minimum Area Cost for a 30 to 70 Gbits/s AES Processor", Alireza Hodjat, Ingrid Verbauwhede, Proceedings of IEEE computer Society Annual Symposium on VLSI, Pages: 83-88.
[94] "NIST and AES in 2004", John Kelsey, AES4 Horst Görtz Institute.
[93] "Linearity of the AES key schedule, Frederik Armknecht, Stefan Lucks, AES4 Horst Görtz Institute.
[92] "Secure and Efficient AES Software Implementation for Smart Cards", E. Trichina and L. Korkishko eprint-iacr 2004/149.
[91] "Power Analysis of an FPGA Implementation of Rijndael: Is Pipelining a DPA Countermeasure?", François-Xavier Standaert, Siddika Berna Ors, Bart Preneel, CHES 2004, LNCS 3156.
[90] "Power-Analysis Attack on an ASIC AES implementation", S. B. Örs, F. Gürkaynak, E. Oswald and B. Preneel Proceedings of the 2003, International Symposium on Information Technology (ITCC 2004), 5-7 April 2004, Las Vegas, NV, USA. IEEE Computer Society 2004, pp. 546-552
[89] "Provably Secure Masking of AES" , Johannes Blömer, Jorge Guajardo Merchan and Volker Krummel eprint-iacr 2004/101. SAC 2004.
[88] "Refined analysis of bounds related to linear and differential cryptanalysis for the AES, Liam Keliher, AES4 Horst Görtz Institute.

[87] “Secure and Efficient Masking of AES - A Mission Impossible?”, Elisabeth Oswald and Stefan Mangard and Norbert Pramstaller, eprint-iacr 2004/134.
[86] “Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results”, Elena Trichina and Tymur Korkishko, Proceedings of the Fourth Conference on the Advanced Encryption Standard (AES) , 2004.
[85] “Strong Authentication for RFID Systems using the AES Algorithm”, Martin Feldhofer and Sandra Dominikus and Johannes Wolkerstorfer, CHES 2004, LNCS Vol. 3156.
[84] “Towards an AES Crypto-chip Resistant to Differential Power Analysis", Pramstaller N.,Gürkaynak F., Haene S., Kaeslin H.,Felber N., and Fichtner W.,Proceedings of ESSCIRC 2004, Leuven, Belgium, Sept. 21-23, 2004
[83] “Two Power Analysis Attacks against One-Mask Methods”, M.-L. Akkar and R. Bevan and L. Goubin, FSE 2004 -- Pre-proceedings , pp. 308-325.

2003

[82] “A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZA”, Gilles Piret, Jean-Jacques Quisquater, CHES 2003, LNCS Vol. 2779.
[81] “A Highly Regular and Scalable AES Hardware Architecture”, Stefan Mangard and Manfred Aigner and Sandra Dominikus, IEEE Transactions on Computers, Vol. 52, pp. 483-491 , 2003.
[80] “A Generic Protection against High-Order Differential Power Análisis”, Mehdi-Laurent Akkar and Louis Goubin, FSE 2003, Revised Papers , LNCS Vol. 2887, pp. 192-205.
[79] “A Tool Box for Cryptanalysis: Linear and Affine Equivalence Algorithms”, A. Biryukov, C. De Cannière, A. Braeken, B. Preneel, EUROCRYPT'2003, pp.33-50.
[78] “AES- The State of the Art of Rijndael’s Security”, E. Oswald, J. Daemen, V. Rijmen, October 2002.
[77] “An Efficient AES Implementation for Reconfigurable Devices”, Norbert Pramstaller
[76] “Combinational Logic Design for AES SubByte Transformation on Masked Data”, Elena Trichina eprint-iacr 2003/236.
[75] “Cryptanalysis of the Advanced Encryption Standard (AES): 2002 ± 5", R. C.-W. Phan , Techno-Legal Aspects of Information Society and New Economy: an Overview, Formatex, Spain, pp. 269-276, 2003.
[74] “Design and Performance Testing of a 2.29 Gb/s Rijndael Processor”, Ingrid Verbauwhede and Patrick Schaumont and Henry Kuo, IEEE Journal of Solid-State Circuits , pp. 569-572.
[73] “Did Filiol Break AES ?”, Nicolas T. Courtois and Robert T. Johnson and Pascal Junod and Thomas Pornin and Michael Scott, eprint-iacr 2003/022.
[72] “Differential Fault Analysis on A.E.S.”, P. Dusart, G. Letourneux and O. Vivolo, eprint-iacr ,2003/010.

