SEGURIDAD EUROPEA PARA EEUU.

Algoritmo Criptografico Rijndael

Autor: **Alfonso Muñoz Muñoz.**
Sept-2004. Madrid.

Agradecimientos.

Gracias a Kriptopolis por la distribución de este trabajo.

Gracias a todo lector que se sienta necesitado de leer esta información.

Prefacio.

Todos los hombres hemos sido creados iguales,
entendemos que es una verdad evidente.

Nuestro Creador nos ha dotado de derechos inalienables,
entre los que se encuentra la vida, la libertad y la búsqueda
de la felicidad.

Pensamos que para asegurar estos derechos se han instituido
entre los hombres los gobiernos, cuyo poder debe estar
supeditado al consentimiento de los gobernados.
Y siempre y cuando la forma de gobierno perjudica estos
fines, el pueblo tiene la obligación de incitar a la desobediencia
civil, abolir el sistema e instituir uno nuevo.

¿Comprenden lo que digo?. Hasta que no tengan conciencia
de su fuerza, no se rebelarán y hasta después de haberse rebelado, no serán
conscientes. ¿Comprenden el problema?.

Comprendan el "CÓMO" y el "PORQUE". Si concede ésto, lo
demás vendrá por sus pasos contados.

Alfonso Muñoz.
Madrid, Septiembre 2004

INDICE.

1. ¿Por qué este trabajo?.....5
2. Algoritmo AES (Advanced Encryption Standard).....6
2.1. Historia del algoritmo. 6
2.1.1 Concurso Público AES. 6
2.1.1.1 Primera Ronda. 8
2.1.1.2 Segunda Ronda. 9
2.1.1.3 Tercera Ronda. 9
2.1.2 Comparación de los Algoritmos Finalistas. 10
2.2. Algoritmo Rijndael. 12
2.2.1 Conceptos Matemáticos Preliminares. 12
2.2.2 Estructura del Algoritmo 19
2.2.3 Descripción del proceso de cifrado. 22
2.2.3.1. Función ByteSub. 23
2.2.3.2. Función ShiftRow. 26
2.2.3.3. Función MixColumn. 27
2.2.3.4. Función AddRoundKey. 30
2.2.3.4.1. Función de selección de clave. 32
2.2.3.4.2. Función de expansión de clave. 32
2.2.4 Descripción del proceso de Descifrado. 36
2.2.5. Motivos de Diseño. 37
2.2.5.1. Estructura del Algoritmo 37
2.2.5.2. Función ByteSub. 39
2.2.5.3. Desplazamientos en la función ShitfRow. 40
2.2.5.4. Función MixColumn. 40
2.2.5.5. Número de Vueltas. 40
2.2.5.6. Expansión de Clave 42
2.2.6. Seguridad del Algoritmo. 44
2.2.6.1. Ataques Algebraicos. 45
3. Tendencias y Opinión del Autor. 47

1. ¿Por qué este trabajo?

El presente documento resulta de los estudios personales sobre algoritmos criptográficos, y en concreto del algoritmo criptográfico Rinjdael, algoritmo que constituye el nuevo estándar de cifrado AES.

Puesto que AES es el sustituto del clásico algoritmo DES, y ya que la mayoría de las comunicaciones aplicarán este algoritmo criptográfico veo necesario su estudio, destacando sus multiples virtudes, y porque no los estudios de criptoanálisis que se van realizando.

La fuente fundamental utilizada para realizar este trabajo, por lo menos en cuanto a la descripción del algoritmo, ha sido ***"The Design of Rijndael. AES- The Advanced Encryption Standard"*** de los creadores de dicho algoritmo, Joan Daemen y Vicent Rijmen.

Espero que el documento, sirva de ayuda al lector, al igual que me lo sirvió a mi al realizarlo. Para cualquier duda, o mejora: ***sito.handler@gmail.com***

"*Nuestra causa es un secreto dentro de otro secreto, el secreto de algo que permanece velado, un secreto que sólo otro secreto puede explicar, es un secreto sobre un secreto que se satisface con otro secreto*"

2. Algoritmo AES (Advanced Encryption Standard).

2.1. Historia del algoritmo.

El 23 de Noviembre de 1976 se establece el primer estándar de cifrado para comunicaciones, el denominado **algoritmo DES**. Desde entonces se han descrito multitud de ataques que permiten criptoanalizarlo más rápidamente que con un ataque por fuerza bruta. Pero sin duda la publicación de la asociación EFF (*Electronic Frontier Foundation*) del diseño de una máquina que permite atacarlo por fuerza bruta en un tiempo ínfimo (DES-CRACKER), puso el grito en el cielo de la comunidad científica. El estándar había consumido su tiempo de vida.

En el año 1997, el Instituto Nacional de Estándares y Tecnología de EEUU (NIST), emprende un proceso abierto para la selección de un nuevo algoritmo de cifrado, que sustituya al actual estándar de cifrado, criticado por especialistas e instituciones en seguridad.

Este nuevo algoritmo sería útil no sólo para proteger la información del Gobierno EEUU, sino que también sería utilizado masivamente por el sector privado, y adoptado como estandar por el resto de países, entre ellos los Europeos.

2.1.1 Concurso Público AES.

Para evitar las quejas que se procedieron con la implantación del algoritmo DES debido a partes del algoritmo no documentadas que daban la sensación que el gobierno EEUU mantenía puertas traseras, se decide iniciar un proceso abierto para seleccionar el algoritmo que formaría el nuevo estándar de cifrado AES.

Se inicia entonces los primeros pasos para la consolidación de un **Estándar de Cifrado Avanzado (AES)** que permita proteger los datos confidenciales del gobierno, así como la información sensible de los ciudadanos.

En Septiembre de 1997 se presentan los criterios de evaluación y requisitos mínimos que debían cumplir todos los algoritmos que optarán a ganar el concurso, entre ellos destacaban:

- El algoritmo debe ser público.
- Debe ser un algoritmo de cifrado en bloque simétrico.
- La longitud de la clave debe ser como mínimo 128 bits.
- Su diseño debe permitir aumentar la longitud de la clave según las necesidades.
- Debe ser implementable tanto en HW como en SW

Los algoritmos que cumplieran los requisitos anteriores serían juzgados por los siguientes factores:

- Seguridad.
- Eficiencia computacional.
- Requisitos de memoria.
- Implementable en HW y SW.
- Simplicidad de diseño.
- Flexibilidad.

Los algoritmos que se presentaron a este concurso además tenían que soportar obligatoriamente una longitud de bloque de 128 bits como mínimo, y una longitud de clave de 128, 192 y 256 bits, al margen de cualesquiera otras longitudes posibles.

NIST, propuso que cualquier organización, institución o persona pudiera participar de forma activa en este concurso, ya fuera presentando algoritmos, o enviando informes o pruebas de cualquier tipo para poner en evidencia las características de cualquier de los algoritmos candidatos.

La intención de este estándar es que sea robusto, por lo menos, hasta la mitad del presente siglo, o por lo menos hasta que se publiquen estudios criptoanaliticos o incluso posibles máquinas futuras de supercomputación, como la soñada computación cuántica, que debilite seriamente su seguridad.

Para llevar a cabo la elección del algoritmo se propuso crear dos rondas de selección. En la primera ronda se seleccionaría los 5 algoritmos mejores, que cumplieron las especificaciones iniciales, y en la segunda ronda se decidiría el algoritmo ó algoritmos ganadores.

Para realizar estas rondas de selección, se convocaron tres Conferencias, en distintos lugares del mundo, en las que los algoritmos candidatos, pudieron ser probados, comentados y revisados con lupa por todo el mundo que lo deseó.

Durante todo el desarrollo del proceso AES, **todos los algoritmos y criterios de diseño estuvieron disponibles de forma pública y abierta**, por lo que el escrutinio al que han sido sometidos todos los finalistas ha sido enorme, acorde con la importancia del nuevo AES. Todos los participantes contribuyeron al proceso, analizando las posibles vulnerabilidades de sus competidores.

2.1.1.1 Primera Ronda.

En Agosto de 1998 comenzó la primera ronda aceptándose quince candidatos durante la "***Primera Conferencia de candidatos a AES***". Los quince algoritmos candidatos fueron:

- *CAST-256 [Entust Technologies, Inc. C.Adams]*
- *CRYPTION. [Future Systems, Inc. Chae Hoon Lim]*
- *DEAL. [L.Knudsen, R. Outerbridge]*
- *DFC. [CNRS-Ecole Normale Superiere. S.Vaudenay]*
- *E2. [NTT Nipón Telegraph and Telephone Corporation. M.Kanda].*
- *FROG. [TecApro International S.A. (D. Georgoudis, Leroux, Chaves]*
- *HPC. [R. Schoeppel]*
- *LOKI97. [L.Brown, J.Pieprzyk, J.Seberry]*
- *MAGENTA [Deutshe Telekom A.G. K, Huber]*
- *MARS. [IBM. Nevenki Zunic]*
- *RC6. [RSA Laboratories. Rivest, M.Robshaw, Sidney, Yin]*
- *RIJNDAEL [Joan Daemen, Vicent Rijmen]*
- *SAFER+. [Cylink Corporation. L.Chen]*
- *SERPENT. [R.Anderson, E.Biham, L.Knudsen]*
- *TWOFISH[B.Schneier,J.Kelsey,D.Whiting, D.Wagner,C.may,N.Ferguson]*

2.1.1.2 Segunda Ronda.

En Marzo de 1999, se celebraría ***la segunda conferencia de Candidatos a AES***, en la que se discutió los resultados de las numerosas pruebas y criptoanálisis realizados por la comunidad criptográfica mundial sobre los quince candidatos iniciales. Basándose en estos comentarios y análisis, NIST seleccionó cinco candidatos finalistas. Cabe decir como anécdota que uno de los quince algoritmos seleccionados, concretamente "Magenta", fue criptoanalizado en el mismo encuentro en el que se presentó. Los cinco algoritmos afortunados fueron:

→ **MARS.** [IBM. Nevenki Zunic]
→ **RC6.** [RSA Laboratories. Rivest, M.Robshaw, Sidney, Yin]
→ **RIJNDAEL** [Joan Daemen, Vicent Rijmen]
→ **SERPENT.** [R.Anderson, E.Biham, L.Knudsen]
→ **TWOFISH** [B.Schneier, J.Kelsey, D.Whiting, D.Wagner, C.may, N.Ferguson]

2.1.1.3 Tercera Ronda.

En abril del 2000 se celebró la "***Tercera Conferencia de Candidatos AES***" en Nueva York, durante la cual los asistentes presentaron nuevos documentos de evaluación y criptoanálisis de los últimos cinco candidatos. Varios de los algoritmos recibieron un varapalo criptográfico. RC6 resultó el más afectado: dos grupos se las ingeniaron para romper 15 de 20 ciclos del algoritmo más rápidamente que con fuerza bruta. RIJNDAEL resistió algo mejor: 7 ciclos rotos de 10/12/14 ciclos. Se presentaron varios ataques contra MARS; el más interesante rompió 11 de 16 ciclos del núcleo criptográfico. SERPENT y TWOFISH se comportaron mejor: el ataque más fuerte contra SERPENT rompió 9 de 32 ciclos, y no se presentaron nuevos ataques contra TWOFISH.

