Criptografía de clave secreta

Fernando Martínez
fernando.martinez@upc.edu

Departament de Matemàtiques • Universitat Politècnica de Catalunya

21 de septiembre de 2023

Criptografía de clave secreta o simétrica

1. Cifrado de flujo
   - ChaCha20
   - Poly1305
   - AEAD_ChaCha20_Poly1305
2. Cifrado de bloque
   - DES
   - AES
   - Modos de operación
3. Lightweight Cryptography

Criptografía de clave secreta o simétrica

Se utiliza la misma clave para cifrar y descifrar.
La clave ha de mantenerse en secreto.

Cifrado de flujo: La transformación de cifrado varía de símbolo a símbolo. **ChaCha20**.

Cifrado de bloque: La transformación de cifrado se aplica a bloques del mensaje del mismo tamaño. *DES*, **AES**.

```
▾ Cipher Suites (17 suites)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
```

ChaCha20

Cifrado de flujo de clave de 256 bits: $c = m \oplus k_{\text{ext}}$.

- Salsa20 y ChaCha son cifrados de flujo desarrollados por Daniel J. Bernstein.
- Salsa20 se diseñó en 2005 y forma parte del portafolio eSTREAM.
- ChaCha es una modificación de Salsa20 realizada por el propio Bernstein en 2008. `TLS_CHACHA20_POLY1305_SHA256`

RFC 8439 ChaCha20 and Poly1305 for IETF Protocols
https://tools.ietf.org/html/rfc8439

D. Bernstein, "ChaCha, a variant of Salsa20", January 2008, http://cr.yp.to/chacha/chacha-20080128.pdf.

Chacha20

- A partir de una clave `Key` de 256 bits, tratada como la concatenación de 8 enteros de 32 bits (*unsigned little-endian*), un contador `Counter` de 32 bits (usualmente vale 1) y un `Nonce` de 96 bits, concatenación de 3 enteros de 32 bits (*unsigned little-endian*) se inicializa un estado que se visualiza como una matriz 4×4 de enteros de 32 bits:

(0) 0×61707865	(1) 0×3320646e	(2) 0×79622d32	(3) 0×6b206574
(4) Key	(5) Key	(6) Key	(7) Key
(8) Key	(9) Key	(10) Key	(11) Key
(12) Counter	(13) Nonce	(14) Nonce	(15) Nonce

- Este estado, incrementando el contador `Counter` cada vez que sea necesario*, se usa para generar una lista de bytes pseudo-aleatorios cuyo tamaño coincide con la del texto a cifrar.
- Estos bytes se combinan mediante un `XOR` con el mensaje en claro.

*Permite el acceso directo a los bytes cifrado.

Chacha20

Ejemplo

`Key` y `Nonce` son una secuencia de bytes.

```
Key = (00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:
       10:11:12:13:14:15:16:17:18:19:1a:1b:1c:1d:1e:1f)
Nonce = (00:00:00:09:00:00:00:4a:00:00:00:00)
Counter = 1
Estado:
0x61707865  0x3320646e  0x79622d32  0x6b206574
0x03020100  0x07060504  0x0b0a0908  0x0f0e0d0c
0x13121110  0x17161514  0x1b1a1918  0x1f1e1d1c
0x00000001  0x09000000  0x4a000000  0x00000000
```

0x61707865 0x3320646e 0x79622d32 0x6b206574 corresponde al texto ASCII "`expand 32-byte k`" tomado de 4 en 4 caracteres en orden *little-endian*.

Chacha20: QUARTERROUND

La operación básica es el QUARTERROUND que actúa sobre 4 palabras `a, b, c, d` de 32 bits modificándolas de la siguiente forma:

```
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<= 8;
c += d; b ^= c; b <<<= 7;
```

siendo $+$ suma módulo 2^{32}, $\hat{}$ XOR y $<<< n$ desplazamiento cíclico de n bits hacia la izquierda.

Chacha20

ChaCha20 consta de 20 rondas, alternando rondas aplicadas a columnas y rondas aplicadas a diagonales. Cada ronda consiste en 4 QUARTERROUND:

```
Column rounds:
     QUARTERROUND(0, 4, 8, 12)
     QUARTERROUND(1, 5, 9, 13)
     QUARTERROUND(2, 6, 10, 14)
     QUARTERROUND(3, 7, 11, 15)

Diagonal rounds:
     QUARTERROUND(0, 5, 10, 15)
     QUARTERROUND(1, 6, 11, 12)
     QUARTERROUND(2, 7, 8, 13)
     QUARTERROUND(3, 4, 9, 14)
```

Al final de las 20 rondas se le suma el estado inicial. El resultado, reordenado, k, es el que se combina con el mensaje en claro m:

$$c = m \oplus k$$

Chacha20: Difusión de cambios

Difusión de cambios

Poly1305

A partir de una clave (de un solo uso) de 256 bits y un mensaje genera una etiqueta, *tag* o *message authentication code (MAC)*, de 128 bits que se usa para autentificar el mensaje.

