

Advanced Encryption Standard

El AES es el cifrado simétrico más usado actualmente (2010), e incluso si el "estándar" es nado más en EUA, se usa ampliamente en otras industrias y sistemas comerciales (Estándares de seguridad en Internet IPSec, TLS IEEE 802.11i, etc.).

Introducción

En 1999 el Instituto Nacional de Estándares y Tecnología de EUA (NIST) dijo que el DES ya nado más para lo viejito, y que hay que mejor usar el 3DES.

- ¡Problema! • El DES y el 3DES son MUY lentos en implementación por software.
- También, el tamaño de la llave es muy chico para ciertas aplicaciones (funciones hash).

Entonces el NIST hizo un llamado a nuevos estándares.

- 1997 (enero) NIST hace el llamado.
- 1997 (septiembre) Idem (pero formalmente)

Requerimientos:

- Cifrado de bloque con tamaño de bloques de 128 bits.
- Tamaño de llaves variable de 128, 192 y 256 bits.
- Seguridad relativa a otros algoritmos sometidos.
- Eficiencia en hardware y software

- 1998 (agosto) 15 candidatos

- 1999 (agosto) 5 finalistas:

- Mars (IBM)
- RC6 (RSA)
- Rijndael (Joan Daemen y Vincent Rijmen)
- Serpent (Anderson, Biham y Knudsen)
- Twofish (Schnaeier, Kelsey, Whiting, Wagner, Hall, Ferguson)

- 2000 (octubre) Rijndael es coronado AES.

- 2001 (noviembre) AES es aprobado como estándar federal estadounidense.

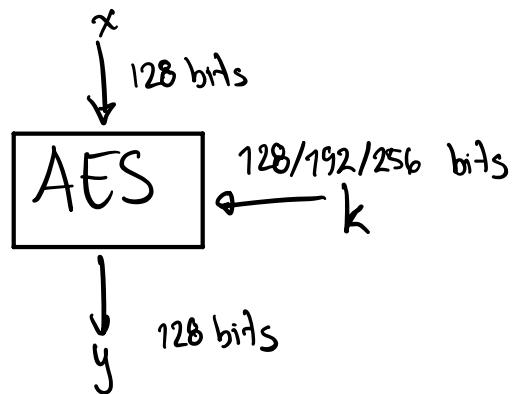
- 2003 La NSA lo declara apto para secretos de estado

para todo tipo de llaves para secreto de nivel normal y de llaves de 192 y 256 bits para secretos de alto nivel.

Vistazo rápido de AES

AES casi no cambió con respecto a Rijndael, aunque Rijndael acepta llaves de 128, 192 y 256 bits, mientras que AES oficialmente es de 128 bits. Esto último es lo que discutiremos.

Parámetros de AES:



El número de "rondas" por iteración depende de la longitud de llaves ($128 \mapsto 10, 192 \mapsto 12, 256 \mapsto 14$)

A diferencia de DES, AES cifra todos los 128 bits de un jalón por iteración

Como tal, AES consiste de 3 capas (por ronda... así) las cuales manipulan los 128 bits por capa:

(Capa de adición de llave: se suma XORicamente una llave/subllave de 128 bits.

(Capa de sustitución de bytes: cada elemento del estado (de los datos) es transformado no linealmente usando tablas. Esto agrega confusión (cambios individuales se propagan)