[71] “Differential Fault Analysis on AES Key Schedule and Some Coutnermeasures”, Chien-Ning Chen, Sung-Ming Yen, ACISP 2003, Proceedings, LNCS Vol. 2727, pp. 118-129.
[70] “DFA on AES”, Christophe Giraud , eprint-iacr , 2003/008.
[69] “Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs”, François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, Jean-Didier Legat, CHES 2003, LNCS Vol. 2779.
[68] “Further observations on the structure of the AES algorithm”, B. Song, J. Seberry, FSE 2003.
[67] "Impossible Differential Cryptanalysis of Mini-AES ", R. C.-W. Phan,Cryptologia, Vol XXVII, No 4, October 2003.
[66] "Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students", R. C.-W. Phan, Cryptologia, Vol XXVI, No 4, October 2002
[65] “Novel Cyclic and Algebraic Properties of AES”, Tri Van Le, eprint-iacr 2003/10
[64] “Plaintext-dependant Repetition Codes Cryptanalysis of Block Ciphers - The AES Case”, Eric FILIOL, eprint-iacr, 2003/003.
[63] “Speed-area trade-off for 10 to 100 Gbits/s throughput AES processor”, Alireza Hodjat, Ingrid Verbauwhede, IEEE Asilomar Conference on Signals, Systems, and Computers.
[62] “Very Compact FPGA Implementation of the AES Algorithm”, Pawel Chodowiec and Kris Gaj, CHES 2003, Proceedings , LNCS Vol. 2779 , pp. 319-333.

2002

[61] “A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion”, Stefan Mangard, ICISC 2002, Revised Papers , LNCS Vol. 2587, pp. 343-358.
[60] “A 2.29 Gb/s, 56 mW non-pipelined Rijndael AES Encryption IC in a 1.8 V, 0.18 um CMOS Technology”, H. Kuo, I. Verbauwhede, P. Schaumont, Custom Integrated Circuits Conference.
[59] “About Filliol's Observations on DES, AES and Hash Functions (draft)”, Nicolas T. Courtois, eprint-iacr, 2002/149.
[58] “An ASIC implementation of the AES SBoxes”, Johannes Wolkerstorfer and Elisabeth Oswald and Mario Lamberger”, CT-RSA 2002, LNCS Vol. 2271, pp. 67-78.
[57] “Algebraic Structures and Cycling Test of Rijndael”, Y. Desmedt, T.V. Le, M. Marrotte, J.J. Quisquater, FSE 2002.
[56] “Cryptanalysis of Block Ciphers with Overdefined Systems of Equations”, *Nicolas* Courtois and Josef Pieprzyk, ePrint 2002/044.
[55] “Comments on the Security of the AES and XSL Technique”, S. Murphy and M.J.B. Robshaw. NES/DOC/RHU/WP5/026/1.
[54] “Efficient Software Implementation of AES on 32-Bit Platforms”, Guido Bertoni and Luca Breveglieri and Pasqualina Fragneto and Marco Macchetti and Stefano Marchesin, CHES 2002, Revised Papers , LNCS Vol. 2523 , pp. 159-171.