Por fin, el 2 de octubre de 2000, el NIST anunció el algoritmo ganador. Las votaciones del concurso establecieron el siguiente ranking:

RIJNDAEL → 86 votos
SERPENT → 59 votos
TWOFISH → 31 votos
RC6 → 23 votos
MARS → 13 votos

El algoritmo Rijndael ganó el concurso, por permitir la mejor combinación de seguridad-velocidad-eficiencia, sencillez y flexibilidad.

Destacando su sencillez, que había permitido un análisis muy intenso de su estructura.

Los creadores de este ingenio son dos ingenieros electrónicos belgas, un equipo bastante modesto, teniendo en cuenta que en el proceso de selección se enfrentaban a algoritmos creados por equipos de multinacionales tan fuertes y poderosas como IBM, Deuche Telekom, así como equipos de criptólogos de reputada fama mundial como, por ejemplo, Bruce Schneier (Twofish), autor de varios libros de criptografía, o Ronald Rivest (RC6), coautor del algoritmo de clave asimétrica RSA.

El algoritmo fue bautizado como Rijndael en un juego de fusión entre los apellidos de sus dos creadores: **Vincent Rijmen**, nacido en 1970, matemático de la facultad de Ciencias de la Universidad Católica de Lovaina, y su ex-colega en dicha universidad **Joan Daemen**, nacido en 1965, ingeniero electrónico y especialista en sistemas de seguridad electrónica bancaria.

Por tanto, en Octubre de 2000, quedó establecido el estándar AES (algoritmo Rijndael) como el estándar actual de comunicaciones de EEUU y por derivación de todo el mundo. Actualizándose paulatinamente todas las aplicaciones y servicios del estándar previo DES al nuevo sistema de cifrado.

2.1.2 Comparación de los Algoritmos Finalistas.

La seguridad de los algoritmos, fue el aspecto más importante que se tuvo en cuenta en la segunda ronda del proceso de selección del NIST, para probar la seguridad de cada algoritmo se lanzaron varios ataques a cada algoritmo, estos ataques han sido llevados a cabo, tanto por los autores de algoritmos rivales, como por cualquier otra persona que enviara la documentación pertinente al NIST.

En cualquiera de los casos, **según los ataques publicados**, los resultados obtenidos en cada algoritmo fueron:

- **MARS: parece tener un margen de seguridad elevado**. MARS recibió críticas basadas en su complejidad, la cual puede haber obstaculizado su análisis de seguridad durante el proceso de desarrollo del AES.

- **RC6: parece tener un margen de seguridad adecuado**. Sin embargo, RC6 recibió alguna crítica debido a su bajo margen de seguridad respecto al que ofrecieron otros finalistas. Por otro lado, RC6 ha sido elogiado por su simplicidad, la cual ha facilitado su análisis de seguridad durante el tiempo especificado en el proceso de desarrollo del AES.

- **RIJNDAEL: parece tener un margen de seguridad adecuado**. El margen de seguridad es un poco difícil de medir, debido a que el número de rondas cambia con el tamaño de la clave. ***Rijndael recibió críticas sobre su margen de seguridad, ya que es de los más bajos, entre los finalistas***, y que su estructura matemática puede conducir a ataques. Sin embargo, su estructura es bastante simple, esto ha facilitado su análisis de seguridad durante el tiempo especificado en el proceso de desarrollo del AES.

- **SERPENT: parece tener un margen de seguridad alto**. Serpent también tiene una estructura simple, que ha facilitado su análisis de seguridad durante el tiempo especificado de desarrollo del AES.

- **TWOFISH: parece tener un margen de seguridad alto**. El concepto de margen de seguridad tiene menos significado para este algoritmo que para los demás finalistas. La dependencia de las S-cajas de Twofish en solo K/2 bits de entropía en el caso de clave K-bits ha producido algunas especulaciones acerca de que Twofish puede ser sensible a un ataque divide y vencerás, aunque tal ataque no ha sido comprobado. Twofish ha recibido alguna crítica por su complejidad, haciendo difícil su análisis durante el tiempo establecido en el proceso de desarrollo del AES.

Otro tema muy importante fue el estudio de la velocidad con la que se ejecutaba cada algoritmo, por ejemplo, dependiendo del tamaño de la clave. La realización de SW de MARS, RC6 y SERPENT no varían significativamente para los tres tamaños de clave de AES. Para Rijndael y Twofish, sin embargo, la configuración de la clave es lógicamente, más lento para claves de 192 bits que para claves de 128 bits, y más lento todavía para claves de 256 bits, aunque en estos casos ofrecen una compensación en el incremento de la seguridad.

Rijndael se presentó como el algoritmo más rápido en multitud de plataformas: 32 bits, procesadores de 8bits, etc.

2.2. Algoritmo Rijndael.

Inicialmente se estudia su estructura y motivos de diseño para permitir cifrar y descifrar información. A continuación, analizaremos la seguridad que ofrece éste algoritmo y finalmente los ataques posibles documentados que puede sufrir.

2.2.1 Conceptos Matemáticos Preliminares.

El algoritmo Rijndael opera a nivel de bytes, interpretando estos como elementos de un cuerpo de Galois GF(2^8), y a nivel de registros de 32 bits, considerándolos como polinomios de grados menor que 4 con coeficientes que son a su vez polinomios en GF(2^8). En este apartado se van a definir las operaciones matemáticas básicas que necesita el algoritmo Rijndael, así como algunos conceptos referentes al tratamiento de polinomios

2.2.1.1 Cuerpos Finitos. GF(2^8)

En este algoritmo todos los bytes se interpretan como elementos de un cuerpo finito. Concretamente, se representan mediante ***Campos de Galois*** que se representan como GF(k) ("Galois Field").

Los campos de Galois son muy interesantes en criptografía, gracias a que existe un inverso aditivo y multiplicativo que permite cifrar y descifrar en el mismo cuerpo Z_k, eliminando así los problemas de redondeo o truncamiento de valores si tales operaciones de cifrado y descifrado se hubiesen realizado en aritmética real.

En nuestro caso, interesa utilizar una aritmética en módulo "p" sobre polinomios de grado "m", siendo "p" un número primo. Este campo de Galois queda representado como: GF(p^m), en donde los elementos de GF(p^m) se representan como polinomios con coeficientes en Z_p de grado menor que m, es decir:

$$GF(p^m)=\{ \lambda_0 + \lambda_1 x + \lambda_2 x^2 + \dots \lambda_{m-1} x^{m-1} ; \lambda_0, \lambda_1, \lambda_2 \dots \lambda_{m-1} \in Z_p\}$$

Cada elemento de GF(p^m) es un resto módulo p(x), donde p(x) es un polinomio irreducible de grado "m", esto es, que no puede ser factorizado en polinomios de grado menor que "m".

En el caso del algoritmo Rijndael, será interesante los campos del tipo $GF(2^m)$ puesto que los coeficientes en este caso serán los restos del módulo 2, es decir, 0 y 1, lo que permite una representación binaria. Por lo tanto, cada elemento del campo se representa con "m" bits y el número de elementos será 2^m.

Por ejemplo, para el campo $GF(2^3)$ sus elementos son: 0, 1, x, x+1, x^2 +1, x^2 +x, x^2 +x +1 que son precisamente todos los restos de un polinomio de grado "m-1".

En el caso del algoritmo Rijndael, se definen operaciones a nivel de byte, encontrándonos en el campo $GF(2^8)$.

Un byte "B", se compone de los bits b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0, si lo consideramos como un polinomio con coeficientes en {0,1} tenemos el polinomio:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Por ejemplo, un byte con el valor hexadecimal "57", en binario 01010111, corresponde con el polinomio:

$$x^6 + x^4 + x^2 + x + 1$$

2.2.1.2 Suma en $GF(2^8)$

En $GF(p^m)$ hay que considerar que las operaciones matemáticas sobre los coeficientes se hacen en módulo "p" con lo cual en $GF(2^m)$ se reducen los resultados de la suma de los coeficientes módulo 2. Este procedimiento tanto para la suma como para la resta se realiza simplemente con una operación Or-Exclusiva, ya que si los coeficientes son iguales darán como suma o resta un 0 y coeficientes distintos darán un 1.

Por tanto, para sumar dos polinomios, basta con aplicar la operación Or-Exclusiva, a cada elemento de los polinomios, dos a dos. La función Or-Exclusiva de dos bits, produce los siguientes resultados:

$$1 \oplus 1=0$$
$$0 \oplus 1=1$$
$$1 \oplus 0=1$$
$$1 \oplus 1=0$$

Un ejemplo de suma de dos polinomios de tamaño byte expresados dentro de GF(2^8) es:

$$A = 57_{16} = 0101\ 0111_2 = x^6 + x^4 + x^2 + x + 1$$
$$B = 83_{16} = 1000\ 0011_2 = x^7 + x + 1$$

Sumando:

$$A+B = (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) \bmod 2$$
$$A+B = (x^7 + x^6 + x^4 + x^2 + 2x + 2) \bmod 2 = x^7 + x^6 + x^4 + x^2 =$$
$$= 1101\ 0100 = D4_{16}$$

Y lo mismo se obtiene utilizando la operación Or-Exclusiva:

$$0101\ 0111 \oplus 1000\ 0011 = 1101\ 0100 = D4_{16}$$
$$\{57\} \oplus \{83\} = \{D4\}$$

2.2.1.3 Multiplicación en GF(2^8)

En la multiplicación de polinomios en GF(2^m), es posible que el resultado contenga elementos que estén fuera del cuerpo del polinomio (potencias iguales o mayores que ”m”) por lo que deberemos reducir los exponentes mediante un polinomio p(x) necesariamente irreducible y grado “m”.