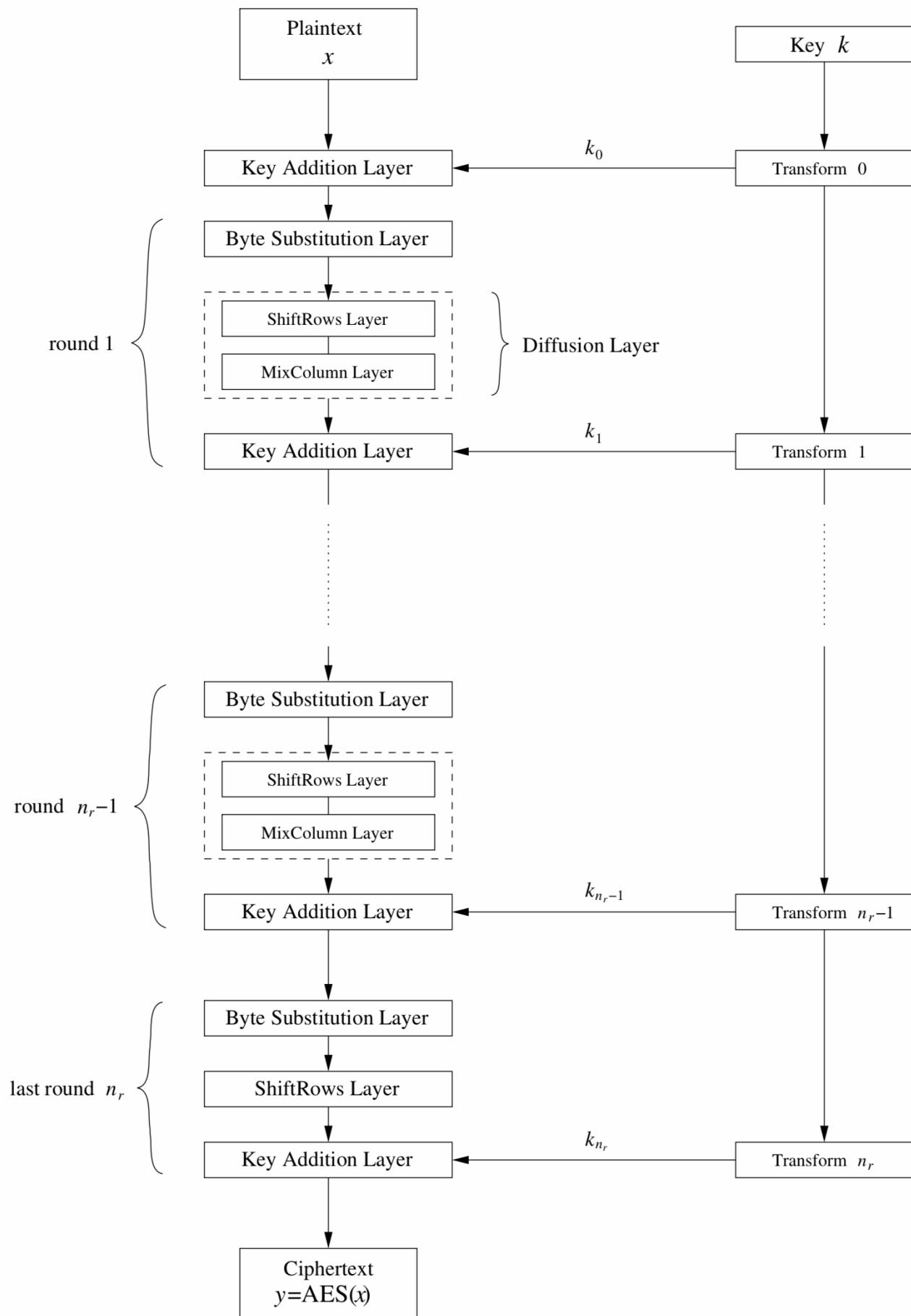
(Capa de difusión: consiste de dos subcapas:

—Subcapa de desplazamiento de columnas

(Shift Rows): permuta los datos a nivel bytes.

—Subcapa de mezclado de columnas

(Mix Column): es una operación matricial que combina/mezcla bloques de 4 bytes.

**Fig. 4.2** AES encryption block diagram

Conceptos Matemáticos

Grupos ✓

Campos ✓

Teorema 4.3.1 Un campo de orden m existe sólo si m es una potencia de un primo.

Denotamos por $GF(p)$ al campo de orden p . Como $GF(p)$ tiene tablas de operaciones explícitas y finitas, estas pueden usarse para facilitar/agilizar procesos.

La tabla de suma en $GF(2)$ es XOR.

La tabla de multiplicación en $GF(2)$ es AND.

En AES se usa $GF(2^8)$. Esto es especial porque cada elemento de $GF(2^8)$ se puede representar usando un solo byte.

Para la S-Box y las transformaciones MixColumn, AES usa cada byte de los datos internos como un elemento de

$GF(2^8)$ y usa la aritmética de $GF(2^8)$ para manipular los datos.

(Como 2^8 no es primo, $GF(2^8)$ es una extensión de campo, por lo que sus elementos los trataremos como polinomios en $GF(2)$ de grado a lo más 7 (\Rightarrow hay 8 coeficientes, no todos no cero, en cada elemento)):

$$\forall A \in GF(2^8) \quad A(x) = a_7x^7 + \dots + a_1x + a_0 \quad (\text{con } a_i \in GF(2))$$

$$\Rightarrow GF(2^8) \xrightarrow{\psi} GF(2)^8$$

ψ es un isomorfismo de grupos abelianos.