D. Bernstein, "The Poly1305-AES message-authentication code", March 2005, http://cr.yp.to/mac/poly1305-20050329.pdf.

Poly1305

- Sea el primo $p = 2^{130} - 5$.
- A partir de una clave k de 256 bits se generan dos subclaves r y s de 128 bits cada una.
- Se rompe el mensaje en bloques m_i de 16 bytes (salvo tal vez el último m_l) y se le añade $2^{8\cdot 16}$ (si el último tiene t bytes se le añade $2^{8\cdot t}$), sea $y_i = m_i + 2^{128}$ (salvo tal vez el último $y_l = m_l + 2^{8t}$)
- Sea $a = 0$, entonces

$$a = (a + y_i)\, r \mod p,\ i = 1...l$$

- Por último $tag = a + s \mod 2^{128}$.

Cuando se usa conjuntamente con Chacha20 la clave k es el resultado de aplicar aplicar Chacha20 a un estado con la `Key` y `Nonce` dados pero usando como `Counter` inicial el valor 0.

Reducción módulos especiales, p.e. $p = 2^{192} - 2^{64} - 1$

Si $x < p^2$, $x = x_5 2^{320} + x_4 2^{256} + x_3 2^{192} + x_2 2^{128} + x_1 2^{64} + x_0$, siendo $x_i < 2^{64}$.

Teniendo en cuenta

$$\begin{aligned} 2^{192} &\equiv 2^{64} + 1 \mod p \\ 2^{256} &\equiv 2^{128} + 2^{64} \mod p \\ 2^{320} &\equiv 2^{192} + 2^{128} \mod p \end{aligned}$$

tenemos

$$\begin{aligned} x &\equiv x_4 2^{256} + (x_5 + x_3)2^{192} + (x_5 + x_2)2^{128} + x_1 2^{64} + x_0 \mod p \\ x &\equiv (x_5 + x_3)2^{192} + (x_5 + x_4 + x_2)2^{128} + (x_4 + x_1)2^{64} + x_0 \mod p \\ x &\equiv (x_5 + x_4 + x_2)2^{128} + (x_5 + x_4 + x_3 + x_1)2^{64} + x_5 + x_3 + x_0 \mod p \end{aligned}$$

Si fuera necesario, $x \equiv p - (x_5 + x_4 + x_2)2^{128} + (x_5 + x_4 + x_3 + x_1)2^{64} + x_5 + x_3 + x_0 \mod p$

Authenticated encryption with additional data (AEAD)

Es una construcción que asegura confidenciabilidad de los datos e integridad del criptograma.

Entrada: los datos a cifrar, unos datos asociados de tamaño variable –additional authenticated data (AAD)–, un `nonce` de tamaño fijo y la clave también de tamaño fijado

Salida: los datos cifrados y una etiqueta que permite comprobar la integridad.

AEAD ChaCha20 Poly1305

```
def chacha20_aead_encrypt_rfc(aad, key, nonce, plaintext):
'''
    aad: bytearray
    key: bytearray de 32 bytes (256 bits, 8 palabras (enteros) de 32 bits)
    nonce: bytearray
    plaintext: bytearray
    '''

    ciphertext = chacha20_encrypt(key, 1, nonce, plaintext)

    otk = poly1305_key_gen(key, nonce)

    mac_data = b''.join([aad, pad16(aad)])
    mac_data = b''.join([mac_data, ciphertext, pad16(ciphertext)])
    mac_data = b''.join([mac_data, len(aad).to_bytes(8,byteorder='little')])
    mac_data = b''.join([mac_data, len(ciphertext).to_bytes(8,byteorder='little')])

    tag = poly1305_mac(mac_data, otk)

    return (ciphertext, tag)
```