Para GF(2^8) la multiplicación de polinomios se realiza modulo con un polinomio irreducible de grado 8. Este polinomio irreducible se representa por m(x) (es irreducible porque sus únicos divisores son el 1 y el mismo polinomio). El polinomio irreducible utilizado en el algoritmo Rijndael es:

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

Ejemplo de una multiplicación de polinomios: ‘57’ • ‘83’ = ‘C1’, esto es así porque:

$$A = 57_{16} = 0101\ 0111_2 = x^6 + x^4 + x^2 + x + 1$$
$$B = 83_{16} = 1000\ 0011_2 = x^7 + x + 1$$

$A \bullet B = (x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + 2x^7 + x^6 + x^5 + x^4 + x^3 + 2x^2 + 2x + 1 \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod x^8 + x^4 + x^3 + x + 1 = x^7 + x^6 + 1 =$
$= 1100\ 0001 = C1_{16}$

Otra forma de calcularlo sería razonando qué resultado de la multiplicación hay que reducirlo por m(x) para cada valor de x que esté fuera del cuerpo de 8 bits:

Sea $m(x) = x^8 + x^4 + x^3 + x + 1 \Rightarrow$ $\mathbf{x^8 = x^4 + x^3 + x + 1}$

$A \bullet B \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$x^{13} = x^5 * x^8 = x^5 * (x^4 + x^3 + x + 1) = x^9 + x^8 + x^6 + x^5 = x * x^8 + x^8 + x^6 + x^5 = x * (x^4 + x^3 + x + 1) + (x^4 + x^3 + x + 1) + x^6 + x^5 = (x^5 + x^4 + x^2 + x) + (x^4 + x^3 + x + 1) + x^6 + x^5$

$x^{13} = x^6 + x^3 + x^2 + 1$

$x^{11} = x^3 * x^8 = x^3 * (x^4 + x^3 + x + 1) = \mathbf{x^7 + x^6 + x^4 + x^3}$
$x^9 = x * x^8 = x * (x^4 + x^3 + x + 1) = \mathbf{x^5 + x^4 + x^2 + x}$

Por tanto, el resultado de multiplicar los polinomios, se obtiene sustituyendo:

$A \bullet B \bmod 2 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$
$A \bullet B \bmod 2 = (x^6 + x^3 + x^2 + 1) + (x^7 + x^6 + x^4 + x^3) + (x^5 + x^4 + x^2 + x) + (x^4 + x^3 + x + 1) + x^6 + x^5 + x^4 + x^3 + 1 \bmod 2$

$A \bullet B \bmod 2 = x^7 + x^6 + 1 = \mathbf{1100\ 0001 = C1_{16}}$

Como era lógico esperar, el polinomio resultante tendrá grado menor que 8. La multiplicación de polinomios es asociativa y su elemento neutro es el "01". Para cualquier polinomio binario b(x) de grado menor que 8, se puede aplicar el algoritmo extendido de Euclides para calcular un polinomio inverso de b(x). En este caso se habla de "inversa multiplicativa". a(x) es un polinomio inverso de b(x) si:

$$a(x) \bullet b(x) \bmod m(x) = 1 \text{ ó } \quad b^{-1}(x) = a(x) \bmod m(x)$$

es decir, a(x) es la inversa multiplicativa de b(x). El polinomio extendido de Euclides se puede ver como:

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

2.2.1.4 Multiplicación por x.

Se va a analizar un caso interesante, que es la multiplicación de un polinomio por x. Si multiplicamos un polinomio b(x) por x tenemos:

$$b(x)=b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$
$$b(x)\bullet x = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

Una vez tenemos este resultado se debe realizar la reducción modulo m(x). Si b_7=0 el resultado es el mismo polinomio. Si b_7=1, m(x) debe anular el valor de x^8.

En general, para utilizar este tipo de multiplicación los autores definen una función denominada "xtime" que simplifica la multiplicación de un polinomio por potencias de x, este hecho es gracias a que la función xtime se puede ejecutar de forma reiterativa. La función "xtime" consiste en aplicar un desplazamiento a la izquierda al valor que representa el polinomio y una operación or-exclusiva con el valor 0x11B (0x11B=000100011011= $m(x)=x^8+x^4+x^3+x+1$) cuando el resultado de la multiplicación debe ser reducido módulo m(x). Esta función se puede programar fácilmente de la siguiente manera:

```
int xtime (int valor){
        valor=valor<<1;
        if(valor&0x100)
                valor^=0x11B;
        return valor;
}
```

Ejecutar una vez la función xtime equivale a multiplicar el polinomio representado por su "valor" por x, es decir el polinomio • '02'.

La importancia de esta función recae en su uso reiterado para calcular multiplicaciones de polinomios. Por ejemplo la multiplicación de los siguientes polinomios:

$$\text{'57'} \bullet \text{'13'} = (x^6+x^4+x^2+x+1) \bullet (x^4+x+1)$$

se puede ver como la multiplicación de '57' por diversas potencias de x, aplicando la propiedad asociativa. Según ésto, el polinomio '13' se puede descomponer en potencias de x de la siguiente forma:

$$\text{'13'} = \text{'01'} \oplus \text{'02'} \oplus \text{'10'} \rightarrow (x^4+x+1) = 1 \oplus x \oplus x^4$$

Por lo tanto: ‘57’ • ‘13’= ‘57’ • (‘01’ ⊕ ‘02’ ⊕ ‘10’) =
=‘57’⊕ (‘57’•‘02’) ⊕ (‘57’•‘10’)

Resolviendo:

‘57’• ‘02’ = $(x^6+x^4+x^2+x+1) \bullet x$ = xtime(57) = ‘AE’
’57’ • ‘04’ = $(x^6+x^4+x^2+x+1) \bullet x^2$ = xtime(AE) = ‘47’
‘57’ • ‘08’ = $(x^6+x^4+x^2+x+1) \bullet x^3$ = xtime(47) = ‘8E’
‘57’ • ‘10’ = $(x^6+x^4+x^2+x+1) \bullet x^4$ = xtime(8E) = ‘07’

El cálculo de las cuatro llamadas a la función xtime se observa en la siguiente tabla:

Potencia de x	Valor Hexadecimal	Valor Inicial pasado a xtime	Valor desplazado a la izquierda 1 posición	XOR con “0x11B”	Resultado
x	“02”=00000010b	“57”	“AE”		“AE”
x^2	“04”=00000100b	“AE”	“5C”	5C⊕0x11B=47	“47”
x^3	“08”=00001000b	“47”	“8E“		“8E”
x^4	“10”=00001010b	“8E”	“1C”	1C⊕0x11B=07	“07”

Operaciones mediante la función xtime.

Finalmente se calcula:

‘57’ • ‘13’= ‘57’ • (‘01’ ⊕ ‘02’ ⊕ ‘10’) = ‘57’⊕ (‘57’•‘02’) ⊕ (‘57’•‘10’) = ‘57’⊕‘AE’⊕‘07’=‘FE’

Este procedimiento es interesante ya que las multiplicaciones que se realizan en el algoritmo se pueden realizar utilizando esta función. Su uso se centra en la función MixColumn del algoritmo con polinomios representados por ‘01’,’02’,‘03’, ‘09’, ‘0b’, ‘0d’ y ‘0e’.

Consideraciones:

- Multiplicar un polinomio por ‘01’ es igual al mismo polinomio.
- Multiplicar un polinomio por ‘02’ consiste en aplicar la función xtime al polinomio.
- Mutiplicar un polinomio A por ‘03’ es igual a:

$$(A \bullet \text{‘02’}) \oplus A = \text{xtime}(A) \oplus A.$$

- Mutiplicar un polinomio A por ‘09’ es igual a:

$$(A \bullet \text{‘08’}) \oplus A = \text{xtime}(\text{xtime}(\text{xtime}(A))) \oplus A$$

- Mutiplicar un polinomio A por ‘0b’ es igual a:

$$(A \bullet \text{‘08’}) \oplus A = \text{xtime}(\text{xtime}(\text{xtime}(A))) \oplus A$$

- Mutiplicar un polinomio A por ‘0e’ es igual a:

$$(A \bullet \text{‘02’}) \oplus (A \bullet \text{‘04’}) \oplus (A \bullet \text{‘08’}) =$$
$$\text{xtime}(A) \oplus \text{xtime}(\text{xtime}(A)) \oplus \text{xtime}(\text{xtime}(\text{xtime}(A)))$$

- Mutiplicar un polinomio A por ‘0d’ es igual a:

$$(A \bullet \text{‘04’}) \oplus (A \bullet \text{‘08’}) \oplus A =$$
$$= \text{xtime}(\text{xtime}(A)) \oplus \text{xtime}(\text{xtime}(\text{xtime}(A))) \oplus A$$

2.2.2 Especificación del Algoritmo.

El algoritmo Rijndael es un sistema simétrico de cifrado por bloques, por tanto utiliza la misma clave para el proceso de cifrado como para el proceso de descifrado. Su diseño permite la utilización de claves de sistema con longitud variable siempre que sea múltiplo de 4 bytes. La longitud de las claves utilizadas por defecto son 128 (AES-128), 192 (AES-192) y 256 (AES-256) bits. De la misma manera el algoritmo permite la utilización de bloques de información con un tamaño variable siempre que sea múltiplo de 4 bytes, siendo el tamaño mínimo recomendado de 128 bits (el tamaño mínimo de 16 bytes).

Este algoritmo opera a nivel de byte, interpretando éstos como elementos de un cuerpo de Galois $GF(2^8)$, y a nivel de registros de 32 bits, considerándolos como polinomios de grado menor que 4 con coeficientes que son a su vez polinomios en $GF(2^8)$.

2.2.2 Estructura del Algoritmo

La estructura del algoritmo Rijndael[(1)] está formado por un conjunto de "rondas", entendiendo por "rondas" un conjunto de reiteraciones de 4 funciones matemáticas diferentes e invertibles.