\Rightarrow la adición en $GF(2^8)$ es hacer XOR en cada entrada del byte.

Para la multiplicación, lo que hacemos es multiplicar los polinomios y reducir módulo el polinomio irreducible

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

Por ejemplo: $x^8 \equiv 1 \cdot P(x) + (x^4 + x^3 + x + 1) \pmod{P(x)}$

$$\Rightarrow x^8 \equiv x^4 + x^3 + x + 1 \pmod{P(x)}$$

$$\text{Sean } f_1(x) = x^7 + x^5 + x^3 + 1 \quad , \quad f_2(x) = x^4 + x^2 + 1$$

$$\Rightarrow f_1(x) \cdot f_2(x) = (x^7 + x^5 + x^3 + 1)(x^4 + x^2 + 1)$$

$$= x^{11} + x^9 + x^7 + x^4 \\ + x^9 + x^7 + x^5 + x^2 \\ + x^7 + x^5 + x^3 + 1$$

$$= x^{11} + x^7 + x^4 + x^3 + x^2 + 1$$

$$= x^3 x^8 + \dots + 1$$

$$= x^3 (x^4 + x^3 + x + 1) + x^7 + \dots + 1$$

$$= x^7 + x^6 + x^4 + x^3 \\ + x^7 + x^4 + x^3 + x^2 + 1$$

$$= x^6 + x^2 + 1$$

$$\Rightarrow (1, 0, 1, 0, 1, 0, 0, 1) \cdot (0, 0, 0, 1, 0, 1, 0, 1)$$

$$\begin{matrix} & & & & & & 11 \\ & & & & & & \\ (0, 1, 0, 0, 0, 1, 0, 1) \end{matrix}$$

Para la inversión de elementos, se usan tablas prefabricadas. En particular, para $\text{GF}(2^8)$ se usa notación hexadecimal y la tabla siguiente

Byte Substitution

Table 4.2 Multiplicative inverse table in $GF(2^8)$ for bytes xy used within the AES S-Box

	Y																
	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	
X	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

Example 4.7. From Table 4.2 the inverse of

Por ejemplo:

$$x^7 + x^6 + x \rightsquigarrow (1, 1, 0, 0, 0, 0, 1, 0)$$

$$\rightsquigarrow 11000010 \text{ base 2}$$

↓

$$194$$

base 10

$$(128 + 64 + 2)$$

↓

$$C2$$

base 16

$$\Rightarrow (C2)^{-1} = 2F \rightsquigarrow 47 \text{ base 10} \rightsquigarrow 101111 \rightsquigarrow (0, 0, 1, 0, 1, 1, 1, 1)$$

$$\Rightarrow (x^7 + x^6 + x)^{-1} = x^5 + x^3 + x^2 + x + 1$$

Y lo podemos verificar:

$$(x^7 + x^6 + x) (x^5 + x^3 + x^2 + x + 1) =$$

$$\begin{aligned} & x^{12} + x^{10} + x^9 + x^8 + x^7 \\ & + x^9 + x^8 + x^7 + x^6 + x^6 \\ & + x^6 + x^4 + x^3 + x^2 + x \end{aligned}$$

$$\begin{aligned} & x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x \\ & \quad \vdots \end{aligned}$$

$$x^4(x^4 + x^3 + x + 1) + x^3(x^4 + x^3 + x + 1) + x^2(x^4 + x^3 + x + 1) + x^4 + x^3 + x^2 + x$$

$$\begin{aligned} & x^8 + x^7 + x^5 + x^4 \\ & + x^7 + x^4 + x^6 + x^3 \\ & + x^5 + x^6 + x^3 + x^2 \\ & + x^4 + x^3 + x^2 + x \end{aligned}$$

$$\begin{aligned} & x^8 + x^4 + x^3 + x \\ & (x^4 + x^3 + x + 1) + x^4 + x^3 + x = 1 \end{aligned}$$

Dicho esto, hay ocasiones en las que se codifica el proceso de buscar inversas, en vez de usar tablas.

Estructura interna de AES

Dada una cadena de datos de 128 bits, lo primero que se hace es dividir esto en 16 bytes, obteniendo así una cadena $(A_0, A_1, \dots, A_{15}) \in GF(2^8)^{16}$.