AEAD ChaCha20 Poly1305

```
def chacha20_aead_decrypt_rfc(aad, key, nonce, ciphertext, mac):
    '''
    aad: bytearray
    key: bytearray de 32 bytes (256 bits, 8 palabras (enteros) de 32 bits)
    nonce: bytearray
    ciphertext: bytearray
    '''

    otk = poly1305_key_gen(key, nonce)

    mac_data = b''.join([aad, pad16(aad)])
    mac_data = b''.join([mac_data, ciphertext , pad16(ciphertext)])
    mac_data = b''.join([mac_data, len(aad).to_bytes(8,byteorder='little')])
    mac_data = b''.join([mac_data, len(ciphertext).to_bytes(8,byteorder='little')])

    tag = poly1305_mac(mac_data, otk)

    if tag==mac:
        plaintext = chacha20_encrypt(key, 1, nonce, ciphertext)
        return (plaintext, tag==mac)
    else:
        return False
```

Data Encryption Standard (DES): Historia (I)

Cifrado de bloques de 64 bits y clave de 56 bits.

1973 El NBS (National Bureau of Standards) solicita públicamente propuestas de sistemas criptográficos, que deben cumplir los criterios:

- Proporcionar un alto nivel de seguridad y ser eficiente.
- Residir la seguridad en la clave y no en el secreto del algoritmo.
- Ser adaptable para ser utilizado en diversas aplicaciones.
- Ser barato de implementar en dispositivos electrónicos.

1974 Segundo llamamiento: IBM presenta LUCIFER. La NSA (National Security Agency) propone una serie de cambios que son aceptados.

1975 El 17 de marzo el NBS publica los detalles del DES.

1976 El 23 de noviembre es adoptado por el gobierno USA para la transmisión y almacenamiento de información no clasificada. Se revisará cada cinco años.

1981 Diversos organismos privados lo adoptan como estándar.

1983 Se ratifica como estándar sin problemas.

DES: Historia (y II)

1987 La NSA se opone a una nueva ratificación pero, por motivos económicos, finalmente se renueva.

1992 Por falta de alternativas se renueva otra vez.

1997 El 17 de junio el DES es roto. A principios de año RSA lanza el reto y al cabo de cuatro meses es alcanzada la solución después de examinar, aproximadamente, el 25% de las claves.

1998 El 26 de febrero el DES vuelve a ser roto. Sólo han sido necesarios 39 días y se han examinado, aproximadamente, el 85% de las claves.

1998 El 17 de julio la Electronic Frontier Foundation (EFF) presenta su DES craker que puede romper el DES utilizando la fuerza bruta en un tiempo medio de 4.5 días. Su coste: 220000\$.†

1999 El 19 de enero la EFF rompe el DES en menos de 23 horas.

2001 Es sustituido por el AES (Advanced encryption standard), aunque se mantiene el 3DES.

†Actualmente se puede construir una máquina que rompa el DES en un día por unos pocos miles de euros.

DES: Descripción del algoritmo a alto nivel (I)