[53] “Essential algebraic structure within the AES”, S. Murphy and M. Robshaw, Crypto’02, LNCS 2442 pp 17-38.
[52] “In How Many Ways Can You Write Rijndael?”, Elad Barkan and Eli Biham eprint-iacr, 2002/157.
[51] “Multiplicative Masking and Power Analysis of AES”, Jovan D. Golic and Christophe Tymen, CHES 2002, Revised Papers , LNCS Vol. 2535, pp. 198-212.
[50] “On Linear Redundancy in the AES S-Box”, Joanne Fuller and William Millan, eprint-iacr, 2002/111.
[49] “On Some Algebraic Structures in the AES Round Function”, A.M. Youssef and S.E. Tavares eprint-iacr, 2002/144.
[48] “Rijndael for Algebraists”, H.W. Lenstra, April 2002.
[47] “Simplified Adaptive Multiplicative Masking for AES”, Elena Trichina and Domenico De Seta and Lucia Germani, CHES 2002, Revised Papers , LNCS Vol. 2535 , pp. 187-197.
[46] “The Design of Rijndael”, J. Daemen, V. Rijmen, Springer Verlag 2001.
[45] “The Book of Rijndaels”, Elad Barkan and Eli Biham, eprint-iacr, 2002/158.
[44] “Unlocking the Design Secrets of a 2.29 Gb/s Rijndael Core”, P. Schaumont, H. Kuo, I. Verbauwhede, Design Automation Conference 2002.
[43] “XSL Against Rijndael”, D. Coppersmith, CryptoGram October 2002.

2001

[42] “A Compact Rijndael Hardware Architecture with S-Box Optimization”, Akashi Satoh and Sumio Morioka and Kohji Takano and Seiji Munetoh, ASIACRYPT 2001, Proceedings , LNCS Vol. 2248 , pp. 239-254.
[41] “A simple algebraic representation of Rijndael”, N. Ferguson, R. Schroeppel, D. Whiting, Draft.
[40] “AES-hash”, B. Cohen, B. Laurie, NIST Draft.
[39] “An ASIC Implementation of the AES-MixColumn operation”, Johannes Wolkerstorfer, Austrochip 2001 , pp. 129-132.
[38] “An Implementation of DES and AES, Secure against Some Attacks”, Mehdi-Laurent Akkar and Christophe Giraud, CHES 2001, Proceedings , LNCS Vol. 2162, pp. 309- 318.
[37] “Architectural Optimization for a 1.82 Gb/s VLSI Implementation of the AES Rijndael algorithm”, H. Kuo, I. Verbauwhede, Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, pp 51-64.
[36] “Efficient Rijndael Encryption Implementation with Composite Field Arithmetic”, Atri Rudra and Pradeep K. Dubey and Charanjit S. Jutla and Vijay Kumar and Josyula R. Rao and Pankaj Rohatgi, CHES 2001, Proceedings , LNCS Vol. 2162 pp. 171-184.
[35] “Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining”, Pawel Chodowiec and Po Khuon and Kris Gaj, FPGA 2001, Proceedings, pp. 94-102.

[34] "FIPS-197": Advanced Encryption Standard, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[33] "Generalised impossible differentials of advanced encryption standard", R. C. W. Phan & M. U. Siddiqi, IEE Electronics Letters, Vol. 37, Issue 14, pp. 896-898, July 2001.
[32] "High Performance Single-Chip FPGA Rijndael Algorithm Implementations", CHES 2001, Proceedings , LNCS Vol. 2162 , pp. 65-76.
[31] "Improved Impossible Differential Cryptanalysis of Rijndael and Crypton", J.H. Cheon, M. Kim, K. Kim, J. Lee, S. Kang, ICISC 2001, LNCS 2288 pp. 39-49.
[30] "Two Methods of Rijndael Implementation in Reconfigurable Hardware", Viktor Fischer and Milos Drutarovsky, CHES 2001, Proceedings , LNCS Vol. 2162, pp. 77-92.