El funcionamiento de una ronda es entonces así:

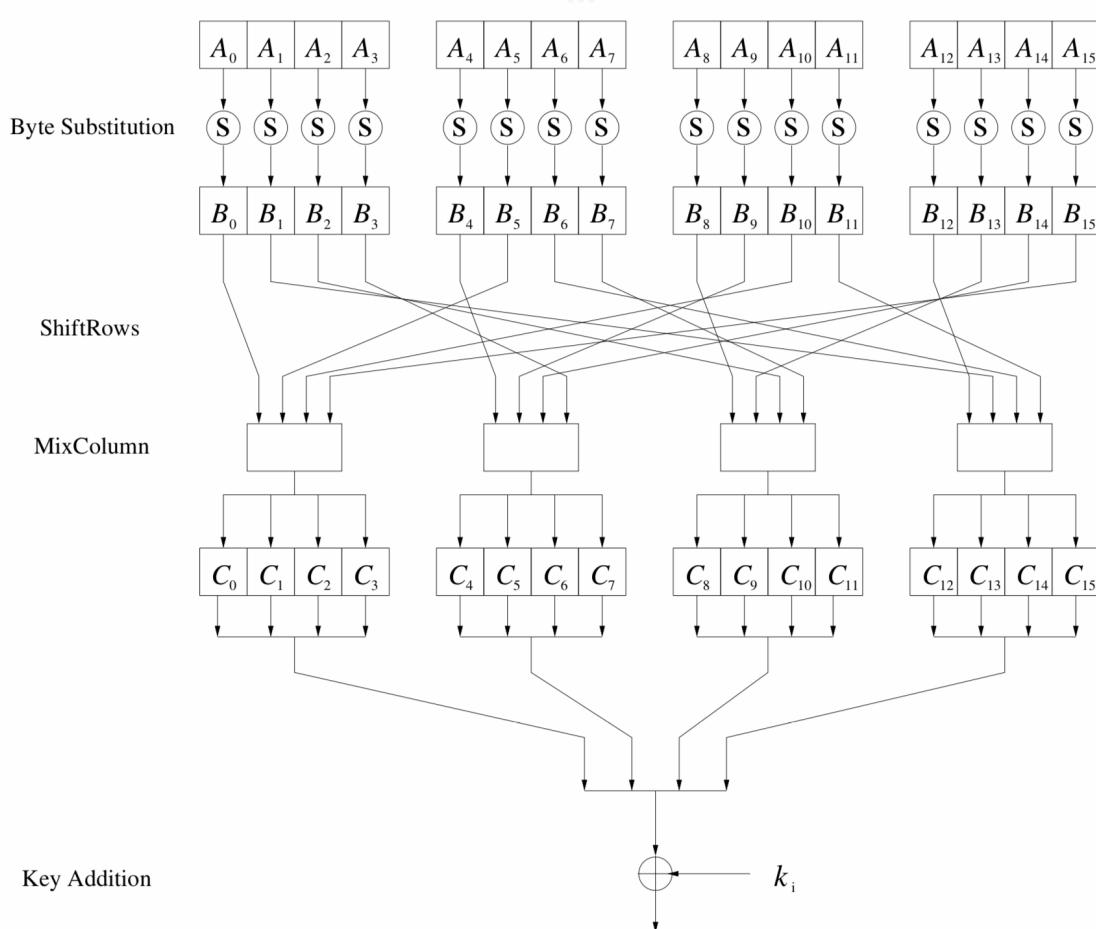


Fig. 4.3 AES round function for rounds $1, 2, \dots, n_r - 1$

Algo que ha sido útil en AES es arcomodar el estado de datos en matrices:

$$(A_0, A_1, \dots, A_{15}) \rightarrow \begin{pmatrix} A_0 & A_4 & A_8 & A_{12} \\ A_1 & A_5 & A_9 & A_{13} \\ A_2 & A_6 & A_{10} & A_{14} \\ A_3 & A_7 & A_{11} & A_{15} \end{pmatrix}$$

La llave de igual forma la ponemos en bytes y luego en matrices de tamaño 4×4 (128 bits), 4×6 (192 bits) o 4×8 (256 bits).

Capa de sustitución

Para la capa de sustitución, en implementaciones de software se usa una función $S(A_i) = B_i$ donde el resultado se busca en una tabla como esta:

AES S-Box: Substitution values in hexadecimal notation for input byte (xy)

		y																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
x		0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
x		1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
x		2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
x		3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
x		4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
x		5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
x		6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
x		7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
x		8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
x		9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
x		A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
x		B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
x		C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
x		D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
x		E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
x		F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Matemáticamente S es $S(A_i) = F(A_i')$ donde F es una transformación afín

Ejemplo:

$$11000010 \rightsquigarrow 00101111$$

$$\begin{matrix} \\ \\ C_2 \end{matrix}$$

$$\begin{matrix} \\ \\ 2F = (C_2)^{-1} \end{matrix}$$

$$T \in GL(8, \mathbb{Z}/2\mathbb{Z}), \quad v \in GF(2^8)$$

$$S(C_2) = T \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + v$$

Observación: Teniendo el byte con notación binaria, para pasarlo a vector se lee de derecha a izquierda.