Por tanto, el algoritmo se basa en aplicar un número de rondas determinado a una información en claro para producir una información cifrada. La información generada por cada función es un resultado intermedio, que se conoce como Estado, o si se prefiere Estado Intermedio.

El algoritmo representa el "Estado" como un matriz rectangular de bytes, que posee 4 filas y N_b columnas. Siendo el número de columnas Nb en función del tamaño del bloque:

N_b = tamaño del bloque utilizado en bits / 32

[(1)] **Nota:** El algoritmo Rijndael no posee una estructura tipo Feistel, a diferencia de muchos algoritmos simétricos, como por ejemplo el antiguo estándar DES. Este hecho le permite producir una mayor difusión de la información cifrada con un menor número de vueltas o aplicación de funciones matemáticas.

Por ejemplo la representación de una matriz de Estado para un tamaño de bloque de 160 bits ($N_b = 5$), sería:

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$

La clave del sistema se representa con una estructura análoga a la del "Estado", es decir, se representa mediante una matriz rectangular de bytes de 4 filas y Nk columnas. Siendo el número de columnas N_k en función del tamaño de la clave:

N_k = tamaño de la clave en bits / 32

Por ejemplo la representación de una clave de 128 bits ($N_k = 4$), en forma de matriz rectangular sería:

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

Una vez establecido estos parámetros iniciales el bloque que se pretende cifrar o descifrar se traslada byte a byte sobre la matriz de Estado, siguiendo la **secuencia $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, ..., $a_{3,4}$**, y análogamente los bytes de la clave se copian en la matriz de la clave siguiendo el mismo criterio, **$k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$ ... $k_{3,3}$.**

A partir de este momento la matriz de Estado sufre 4 transformaciones por "ronda" (vuelta), utilizándose en el proceso subclaves para cada ronda que se generan de la clave de sistema elegida. Las 4 transformaciones que aplica el algoritmo a la matriz de Estado por ronda son:

- **Función ByteSub:** Sustitución con propiedades óptimas de no linealidad.
- **Función ShiftRow y MixColumn:** Permiten un alto nivel de difusión de la información a lo largo de las diferentes rondas.
- **Función AddRoundKey:** Permite aplicar a la matriz de Estado una operación "or exclusiva" con la subclave correspondiente a cada ronda.

El número de reiteraciones o vueltas de las 4 transformaciones sobre la información, o mejor dicho sobre la matriz de Estado Intermedio depende de la versión del algoritmo que se utilice.

Los autores definen que para tamaños de bloques y claves entre 128 y 256 bits (con incrementos de 32 bits) el número de vueltas Nr es determinado por la siguiente expresión:

$$N_r = \max(N_k, N_b) + 6$$

Por ejemplo, para un algoritmo Rijndael de tamaño de clave y de bloque 128 bits, el número de vueltas es 10. Se observa claramente que el número de vueltas o reiteraciones del algoritmo dependen del tamaño de bloque y clave elegidos.

Número de rondas para Rijndael en función de tamaños de clave y bloque:

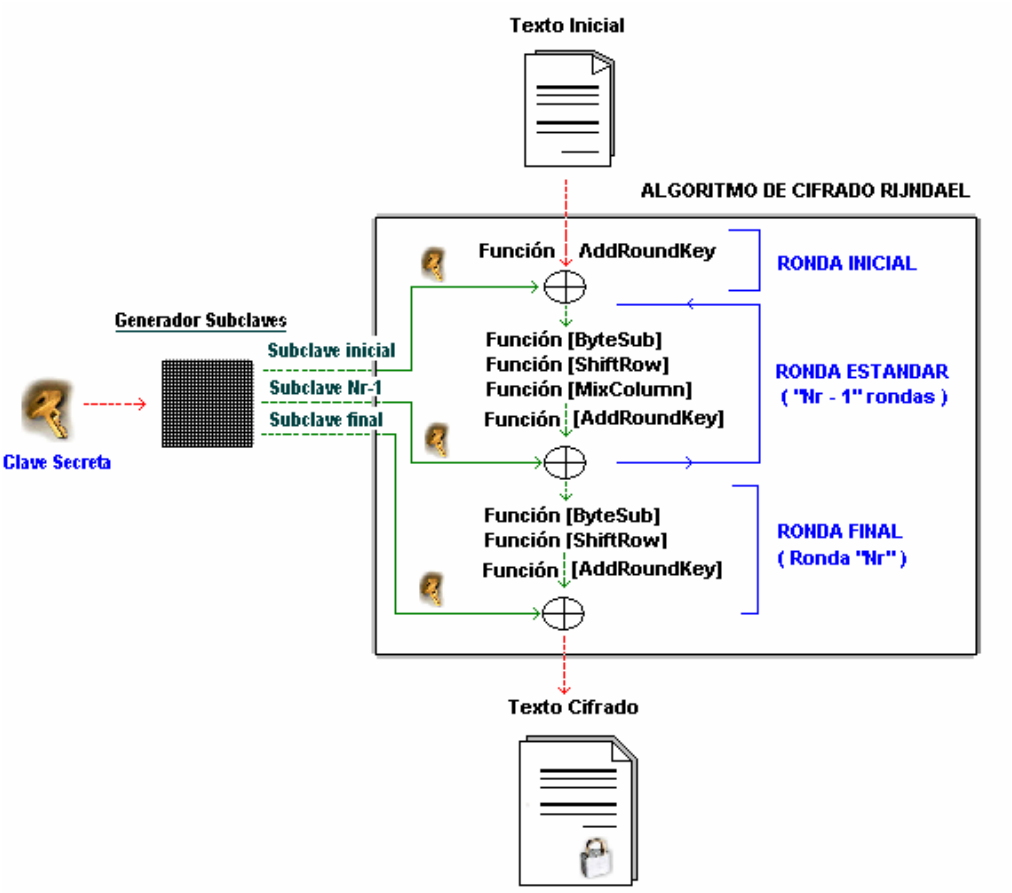
Clave / Bloque	$N_b = 4$ (128 bits)	$N_b = 6$ (192 bits)	$N_b = 8$ (256 bits)
$N_k = 4$ (128 bits)	10	12	14
$N_k = 6$ (192 bits)	12	12	14
$N_k = 8$ (256 bits)	14	14	14

Teniendo en cuenta la estructura general del algoritmo se va a profundizar, a continuación, en el proceso de cifrado y descifrado.

2.2.3 Descripción del proceso de cifrado.

El proceso de cifrado consiste en la aplicación de 4 funciones matemáticas invertibles sobre la información que se desea cifrar. Estas transformaciones se realizan de forma reiterativa para cada ronda o vuelta defina.

Gráficamente la descripción del proceso de cifrado con el algoritmo Rijndael se puede ver como:



En esencia la información a cifrar se va mapeando en la matriz de Estado. Esta matriz de Estado se introduce al cifrador, y sufre una primera transformación, en la ronda inicial, que consiste en una operación or-exclusiva (AddRoundKey) entre una Subclave generada y la matriz de Estado. A continuación, a la matriz de Estado resultante se le aplican 4 transformaciones invertibles, repitiéndose este proceso "N_r-1" veces, en lo que se conoce como Ronda Estándar. Finalmente se le aplica una última ronda o vuelta a la matriz de Estado resultante de las "Nr-1" rondas anteriores, aplicando las funciones ByteSub, ShiftRow y AddRoundKey en este orden. El resultado de la ronda final produce el bloque cifrado deseado.

En esta figura se puede observar la evolución del cifrador, y como entra en juego la clave del usuario generando subclaves que se utilizan para cada ronda.

A continuación vamos a profundizar en cada una de las funciones que configuran el algoritmo cifrador Rijndael para comprender mejor su funcionamiento.

2.2.3.1. Función ByteSub.

La transformación ByteSub consiste en una sustitución no lineal que se aplica a cada byte de la matriz de Estado (estado intermedio 1) de forma independiente, generando un nuevo byte.

Esta transformación consiste en la sustitución de cada byte por el resultado de aplicarle la tabla de sustitución S-Box. Esta tabla logicamente es invertible y se construye mediante dos transformaciones:

→ **1ª Transformación**. Cada byte es considerado como un elemento en GF(2^8) que genera el polinomio irreducible $m(x)= x^8 + x^4 + x^3 + x + 1$, siendo sustituido por su inversa multiplicativa. El valor cero queda inalterado, ya que no tiene inversa.

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

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

Transformación afín en GF(2)

Por ejemplo, si el byte al cual se le aplica la función ByteSub es "A=11001011", deberíamos calcular su inversa multiplicativa. Cada byte en Rijndael se representa como un polinomio a(x), calcular la inversa

multiplicativa consiste en buscar un polinomio b(x) (que es único) que multiplicado por a(x) modulo m(x) es igual a 1, es decir:

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

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

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Ejemplo de un valor concreto.

Obtenemos el valor final de "Y=00011111". Luego el byte "A=11001011" se convierte en Y al aplicar la función ByteSub.

Utilizando estas dos transformaciones para todos los valores posibles de entrada (256 valores ya que se trabaja con un byte) se calcula una tabla de sustitución denominada S-Box útil para el proceso de cifrado.

Gracias a esta tabla aplicar la función byteSub resulta trivial, consiste en dividir el byte de la matriz de Estado en dos partes de 4 bits. Los 4 bits más significativos, denominados por x (toma valores de 0 a 15) actúan de fila en la tabla y los 4 bits menos significativos de columna, denominados por y (toma valores de 0 a 15). El valor para esa fila y columna en la tabla es resultado de aplicar S-BOX a un byte.

Siguiendo con el ejemplo anterior, si tenemos el byte "A=11001011" y le aplicamos la función ByteSub el resultado sera (x=1100 [fila c] y=1011 [columna b]) "Y=0x1F" que en binario equivale a "Y=00011111", valor que es idéntico al calculado previamente.

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

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

Para el proceso de descifrado es necesario calcular la función inversa de ByteSub. Esta función inversa consiste en calcular una tabla inversa a la utilizada en el proceso de cifrado. Tabla inversa S-BOX:

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

2.2.3.2. Función ShiftRow.

Esta transformación consiste en rotar a la izquierda las filas que conforman la matriz de Estado actual (Estado Intermedio 2), es decir, rotar los bytes de las filas de la matriz de estado resultante de la transformación anterior (función ByteSub) a la izquierda.