2000

[29] "A collision Attack on 7 rounds of Rijndael", H. Gilbert, M. Minier, AES3papers-2-11.
[28] "A Comparative Study of Performance of AES Final Candidates Using FPGAs", A. Dandalis, V.K. Prasanna, J.D.P. Rolim, AES3papers-4-23.
[27] "A comparison of the AES candidate on the Alpha 21264", R. Weiss, N.Binkert, AES3papers-3-18.
[26] "A Comparison of the AES Candidates Amenability to FPGA Implementation", N. Weaver, J. Wawrzynek, AES3papers-2-13.
[25] "A Performance Comparison of the Five AES Finalists", B. Schneier, D. Whiting, AES3papers-3-17.
[24] "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, NIST AES3papers-1-08.
[23] "Answer to New observations on Rijndael", J. Daemen, V. Rijmen, draft 2000.
[22] "AES Finalist on PA-RISC and IA-64 Implementations \& Performance", J. Worley, B. Worley, T. Chistopher, C. Worley, NIST.
[21] "Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Key", S. Lucks, NIST AES3papers-1-04.
[20] "Comparison oh the hardware performance of the AES candidates using reconfigurable hardware", K. Gaj, P.Chodowiec, AES3papers-3-22.
[19] "Cryptanalysis of Reduced Variants of Rijndael", E. Biham, N. Keller, AES3papers-5-35.
[18] "Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standar", L. E. Bassham, AES3papers-4-29.
[17] "Fast implementation of the AES Candidates", K.Aoki, H.Lipmaa, AES3papers-3-20.
[16] "Hardware Evaluation of the AES Finalists", T. Ichikawa, T. Kasuya, M. Matsui, AES3papers-3-15.
[15] "Hardware Performance Simulations of Round 2 Advanced Encryption Standar Algorithms", B.Weeks, M.Bean, T.Rozylowicz, C.Ficke, AES3papers-5-37.

[14] “How Well Are High-End DSPs Suited for the AES Algorithms?”, T.J. Wollinger, M. Wang, J. Guajardo, C. Paar, AES3papers-2-12.
[13] “Improved cryptanalysis of Rijndael”, N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Staty, D. Wagner, D. Wagner, D. Whiting, FSE 00, LNCS 1978, pp 213-230.
[12] “New observations on Rijndael”, S. Murphy, M. Robshaw, draft 2000.

[11] “NIST Performance Analysis of the Final Round Java AES Candidates”, J. Dray, AES3papers-4-28.
[10] “On Boolean and Arithmetic Masking against Differential Power Analysis”, Jean-Sebastien Coron and Louis Goubin, CHES 2000, Proceedings , LNCS Vol. 1965, pp. 231-237.
[9] “Performance of the AES Candidate Algorithms in Java”, A. Sterbenz, P. Lipp, NIST AES3papers-1-03.
[8] “Performance Comparison of the 5 AES Candidates with New Performance Evaluation Tool”, M. Takenaka, N.Torii, K. Itoh, J. Yajima, \ AES3papers-4-25.
[7] “Performance Evaluation of AES Finalists on the High-End Smart Card”, F. Sano, M. Koike, S. Kawamura, M.Shiba, AES3papers-2-14.
[6] “Randomness Testing \ of the Advanced Encryption Standar Finalist Candidates”, J. Soto, L. Bassham, AES3papers-4-30.
[5] “Relationships among Differential, Truncated Differential Impossible Differential Cryptanalyses against Word-Oriented Block Ciphers Like Rijndael E2”, M.Sugita, K.Kobara, K. Uehara, S.Kubota, H. Imai, AES3papers-5-32.
[4] “Realization of the Round 2 AES Candidates using Altera FPGA”, V. Fischer, AES3papers-4-24.
[3] “The AES Winner”, D.R. Patel, AES3papers-3-19.
[2] “The Power of NIST’s Statistical Testing of AES Candidates”, S. Murphy, AES3papers-2-09.

1999

[1] "AES Proposal: Rijndael", J.Daemen, V. Rijmen, proposal to AES of NIST.

RFCs

RFC 3268: Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS).
RFC 3394: Advanced Encryption Standard (AES) Key Wrap Algorithm.
RFC 3537: Wrapping a Hashed Message Authentication Code (HMAC) Key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key.
RFC 3565: Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)
RFC 3566: The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec.
RFC 3602: The AES-CBC Cipher Algorithm and Its Use with IPsec.
RFC 3664: The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)
RFC 3686: Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)
RFC 3826: The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based- Security Model.
RFC 3853: S/MIME Advanced Encryption Standard (AES), Requirement for the Session Initiation Protocol (SIP)
RFC 3962: Advanced Encryption Standard (AES) Encryption for Kerberos 5.