Como se usa inversión del campo se rompe la linealidad, haciendo no efectivas muchas estrategias de criptoanálisis.

Como se usa una transformación afín tampoco se puede sacar ventaja de la inversión del campo.

Capa de difusión

Esta capa se divide en dos, una subcapa es una permutación de las filas, mientras que la otra subcapa es la aplicación de una transformación lineal a las columnas.

Shift Rows

Tras la capa de sustitución la matriz quedó

$$\begin{pmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{pmatrix}$$

En Shift Rows se aplican permutaciones a cada fila.

Fila 1 ~ id Fila 2 ~ (1 4 3 2)

Fila 3 ~ (1 3)(4 2) Fila 4 ~ (1 2 3 4)

Esto nos deja la matriz estando como:

$$\begin{pmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{pmatrix}$$

Mix Column

Aquí se toma una matriz $M \in GL(4, GF(2^8))$ y se fija. Luego se toma cada columna de la última matriz es- tado y se le aplica M

$$M \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix} = \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}, \quad M \cdot \begin{pmatrix} B_4 \\ B_9 \\ B_{14} \\ B_3 \end{pmatrix} = \begin{pmatrix} C_4 \\ C_5 \\ C_6 \\ C_7 \end{pmatrix},$$

$$M \cdot \begin{pmatrix} B_8 \\ B_{13} \\ B_2 \\ B_7 \end{pmatrix} = \begin{pmatrix} C_8 \\ C_9 \\ C_{10} \\ C_{11} \end{pmatrix}, \quad M \cdot \begin{pmatrix} B_{12} \\ B_1 \\ B_6 \\ B_{11} \end{pmatrix} = \begin{pmatrix} C_{12} \\ C_{13} \\ C_{14} \\ C_{15} \end{pmatrix}.$$

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

La nueva matriz de estado es:

$$\begin{pmatrix} C_0 & C_4 & C_8 & C_{12} \\ C_1 & C_5 & C_9 & C_{13} \\ C_2 & C_6 & C_{10} & C_{14} \\ C_3 & C_7 & C_{11} & C_{15} \end{pmatrix}$$

Capa de adición de llave

La matriz estado se regresa a sucesión de bits y se le suma (XORea) la subllave de 128 bits.

La subllave se obtiene de la llave original en el proceso de key schedule.

Key schedule

Para la generación de subllaves el proceso varía un poco dependiendo de la longitud de la llave. De igual forma, el número de rondas también depende de esto.

Recordando la tabla del funcionamiento del AES, siempre se empieza con una adición de subllave y ya después se empiezan las rondas (cada una con una subllave).

Por lo tanto, si se usan n_r rondas, se necesitan $n_r + 1$ subllaves.

Por cierto, la adición de subllave antes de las rondas

se llama key whitening / blanqueado de llave.

Como tal, el proceso de obtención de llave se realiza usando palabras (32 bits = 4 bytes). Las palabras se van guardando porque se obtienen de forma iterativa.

$$l(K) = 128 \text{ bits}$$

$n_r = 10 \Rightarrow$ hay 11 subllaves.

k_0 en este caso no es otra cosa mas que K , pero se divide en bytes y luego esos bytes se agrupan en palabras:

$$K \rightsquigarrow (k_0, k_1, \dots, k_{15})$$

$$\rightsquigarrow W[0] = (k_0, k_1, k_2, k_3)$$

$$W[1] = (k_4, k_5, k_6, k_7)$$

$$W[2] = (k_8, k_9, k_{10}, k_{11})$$

$$W[3] = (k_{12}, k_{13}, k_{14}, k_{15})$$

$$\Rightarrow k_0 = (W[0], W[1], W[2], W[3])$$

Ahora, para $i = 1, \dots, 10$, obtenemos $W[4:i]$ como sigue:

$$W[4i] = W[4i-4] \oplus g(W[4i-1])$$

dónde g es una función no lineal que definiremos más abajo.