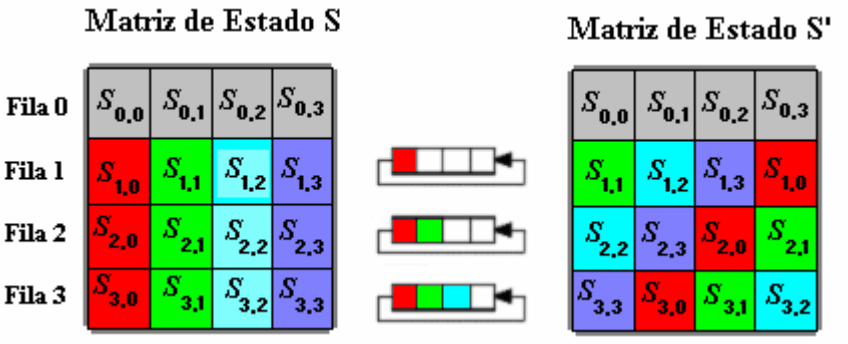
Cada fila f_i se desplaza un número de posiciones C_i diferente. Existen 4 C_i, uno para cada una de las 4 filas que siempre tiene la matriz de estado. C_0 (para la fila0), C_1 (para la fila1), C_2 (fila2) y C_3 (fila3).

La fila0 siempre permanece inalterada (siempre C_0=0). Los valores de C_1,C_2 y C_3 que indican el número de rotaciones de las filas f_0,f_1 y f_2 dependen del tamaño de bloque, y en consecuencia de Nb (tamaño bloque/32).

Tamaño de bloque	C1	C2	C3
128 bits ($N_b = 4$)	1	2	3
192 bits ($N_b = 6$)	1	2	3
256 bits ($N_b = 8$)	1	3	4

[Valores de C_i según el tamaño de bloque].

Por ejemplo, si el tamaño de bloque fuera de 128 bits (N_b = 4) la fila 0 no sería rotada, la fila 1 se rotaría 1 byte (C_1=1), la fila 2 sería rotada 2 bytes (C_2=2) y la fila 3 sería rotada 3 bytes (C_3=3).



[Funcion ShitfRow para bloque de 128 bits].

La función inversa de ShiftRow que permite invertir esta transformación consiste simplemente en rotar a la derecha los bytes de las filas de la matriz de Estado actual. Para ello se desplaza el mismo número de posiciones C_i que se desplazaron para cifrar.

La estructura del algoritmo fue diseñada para permitir cualquier tamaño de bloque que sea múltiplo de 4 bytes, con un número mínimo de 16 bytes. Las funciones **AddRoundKey**, **ByteSub** y **MixColumn** son independientes del tamaño de bloque. Sin embargo la transformación **ShiftRow** si depende de la longitud del bloque, siendo necesarios definir valores C_1,C_2 y C_3 diferentes para distintos bloques. Los autores teniendo, en cuenta ésto, han facilitado valores adicionales de C_1,C_2 y C_3 para otras longitudes comunes de bloques:

Tamaño de bloque	C1	C2	C3
160 bits ($N_b = 5$)	1	2	3
224 bits ($N_b = 7$)	1	2	4

[Valores extras de C_i según el tamaño de bloque].

Para cualquier otra longitud de bloque no establecida en el estándar es conveniente contactar con los diseñadores del algoritmo.

2.2.3.3. Función MixColumn.

La transformación MixColumn actúa sobre los bytes de una misma columna de la matriz de Estado que tiene a la entrada, es decir, el Estado intermedio 3. En esencia esta función permite una mezcla de los bytes de las columnas.

Esta transformación considera las columnas de bytes como polinomios cuyos coeficientes pertenecen a $GF(2^8)$, es decir, son también polinomios.

La función MixColumn consiste en multiplicar las columnas de bytes módulo x^4+1 por el polinomio c(x).

Matemáticamente c(x) viene representado por:

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

Este polinomio c(x) es coprimo con x^4+1, lo que permite que sea invertible. En forma algebraica esta función se puede representa como:

$$s'(x) = c(x) \otimes s(x)$$

Donde s´(x) representa la matriz de Estado resultante de esta transformación (Estado Intermedio 4) y s(x) la matriz de Estado entrante (Estado intermedio 3).

Esta fórmula queda mejor expresada de forma matricial, donde "c" representa el índice de la columna que se procesa:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Desarrollando la matriz, observamos claramente como cada byte nuevo de la matriz de Estado es una combinación de varios bytes de las distintas filas que forman una columna específica:

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

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

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

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

Por ejemplo dado el siguiente Estado Intermedio3:

D4	E0	B8	1e
Bf	B4	41	27
5d	52	11	98
30	Ae	F1	E5

Calcular el byte de la fila0 y columa0 de la matriz de Estado resultante se calcularía de la siguiente forma:

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

Se cogen los bytes de la matriz que hacen falta:

$$s'_{0,0} = (\{02\} \bullet \{d4\}) \oplus (\{03\} \bullet \{bf\}) \oplus \{5d\} \oplus \{30\}$$

Y se calcula el resultado:

{02} • {d4} = {b8} → x • ($x^7+x^6+x^4+x^2$) mod x^4+1 =$x^8+x^7+x^5+x^3$ mod x^4+1 = **$x^7+x^5+x^4+x^3$=10111000={b8}**

{03}•{bf}={d1} → (x+1)•($x^7+x^5+x^4+x^3+x^2+x+1$) = $x^8+ x^6 +x^5 + x^4 +x^3 +x^2 +x +x^7+x^5+x^4+x^3 +x^2 +x+1 =x^8+x^7+x^6+1$ mod x^4+1= **$x^7 +x^6 +x^4$ +1= 11010001={d1}**

$$s'_{0,0} = b8 \oplus d1 \oplus 5d \oplus 30 = 04$$

Luego el byte D4 de la fila y columna0 de la matriz de Estado Intermedia 3 se sustituiría por el byte 04 en la matriz de Estado Intermedia 4. Para el resto de bytes se seguiría un desarrollo como el planteado.

Para descifrar o invertir esta transformación, se deberá realizar el mismo procedimiento descrito pero con el polinomio d(x), que es el inverso de c(x).

La inversa de MezclarColumnas se obtiene multiplicando cada columna de la matriz de estado por el polinomio d(x):

$$d(x) = \text{'0B'}x^3 + \text{'0D'}x^2 + \text{'09'}x + \text{'0E'}$$

Este polinomio, permite realizar la inversa del polinomio c(x), cumpliendo que:

$$c(x) \otimes d(x) = \text{'01'}$$

La transformación inversa se podría mostrar de forma matricial de la siguiente forma:

$$s(x) = d(x) \otimes s'(x)$$

$$\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix}$$

Los resultados de esta multiplicación, dan como resultado la siguiente sustitución de los bytes:

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

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

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

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

El lector, puede comprobar como aplicando estas formulas al ejemplo anterior se recuperan los bytes originales de la matriz de Estado previa.

2.2.3.4. Función AddRoundKey.

Esta transformación consisten en aplicar una operación OR-Exclusiva entre la matriz de Estado que proviene de la transformación anterior (Función MixColumn) y una subclave que se genera a partir de la clave del sistema para esa vuelta (ronda).

El bloque resultante de esta transformación, será la nueva matriz de Estado para la siguiente ronda. Siendo el bloque de salida, si la vuelta es la última.

Para aplicar esta transformación se debería coger una subclave para la vuelta actual, y formar una matriz con el mismo número de filas y columnas que la matriz de Estado con la que se está operando (implícitamente se puede observar como las subclaves dependen del tamaño del bloque empleado) para poder aplicar la operación or-exclusiva.

Estado Intermedio 4				Round Key				Estado Intermedio1 o Bloque de Salida.			
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{,3}$	K_0	K_1	K_2	K_3	$S_{0,0} \oplus K_0$	$S_{0,1} \oplus K_1$	$S_{0,2} \oplus K_2$	$S_{0,3} \oplus K_3$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	K_4	K_5	K_6	K_7	$S_{1,0} \oplus K_4$	$S_{1,1} \oplus K_5$	$S_{1,2} \oplus K_6$	$S_{1,3} \oplus K_7$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	K_8	K_9	K_{10}	K_{11}	$S_{2,0} \oplus K_8$	$S_{2,1} \oplus K_9$	$S_{2,2} \oplus K_{10}$	$S_{2,3} \oplus K_{11}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	K_{12}	K_{13}	K_{14}	K_{15}	$S_{3,0} \oplus K_{12}$	$S_{3,1} \oplus K_{13}$	$S_{3,2} \oplus K_{14}$	$S_{3,3} \oplus K_{15}$

Ejemplo de transformación AddRoundKey para N_b=4

La esencia de esta función recae en las subclaves. El algoritmo Rijndael basándose en el principio de la criptografía moderna, mediante el cual se establece que la seguridad de un algoritmo sólo debe depender de la clave utilizada, utiliza diferentes subclaves K_i tanto en el cifrado como en el descifrado para que el resultado del algoritmo dependa completamente de una información externa al sistema: la clave de usuario.

El número total de bits necesarios para generar todas las subclaves, depende del número de rondas que se aplique al algoritmo y del tamaño del bloque empleado. El número total de bits de subclaves necesarios se puede calcular como:

$$\text{Nº total bits Subclaves} = 32 * N_b * (N_r + 1)$$

Por lo tanto, se puede observar que el número total de bits de subclaves es igual al tamaño del bloque empleado por el número de vueltas del algoritmo (las N_r vueltas del algoritmo más la ronda inicial)

Bloque/ Clave	$N_k = 4$ (128 bits)	$N_k = 6$ (192 bits)	$N_k = 8$ (256 bits)
$N_b = 4$ (128 bits)	1408 bits ($N_r = 10$)	1664 bits ($N_r = 12$)	1920 bits ($N_r = 14$)
$N_b = 6$ (192 bits)	2304 bits ($N_r = 12$)	2496 bits ($N_r = 12$)	2880 bits ($N_r = 14$)
$N_b = 8$ (256 bits)	3840 bits ($N_r = 14$)	3328 bits ($N_r = 14$)	3840 bits ($N_r = 14$)

Nº de bits de sublaves para tamaños estándar de clave y bloque.