Posteriormente tenemos para $j=1, 2, 3$:

$$W[4i+j] = W[4i+j-4] \oplus W[4i+j-1]$$

Dejando así $k_i := (W[4i], W[4i+1], W[4i+2], W[4i+3])$.

Finalmente, para definir g primero definimos $RC(i)$, llamada la constante de i -ésima ronda, como el elemento de $GF(2^8)$ correspondiente al polinomio x^i

$$RC(1) = 00000010, \quad RC(2) = 00000100,$$

$$RC(3) = 00001000, \dots, \quad RC(7) = 10000000,$$

$$RC(8) = 00011011, \quad RC(9) = 00110110$$

Luego, g de una palabra se obtiene primero agrupando en bytes, luego permutando cíclicamente (1 a la izquierda), de ahí aplicando S (de la capa de permutación), luego al byte de la extremidad izquierda se le suma $RC(i)$ y finalmente se agrupan los bytes en una palabra.

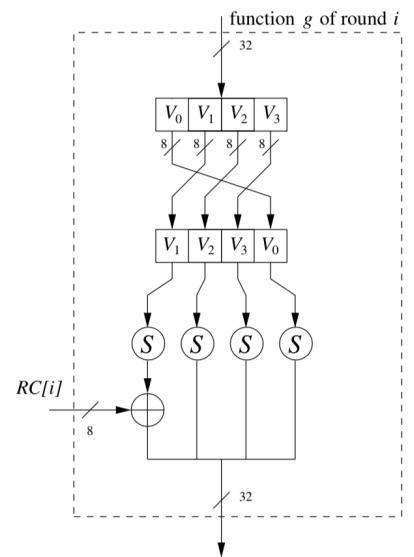
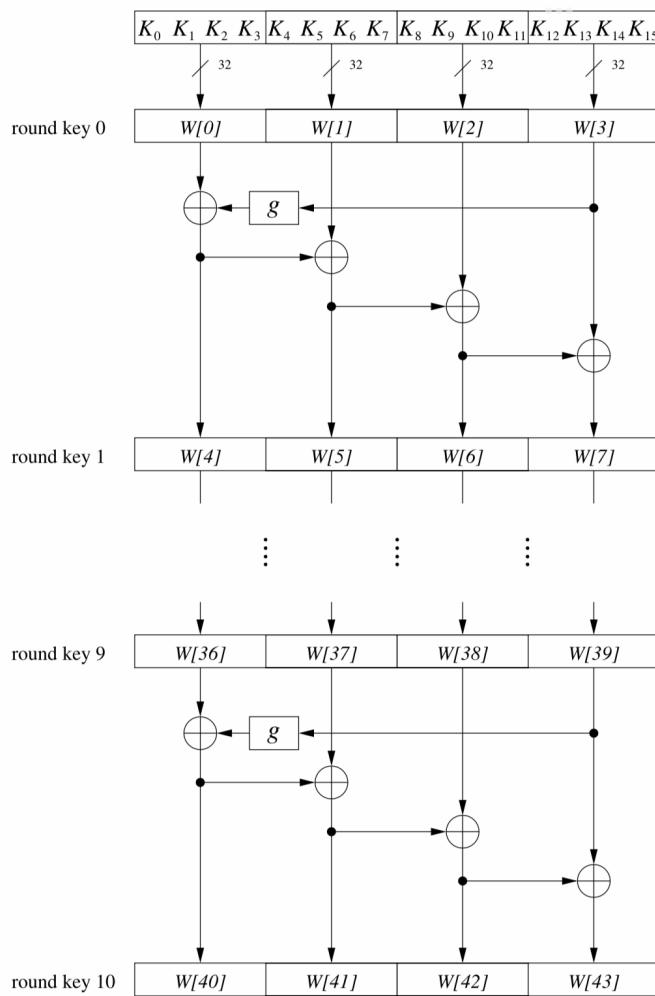


Fig. 4.5 AES key schedule for 128-bit key size

La función g sirve para remover linealidad y simetría.

$$l(K) = 192 \text{ bits}$$

Para este caso $n_r = 12 \Rightarrow$ se necesitan 13 llaves \Rightarrow
 \Rightarrow se necesitan 52 palabras.

El proceso es casi el mismo que para 128 bits pero se hace módulo 6, en vez de módulo 4.

Una vez generadas todas las palabras, se acomodan en grupos de 4 para obtener las llaves.