Se ha observado que el funcionamiento de esta transformación es sencillo, lo verdaderamente interesante es conocer el procedimiento para generar los bytes que forman las subclaves para cada vuelta (RoundKeys), bytes que se derivan de la clave principal K. Para ello el sistema utiliza dos funciones auxiliares: una función de selección y una función de expansión.

2.2.3.4.1. Función de selección de clave.

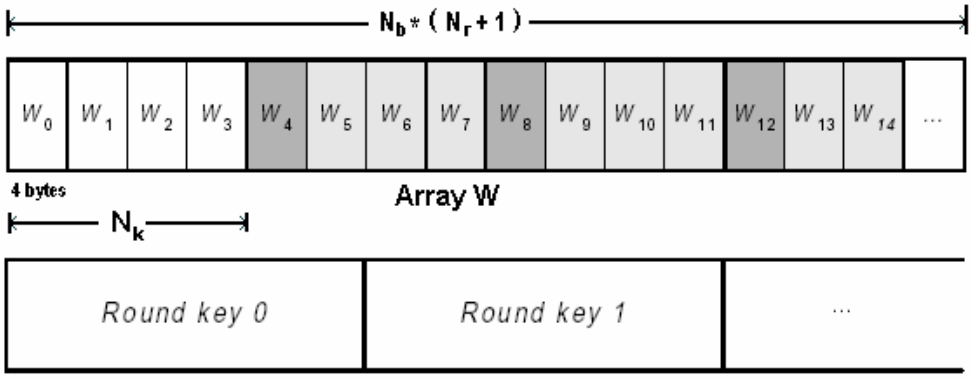
La función de selección simplemente toma consecutivamente de la secuencia obtenida por la función de expansión de clave bytes que va asignado a cada subclave K_i. para formar bloques del mismo tamaño que la matriz de estado. Es decir, coge N_b*4 bytes para cada vuelta.

La generación de la claves (expansión de clave) para el proceso de descifrado se hace forma idéntica al proceso de cifrado. La diferencia reside en la función de selección de clave. En el proceso de descifrado se cogen bytes de la lista de claves desde los valores finales hasta llegar a los iniciales, que es la propia clave de usuario. Es decir, la última subclave que se utilizó para cifrar, será la primera que se utilizará para descifrar.

2.2.3.4.2. Función de expansión de clave.

La función de expansión de clave permite generar bytes útiles como subclaves a partir de la clave de sistema K. Este función de expansión se puede describir como un array lineal, denominado W, de palabras de 4 bytes y con una longitud de N_b*(N_r+1).

Las primeras N_k palabras de este array contienen la clave de cifrado, ya que la clave del usuario se mapea tal cual al array W, mientras que el resto de palabras se van generando a partir de estas primeras N_k palabras.



Ejemplo de subclaves y clave de expansión para N_b=6 y N_k=4

Se observa cómo la función de expansión de clave depende del valor de N_k, ante este hecho los autores definieron dos versiones para esta función, una para $N_k \leq 6$ y otra para $N_k > 6$, por motivos de seguridad. Estas versiones se describen de la siguiente forma utilizando el lenguaje de programación C.

Para $N_k \leq 6$ tenemos:

```
KeyExpansion( byte Key[4*Nk] word W[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 = ByteSub (RotByte(temp)) ^ Rcon[i/Nk];
        w[i] = W[i-Nk]^temp;
  }
}
```

Para $N_k > 6$

```
KeyExpansion(byte Key[4*Nk] word W[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 = ByteSub(RotByte(temp))^Rcon [i/Nk];
        else if ( i%Nk == 4)
          temp = ByteSub(temp);
        w[i] = w[i-Nk]^temp;
 }
}
```

En el código anterior, se observa como se mapea directamente la clave del usuario al array W, se mapean N_k palabras. El resto de funcionamiento permite generar bytes para subclaves, en este proceso entran en juego la función ByteSub que devuelve el resultado de aplicar la S-BOX de Rijndael a cada uno de los bytes de los 32 bits de la palabra que se le pasa como parámetro. La función Rot que rota una posición a la izquierda los bytes de la palabra, de tal forma que si se le pasa como parámetros la palabra de 4 bytes (a,b,c,d) devuelve (b,c,d,a).Y la función Rcon que genera una constante teniendo en cuenta que:

$$Rcon(j)=(R(j),0,0,0)$$

Cada R(j) es el elemento $GF(2^8)$ correspondiente al valor x^{j-1}

Desde un punto de vista más esquemático la función de expansión se puede ver como:

Para $N_k \leq 6$:

Para todo valor de "i"[1] que no sea múltiplo de N_k, las palabras de subclaves se calculan como:

$$W(i) = W(i - N_k) \oplus W(i -1)$$

Para todo valor de "i" que sea multiplo de N_k se calculan como:

$$W(i) = W(i - N_k) \oplus [ByteSub(RotByte[W(i-1)]) \oplus Rcon(i/N_k)]$$

Si por ejemplo se utiliza una clave de 128 bits ($N_k = 4$) y un bloque de 128 bits ($N_b=4$) la longitud del array W sería 44 (4*[10+1]). En las cuatro primeras posiciones del array (0 a la 3) se copia la clave de usuario, el resto de posiciones, de la posición 4 a la 43, se rellenaría con valores calculados. En este caso la variable "i" tomaría los valores de 4 a 43. Para las posiciones del array W(i) con valor de "i" 4, 8, 12, 16, 20, 24, 28, 32, 36, 40 (i múltiplo de N_k), la palabra de 4 bytes se calcularía cogiendo una palabra que se encuentra 4 posiciones antes que ella y realizando una operación or-exclusiva con una transformación de la palabra que se encuentra una posición antes. Para el resto de valores de i la palabra correspondiente a W(i) se calcula realizando una operación or-exclusiva entre la palabra que se encuentra 4 posiciones antes en el array y la palabra que se encuentra una posición antes que la posición en la que nos encontramos.

Este ejemplo permite observar el procedimiento que sigue el algoritmo para generar nuevos bytes de subclaves, apoyándose en palabras ya existentes en el array W, ya sea palabras de la clave de usuario directamente o palabras generadas de esta.

[1] **Nota:** La variable "i" denota la posición dentro del array W(i) que contiene los bytes de subclaves del sistema.

Para $N_k>6$

El funcionamiento para $N_k>6$ es igual que para $N_k\leq6$ salvo cuando el valor de la variable "i" satisface que **"i mod N_k = 4"**, en este caso las palabras de subclaves se calculan como:

$$W(i) = W(i - N_k) \oplus \text{ByteSub}\ (W[i-1])$$

Una vez conocido el funcionamiento de la función de expansión de clave, veáse, para finalizar con la explicación de la función un ejemplo real de generación de bytes de subclaves para un algoritmo Rijndael con clave de 128 bits

[Ejemplo]

Se selecciona clave de usuario.

Clave= 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c (N_k=4)

Se mapea a las 4 primeras posiciones del array la clave del usuario,

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

A partir de este momento se empieza a generar bytes de subclaves.

Calcular W[4]:

i=4 (i múltiplo de Nk)
temp= W[3] = 09 cf 4f 3c
Se aplica función de rotación Rot → temp = cf 4c 3c 09
Se aplica función ByteSub → ByteSub (temp) = 8a 84 eb 01
Rcon[4/4]=Rcon[1]=[x^0,{00},{00},{00}]= {01},{00},{00},{00}]
temp= Rcon[1] ⊕ temp=8a 84 eb 01 ⊕ 01 00 00 00 = 8b 84 eb 01
W[4] = W[0] ⊕ temp = 2b 7e 15 16 ⊕ 8b 84 eb 01 = **a0 fa fe 17**

Siguiendo todos los criterios descritos en esta función se podrían generar todos los bytes de subclaves necesarios para la versión del algoritmo con la que se trabajará.

2.2.4 Descripción del proceso de Descifrado.

La descripción del proceso de descifrado del algoritmo Rijndael es sencilla. Consiste en sustituir las transformaciones utilizadas en el cifrado por sus inversas e invertir el orden de aplicación de dichas transformaciones o funciones matemáticas.

Teniendo en cuenta ésto la estructura del algoritmo de descifrado sería:

Vuelta Final:

```
InvAddRoundKey(Estado,RoundKey);
InvShiftRow(Estado);
InvByteSub(Estado);
```

Vuelta Estándar:

```
InvAddRoundKey(Estado,RoundKey);
InvMixColumn(Estado);
InvShiftRow(Estado);
InvByteSub(Estado);
```

Salida:

```
InvAddRoundKey(Estado,RoundKey);
```

De esta forma tan sencilla se construye el algoritmo para descifrar una información cifrada con el algoritmo Rijndael.

2.2.5. Motivos de Diseño.

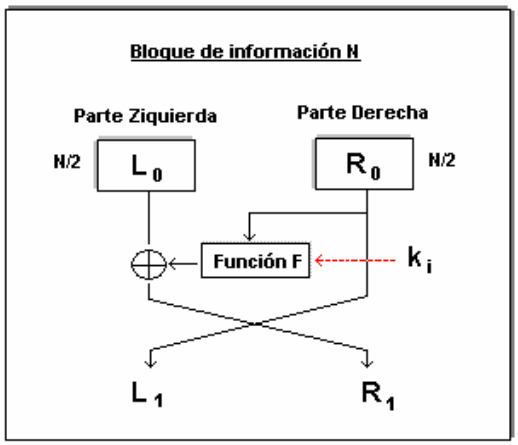
En este apartado se van a discutir los parámetros de diseño del algoritmo Rijndael, analizando sus motivos de elección y la influencia que tienen sobre la seguridad del algoritmo. La base de los siguientes razonamientos se encuentran en un estudio que sus autores publicaron al respecto. Teniendo en cuenta esto el algoritmo Rijndael se estructura en torno a 3 axiomas:

- Resistencia contra todos los ataques conocidos.
- Rapidez y compatibilidad en un gran abanio de plataformas
- Sencillez en el diseño.

Teniendo en cuenta estos principios se va a profundizar en los motivos de diseño de las funciones matemáticas que utiliza el algoritmo, el número de vueltas y su estructura algebraica. Este conocimiento ofrecerá al lector una idea más amplia de la robustez del algoritmo Rijndael.