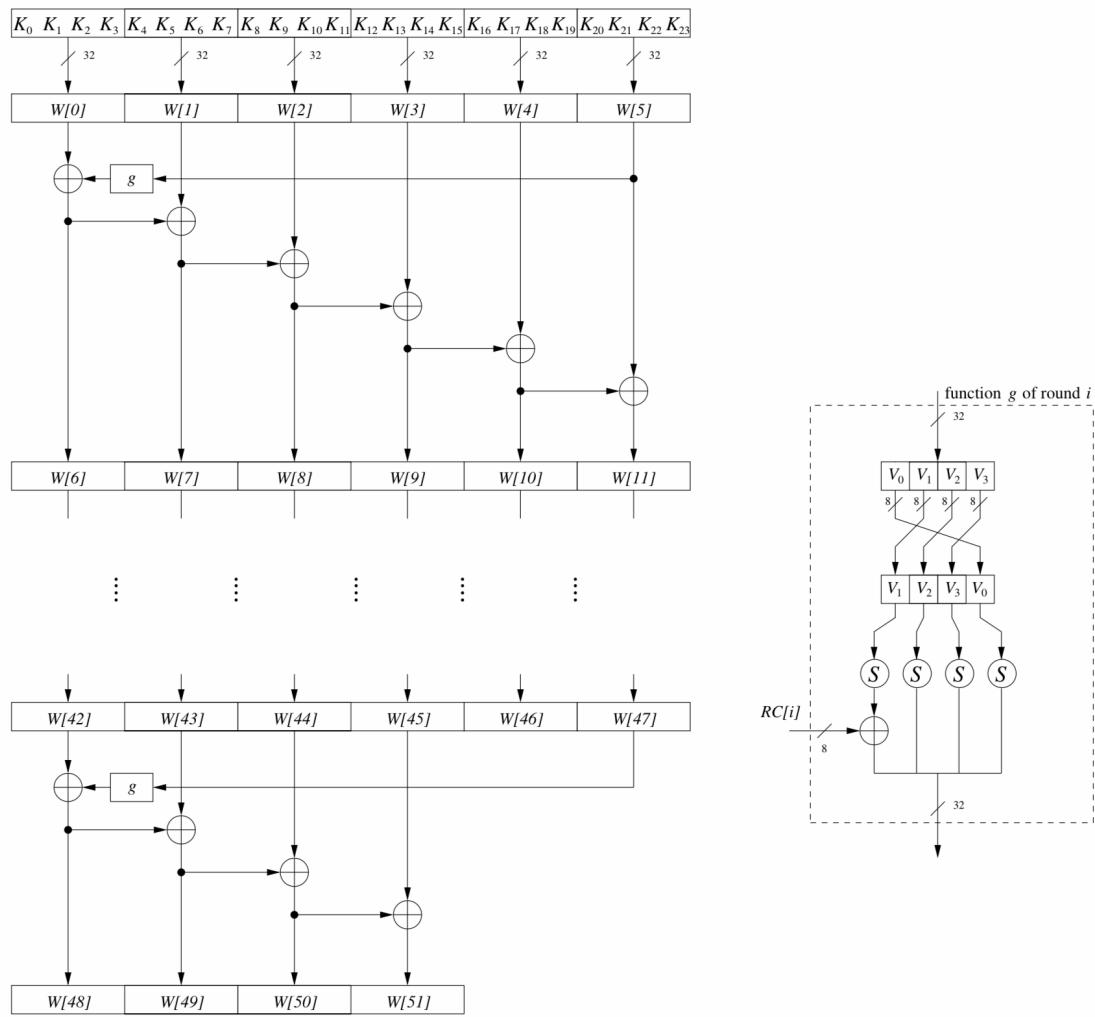


Fig. 4.6 AES key schedule for 1024 bit key sizes

$$l(k) = 256 \text{ bits}$$

Aquí tenemos $n_r = 14 \Rightarrow$ Hay 15 subllaves $\Rightarrow 60$ palabras.

El proceso es casi el mismo que los anteriores pero módulo 8, aunque se agrega una función h (que es aplicación de S a cada byte) cada palabra congruente con 4 módulo 8

Una vez generadas todas las palabras, se acomodan en grupos de 4 para obtener las llaves.