2.2.5.1. Estructura del Algoritmo

La mayoría de los cifradores simétricos tienen una estructura tipo Feistel[(1)]. En esta estructura la vuelta o ronda de transformación consiste en separar un bloque de un mensaje en dos partes, una parte izquierda y otra derecha, e ir conmutando las partes de izquierda a derecha aplicándole una función unidireccional. Este procedimiento reiterado un número de veces constituye la estructura del algoritmo. Muestra de esta estructura se encuentra en el antiguo estándar de cifrado DES.



(1) **Nota:** Estructura desarrollada por Horst Feistel. Investigador del laboratorio Thomas J. Watson de la IBM. En la década de los 70 desarrollo el algoritmo Lucifer, a partir del cual se baso el antiguo estándar de cifrado DES.

Un sistema Feistel procesa la mitad de los bits del bloque en una vuelta quedando el resto inalterados. Rijndael no tiene una estructura tipo Feistel (trata todos los bits por vuelta), en vez de ello se decidió definir la vuelta de transformación como tres funciones invertibles llamadas capas. La elección de las diferentes capas está basada en gran parte en la obra "***Cipher and hash function design strategies based on linear and differential cryptanalysis***", que permitió a los autores pensar en una estructura de algoritmo que llevara implícita la resistencia contra el criptoanálisis lineal y diferencial. Teniendo en cuenta esto, cada capa cobraría su propia utilidad (estas capas está constituidas por las 4 transformaciones matemáticas del algoritmo):

- Una capa de mezcla lineal, que garantiza una alta difusión de la información a través de la aplicación de varias vueltas.

- Una capa no lineal, que permita propiedades aptas de no linealidad.

- Una capa de adición de clave, que consiste en una simple operación "or exclusiva" entre la subclave de cada vuelta y el Estado intermedio.

Se puede observar como el algoritmo utiliza aparte de las rondas estándar una ronda final y otra inicial. La ronda inicial es útil para aplicar una subclave antes de la primera vuelta. El motivo de aplicar esta clave consiste en que no se pueden atacar directamente a las capas sin conocimiento de la clave. Tal es esta propiedad que se utiliza en otros algoritmos como el algoritmo IDEA, SAFER y Blowfish. Por otro lado para que el algoritmo cifrador y el descifrador tengan una estructura lo más similar posible, la capa de mezcla lineal de la última vuelta es diferente de la capa de mezcla lineal de las otras vueltas [(1)], este es el motivo por el cual la ronda final difiere de la ronda estándar.

[(1)] **Nota:** Se ha demostrado que modificar la capa de mezcla lineal de la ultima vuelta no reduce la seguridad del cifrador.

2.2.5.2. Función ByteSub.

La función ByteSub aporta una capa no lineal diseñada específicamente para ofrecer resistencia frente a ataques como el criptoanálisis diferencial, el criptoanálisis lineal, ataques de interpolación, etc. Los criterios para diseñarla fueron:

- Que sea inversible
- Minimizar la relación lineal entre bits de entrada y bits de salida.
- Complejidad mediante expresiones algebraicas en $GF(2^8)$
- Sencillez de diseño.

Existen diversas formas para construir la S-Box, los autores en un primer paso definieron la tabla de sustitución mediante el calculo de inversas multiplicativas en $GF(2^8)$ la cual permite propiedades de no-linealidad. Sin embargo, la simplificidad algebraica de este diseño podría permitir manipulaciones algebraicas que permitieran ataques. Para evitarlo se añadió una transformación afín invertible adicional. Esta transformación afin está formada por una matriz que tiene una representación sencilla pero esconde una compleja expresión algebraica, más un vector ("01100011") elegido de tal forma que para ningún valor la tabla S-Box ofrezca las siguientes relaciones:

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

$$\text{S-Box}(a) = \overline{a}$$

Los autores descatacan que existen otras tablas S-Box que satisfacen los criterios de diseño, es más, en el caso de sospecha de una puerta trasera en el cifrador, la tabla S-BOX actual podría ser remplazada por otra. La estructura del cifrador y el número de vueltas están definidos incluso para usar tablas S-Box que no optimizen las propiedades contra ataques como el criptoanálisis diferencial y lineal, sin perder la seguridad del algoritmo.

2.2.5.3. Desplazamientos en la función ShitfRow.

El número a desplazar en la transformación ShiftRow fueron elegidos para ofrecer resistencia contra ataques conocidos como el denominado "truncated differentials" y resistencia contra el ataque "Square". Se demostró que para ciertos desplazamientos los ataques por "truncanted differentials" y "Square" encuentran más resistencia por parte del algoritmo. Teniendo en cuenta esto, se elegieron los mejores desplazamientos contra estos ataques.

2.2.5.4. Función MixColumn.

La función MixColumn ha sido seleccionada como transformación lineal de 4 en 4 bytes, teniendo en cuenta los siguientes criterios:

- Que sea inversible
- Linealidad en GF(2)
- Propiedades de alta difusión.
- Velocidad en procesadores de 8 bits.
- Simetría.
- Sencillez de descripción.

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

2.2.5.5. Número de Vueltas.

La estructura del algoritmo Rijndael ya es conocida, consistiendo su funcionamiento en la reiteración de las funciones matemáticas que se definen en esta estructura. Son estas reiteraciones las que se conocen como "vueltas o rondas".

El número de vueltas fue determinado por los autores, buscando el mínimo numero de vueltas necesarias para ofrecer un margen de seguridad considerable frente a los ataques conocidos.

A) Por ejemplo, para un tamaño de bloque y de clave de 128 bits se utilizan 10 vueltas, ésto es así porque no se han encontrado atajos en ataques para versiones reducidas con más de 6 vueltas. A estas 6, los autores, le suman otras 4 vueltas como margen de seguridad. Esto es un mecanismo de precaución, porque:

“Dos vueltas de Rijndael producen una difusión completa, en el sentido de que, cada bit del estado depende de todos los bits de las 2 vueltas anteriores, es decir, un cambio en un bit del estado es similar a cambiar la mitad de los bits del estado después de dos vueltas. La alta difusión de una vuelta de Rijndael es gracias a su estructura uniforme que opera con todos los bits del estado. Para cifradores tipo Feistel (como por ejemplo. DES), un vuelta sólo opera con la mitad de los bits de estado y una difusión completa se obtiene en el mejor de los casos después de 3 vueltas y en la práctica 4 o más”.

En general, ataques como el criptoanálisis lineal y diferencial aprovechan el rastro dejado a través de “n” vueltas para atacar “n+1” o “n+2” vueltas. Este es además el caso del ataque “Square” que usa 4 vueltas de la propagación del algoritmo para atacar 6 vueltas. En este sentido, se suman 4 vueltas para actualmente doblar el número de vueltas mediante las cuales se pueden encontrar pista, rastros que faciliten este tipo de ataques.

B) Para versiones de Rijndael con la clave más grande, el número de vueltas es incrementado por uno por cada 32 bits adicionales de la clave de cifrado. Esto es así por los siguientes motivos:

“El principal objetivo es evitar atajos más efectivos que un ataque por fuerza bruta. Como con el tamaño de la clave la cantidad de trabajo para una búsqueda exhaustiva de la clave crece, los “atajos” son menos eficientes para claves más grandes. Conocer parte de la clave o ataque con claves relacionadas explotan el conocimiento de bits de la clave de cifrado o la habilidad para aplicar diferentes claves de cifrado. Si la clave de cifrado crece, las posibilidad disponibles para el criptoanalista disminuyen”.

C) Para versiones con un tamaño de bloque mayor de 128 bits, el numero de vueltas es incrementado por uno por cada 32 bits adicionales en el tamaño de bloque, por las siguientes razones:

“Para un bloque mayor de 128 bits, el tamaño del bloque disminuye la difusión completa del estado en una vuelta. El mayor tamaño del bloque produce rangos de posibles patrones que pueden ser aplicados a la entrada/salida de una secuencia de una vuelta para incrementarla. Esta flexibilidad añadida podría facilitar ataques”.

Los autores, reiteran de forma persistente que estos márgenes son conservadores. Siempre ampliables para incremento de la seguridad.

2.2.5.6. Expansión de Clave.

La función "Expansión de clave" permite derivar de la clave de cifrado subclaves para cada vuelta. Su utilidad reside en permitir la resistencia contra los siguientes tipos de ataques:

- **Ataques en los cuales parte de la clave de cifrado es conocida por los criptoanalistas.**

- ***Ataques donde la clave de cifrado es conocida o puede ser elegida***, por ejemplo, si el cifrador es usado como la función de comprensión de una función hash.

- ***Ataques por clave relacionada*.** Se deben evitar que distintas claves de cifrado puedan producir un gran conjunto de subclaves comunes para cada vuelta.

Además de esto, la función de "Expansión de clave" juega un papel vital en la eliminación de la simetría de la estructura del algoritmo, es decir:

- **Simetría en una ronda:** La ronda de transformación trata todos los bytes de un Estado en general de la misma forma. Esta aparente simetría es alterada gracias a las subclaves que se obtienen para cada vuelta.

- **Simetría entre rondas:** La ronda de transformación es idéntica para todas las vueltas o rondas. Esta igualdad es alterada obteniendo subclaves que dependen de cada ronda.

La eliminación de la simetría y la propia estructura del algoritmo permite que resulte prácticamente imposible la existencia de claves débiles o semidébiles como en otros algoritmos criptográficos como DES, IDEA, etc. Por tanto, el algoritmo no tiene restricción en utilizar cualquier clave dentro del espacio de claves permitido.

Teniendo en cuenta todo esto, se puede resumir que la función de "Expansión de clave" ha sido diseñada teniendo en cuenta los siguientes criterios:

- Usará una transformación invertible. Por ejemplo, el conocimiento de cualquiera N_k palabras consecutivas de la clave expandida permitirá regenerá todo el array de bytes de subclaves.

- Rápidez en un gran abanico de procesadores.

- Proporcionar constantes dependientes de cada ronda que elimine la simetría.

- **El conocimiento de parte de la clave** de cifrado o bits de una subclave de una ronda no permitirá calcular muchos bits de otras subclaves.

- **Proporcionar no-linealidad** que prohiba determinar diferencias en la subclave de cada ronda a partir de diferencias de la clave de cifrado.

- Sencillez de diseño.