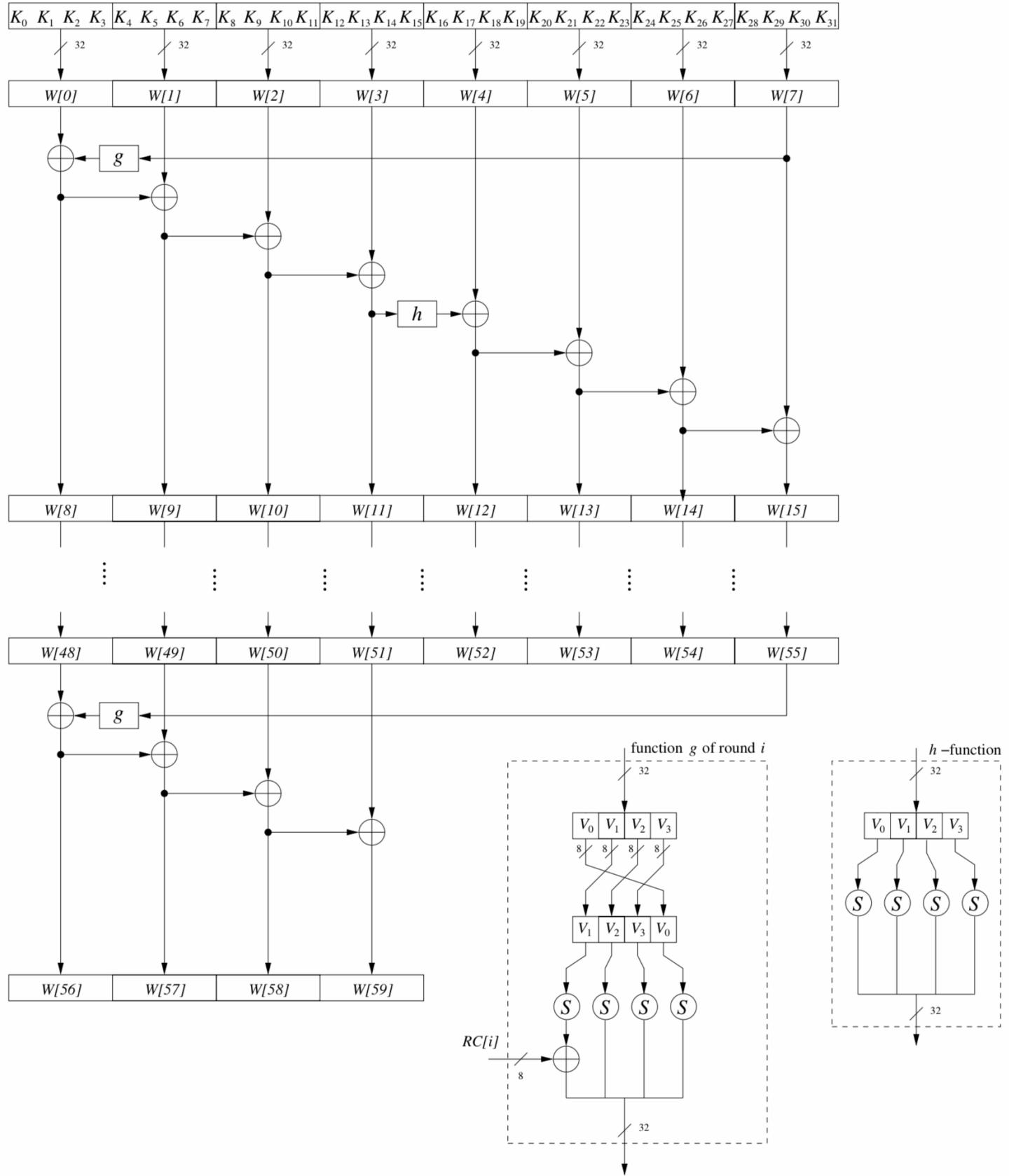


Fig. 4.7 AES key schedule for 256-bit key size

Producción de palabras

Hay en general 2 estrategias para la producción de las palabras:

1) Computación previa: Antes de cifrar o descifrar se computan todas las palabras, se crean las subllaves y se guardan. Esto hace que se necesite una memoria dedicada para esto de al menos $(n_r+1) \cdot 16$ bytes. Esto hace que esta estrategia no sea popular en implementaciones de hardware con limitaciones en memoria.

Esta estrategia se usa cuando una PC o un servidor va a estar cifrando y descifrando con la misma llave varias veces.

2) Al momento: Como lo dice el nombre, las palabras se calculan al momento y conforme se van necesitando para formar las llaves.

Como para el descifrado se necesita la última subllave para empezar, esto hace que para descifrar se calculen todas primero y luego se empieza $\Rightarrow t(\text{Descifrado}) > t(\text{Cifrado})$.