2.2.6. Seguridad del Algoritmo.

El algoritmo Rijndael es el sistema de criptografía simétrica más robusto que se conoce en la actualidad. Según los estudios publicados, no existen ningún mecanismo comprobado para invertir el algoritmo estándar AES más eficiente que una búsqueda exhaustiva de claves (ataque por fuerza bruta). Por tanto, la fortaleza del sistema dependerá sólo de la longitud de la clave. Un atacante que desee realizar un ataque por fuerza bruta deberá tener en cuenta las siguientes consideraciones:

Texto en claro desconocido y clave desconocida.

Si el atacante sólo dispone de un bloque cifrado, debería cifrar con todas las claves posibles todos los bloques en claro posibles, para ir comparando el resultando con el bloque cifrado. Para valores estándar del algoritmo, como un clave de 256 bits y un tamaño de bloque de 256 bits, se demuestra fácilmente que el algoritmo no se puede invertir. Si se considera que el tamaño del bloque es de ***"v bits"*** y el tamaño de la clave es ***"n bits"***. El atacante debería probar para cada clave 2^v bloques posibles, y repetir este proceso para todas las claves (2^n). Actualmente resulta computacionalmente imposible realizar estos cálculos para valores estándars del algoritmo. De todas formas aunque fuera factible el cálculo encontraríamos un número muy elevado de parejas clave-texto en claro diferentes que producen el mismo bloque cifrado, luego el atacante no tendría capacidad para identificar cual de todas esas parejas es la válida.

Esta es la situación normal, que es cuando el atacante sólo dispone de texto cifrado. Asegurándose que el algoritmo resulta imposible de invertir.

Texto en claro conocido y texto cifrado conocido

El atacante posee el texto en claro y el texto cifrado. Ante esta situación puede hacer lo que se conoce como un ataque de fuerza bruta puro, es decir cifrar el texto en claro con todas las claves posibles hasta producir un resultado que coincida con el texto cifrado. En esta situación un atacante necesitaría aplicar el algoritmo Rijndael al texto en claro para estar seguro de que ha obtenido la clave el siguiente número de veces:

- Para una clave de 128 bits, se necesitaría aplicar 2^{127} veces el algoritmo Rijndael sobre el texto en claro y compararlo con el texto cifrado.

- Para una clave de 192 bits, el ataque necesita aplicar 2^{191} veces el algoritmo.
- Para una clave de 256 bits, el ataque necesita aplicar 2^{255} veces el algoritmo.

Este tipo de cálculo resulta hoy por hoy impracticable, siempre a la espera de futuros ordenadores cuánticos. Estos cálculos no son factibles debido al enorme número de combinaciones a calcular. Para hacerse una idea de las dimensiones de este número de cálculos, romper un algoritmo Rijndael de 128 bits es 4.722.366.482.869.645.213.696 veces menos probable que una persona gane la lotería y muera fulminado por un rayo el mismo día.

Ninguno de los siguientes ataques conocidos contra cifradores de bloques se pueden aplicar a Rijndael de una forma más efectiva que una búsqueda exhaustiva de clave:

- Linear Cryptanalysis.
- Differencial cryptanalysis.
- Truncated differentials.
- Interpolation attacks.
- Square attack.

El propio diseño del algoritmo anula los ataques sofisticados que se conocen contra los cifradores de bloques, sin embargo debido a su estructura algebraica puede inducir a una nueva serie de ataques, que en ciertas situaciones pudieran necesitar menos cálculos que una búsqueda exhaustiva de clave.

2.2.6.1. Ataques Algebraicos.

La estructura algebraica de Rijndael le conduce a la aplicación de nuevos tipos de ataques.

[ATAQUE 1]. Nicolas Courtois y **Jose Pieprzyk** han mostrado que el algoritmo puede describirse como un sistema de ecuaciones cuadráticas multivariables, y a continuación aplicar técnicas para tratar los términos de estos polinomios como variables individuales. Este hecho permitiría definir un sistema de ecuaciones lineales con un gran número de variables cuadráticas que hay que resolver. Su teoría indica que podría romperse un Rijndael de 128 bits recuperando la clave secreta. Para ello necesitaría un sólo bloque de texto en claro, y una representación del algoritmo con un sistema de 8000 ecuaciones cuadráticas con 1600 incognitas binarias.

Teniendo en mente esto, se desarrolló el ***algoritmo XL*** que permitiría resolver tales sistemas en un tiempo no-exponencial, lo que suponía una revolución, ya que la seguridad de cifradores como Rijndael no crecería exponencialmente con el número de vueltas ante este tipo de ataques. En la práctica el algoritmo XL se demostró rotundamente ineficiente para romper Rijndael, pero sin embargo presentó muchas propiedades interesantes.

Teniendo en cuenta estas propiedades han surgido un nuevo tipo de ataques, los denominados ***ataques XSL***. Ataques genéricos que pueden ser aplicados (al menos en teoría) a cualquier bloque cifrado.

[ATAQUE 2]. Por otro lado, **Fuller y Millan** publicaban un documento demostrando que la S-BOX de 8x8 bit de AES era en realidad una caja-S de 8x1 bit, demostrando que sólo hay una parte de no-linealidad en el cifrado; todo lo demás es lineal. Otro documento provino de **Filiol**. Afirmó haber detectado algunas desviaciones en las funciones booleanas de AES, que quizás podrían utilizarse para romper AES. Pero el documento da tan pocos detalles que esa afirmación aún no procede.

[ATAQUE 3]. En Crypto 2002, **Murply y Robshaw** publicaron un resultado sorprendente, que permitía expresar todo AES en un solo campo. Presentaron un cifrado llamado BES que trata cada byte de AES como un vector de 8 bytes. BES opera sobre bloques de 128 bytes; para un subconjunto especial de textos en claro y claves, BES es isomórfico respecto a AES. Esta representación tiene varias propiedades interesantes, propocionando al método XSL una representación mucho más concisa, que resulta en ecuaciones más sencillas que son más fáciles de resolver.

Aplicando estos principios el ataque de Courtois y Pieprzyk adquiere una complejidad de 2^{100} contra AES, avance que es importante.

Sin embargo, de forma pública, los ataques conocidos basados en variantes de XSL se demuestran ineficientes contra AES debido a su margen de seguridad. Sin embargo hay tendencias a pensar que la complejidad podría disminuir en pocos años.

En la siguiente página http://www.cryptosystem.net/aes/ se pueden encontrar muchos de los estudios actuales sobre el criptoanálisis de este algoritmo.

3. Tendencias y Opinión del Autor.

El algoritmo DES, a estado presente en nuestras comunicaciones desde hace ya casi tres decadas, y casi desde que se estableció ya estaba roto. En 1977, cuando DES se implanta, W.Diffie y M.E Hellman ya propusieron una máquina de propósito especial, "una computadora paralela que utilizaría un millón de chips para probar un millón de claves cada uno por segundo", estimando su costa en unos 20 millones de dólares. Dicha máquina sería capaz de encontrar la clave de cifrado en doce horas partiendo de un texto en claro y su correspondiente cifrado.

Hay muchas hipótesis, sobre el diseño del DES, que hacen pensar en la posibilidad de que el gobierno EEUU e inicialmente IBM, pudiera recuperar claves de forma sencilla. Recordar que DES se implantó en un concurso anterior al AES, en el cual ganó el algoritmo de IBM LUCIFER de 128 bits, que luego fue adaptado y reducida su clave a 56 bits estableciendo el DES. (Pensar en la diferencia de un ataque de fuerza bruta a 2^{56} combinaciones frente a 2^{128} combinaciones).

En 1993, M.Wiener describió detalladamente el diseño de una máquina para la búsqueda exhaustiva de claves constituida por 5760 chips que realizaban 16 cifrados simultáneamente y capaz de encontrar las claves en menos de dos días de trabajo ininterrumpido, con un coste de la máquina de 100.000 dólares. Este mismo investigador propuso otra nueva máquina capaz de realizar la misma tarea en poco más de 3 horas, a costa de incrementar sus requisitos, en concreto, utilizando 57600 de los chips antes referidos, con un coste que ascendía a más de un millón de dólares.

Actualmente, y aprovechándose del diseño de una maquina que permitiera romper el DES, asociaciones como la Electronic Frontier Foundation (EFF), ofrecen el diseño de la máquina DES-CRACKER que permite atacar el DES por fuerza bruta en cuestión de horas y con la ayuda de maquinas conectadas a Internet.

Por ejemplo, hace ya un tiempo me baje de Internet el diseño de esta máquina, que es de libre distribución. Y por unos 20 millones de pesetas (unos 12.000 dólares) cada "ciudadano de a pie" se puede construir una máquina que ayude a romper el algoritmo DES en unas cuatro horas (este diseño es paralelizable, es decir, a más dinero, menos tiempo tarda en romper el sistema. Actualmente, he visto un diseño que consiguen romperlo en 30 minutos.)

Por tanto, ya no es que el algoritmo DES sea débil frente a instituciones poderosas como la NSA (National Security Agency), sino que es un algoritmo que puede ser franqueado por un civil “con recursos”, sin más.

Ante esta situación, surge la necesidad de establecer un nuevo estándar, que por lo menos, dificulte el descifrado de las comunicaciones a las personas de a pie. Me parece paradojico, que EEUU convoque un concurso criptográfico, que lo gane un algoritmo Europeo, y que EEUU “imponga” este estándar al resto del mundo, al final es lo de toda la vida.

Al leer los planteamientos de los autores del algoritmo, se nota un amplio conocimiento de esta ciencia, porque en sus decisiones siempre hay un toque de precaución o incertidumbre por si existe algún ataque no publicado que las tire por tierra. Sinceramente, en el documento se han expresado los motivos por los cuales fue elegido el algoritmo Rijndael, pero sin duda y conociendo los antecedentes del DES, no sé si alguna entidad de inteligencia conocen la forma de invertilo y permitirle descifrar las comunicaciones, no obstante siempre es mejor conocer como funcionan las cosas a no saber nada.

El tema de la criptografía es un asunto peliagudo, hemos pasado de la criptografía simétrica clásica heredada a lo largo de las decadas, a criptográfia pública, ahora el auge de la criptografía cuántica, estudios sobre aplicaciones concretas como criptografía “fluorescente”,etc. El tiempo nos dira o no que sistema será finalmente la panacea de la seguridad, mientras tanto lo mejor será estar atentos.