1. Dado un bloque x, se le aplica una permutación inicial π,

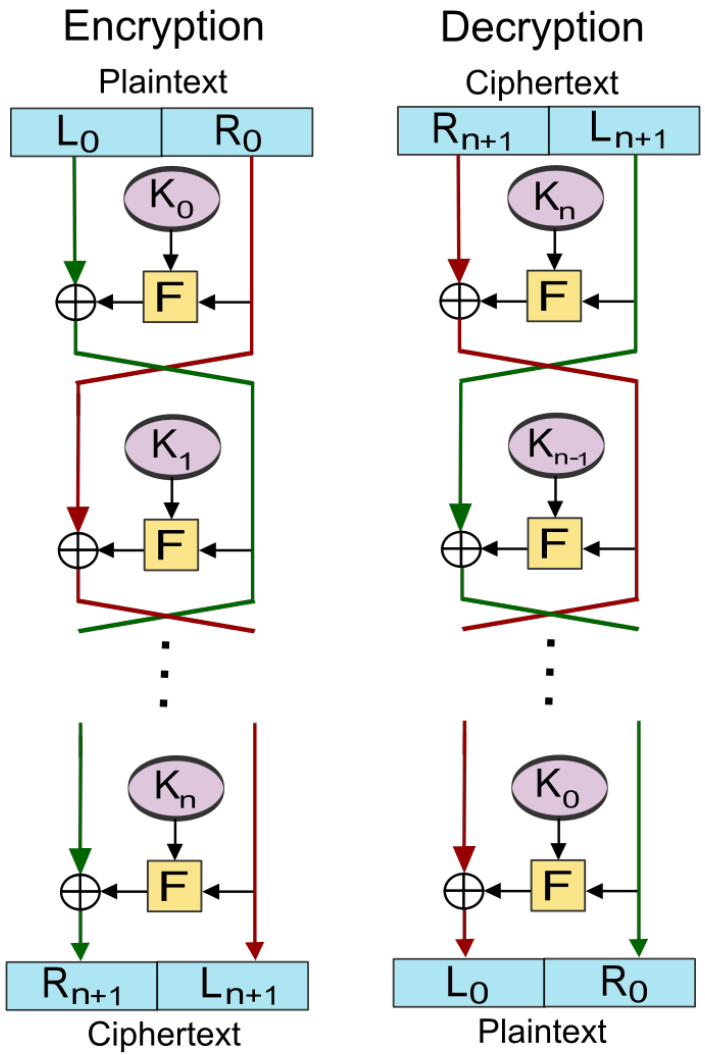
$$x_0 = \pi(x) \equiv L_0 R_0,$$

2. Se realizan 16 iteraciones del tipo *(Feistel cipher)*:

$$L_i = R_{i-1},$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i),$$

3. Se aplica la permutación inversa de π a $R_{16}L_{16}$,

$$y = \pi^{-1}(R_{16}L_{16}).$$



DES: Descripción del algoritmo a alto nivel (y II)

Si definimos:

- θ función que intercambia parte derecha e izquierda de un bloque de 64 bits,
- $\lambda_{f_i} : x \longrightarrow y$

$$L_y = L_x \oplus f(R_x, k_i),$$
$$R_y = R_x,$$

entonces el DES se puede escribir en forma más compacta como

$$\pi^{-1}\lambda_{f_{16}}\theta\lambda_{f_{15}}\theta\lambda_{f_{14}}\theta...\theta\lambda_{f_3}\theta\lambda_{f_2}\theta\lambda_{f_1}\pi,$$

y para descifrar

$$\pi^{-1}\lambda_{f_1}\theta\lambda_{f_2}\theta\lambda_{f_3}\theta...\theta\lambda_{f_{14}}\theta\lambda_{f_{15}}\theta\lambda_{f_{16}}\pi.$$

DES: Seguridad

1. Tamaño de la clave: Se considera muy pequeño.
2. Tamaño del bloque: Se considera pequeño.
3. Las razones de la elección de las S-box no son públicas.
4. No se conoce ninguna técnica de criptoanálisis para atacarlo más eficiente que la fuerza bruta.
5. El criptoanálisis diferencial permite atacar versiones débiles del DES.
6. El DES, *muy probablemente*, no tiene estructura de grupo. Se puede aumentar la seguridad mediante aplicaciones sucesivas del DES con distintas claves.
   Dos variantes son el 3DES[‡] ($E_{k_1} D_{k_2} E_{k_3}$) y la función `crypt(3)`.

[‡] Update to Current Use and Deprecation of 3DES (11/07/2017)

CRYPT(3)

Función, basada en el DES, encargada de cifrar los passwords.[§]
Introduce el *salt*, 12 bits, que se usa para permutar bits.

```
hobbes:1gDPklM/j4y.i:504:100:Calvin & Hobbes:/home/hobbes:/bin/bash
```

El objetivo del *salt* es dificultar ataques de diccionario:

- **John the Ripper** http://www.openwall.com/john/
- **Rainbow Tables**
  - *Making a Faster Cryptanalytic Time-Memory Trade-Off.* Philippe Oechslin https://infoscience.epfl.ch/record/99512
  - OPHCRACK (the time-memory-trade-off-cracker) https://ophcrack.sourceforge.io/
  - Project RainbowCrack: https://project-rainbowcrack.com/

have i been pwned?

[§] Ahora cada entrada en el fichero `/etc/shadow` puede ser de la forma idsalt$encrypted, id identifica el algoritmo usado: 1→MD5, 5→SHA256, 6→SHA512

Passwords: Recomendaciones (I)

NIST Special Publication 800-63B. Digital Identity Guidelines (`memorized secrets` se refiere a contraseñas):

`Verifiers SHOULD NOT impose other composition rules (e.g., requiring mixtures of different character types or prohibiting consecutively repeated characters) for memorized secrets.`

`Verifiers SHOULD NOT require memorized secrets to be changed arbitrarily (e.g., periodically). However, verifiers SHALL force a change if there is evidence of compromise of the authenticator.`

Passwords: Recomendaciones (II)

When processing requests to establish and change memorized secrets, verifiers SHALL compare the prospective secrets against a list that contains values known to be commonly-used, expected, or compromised. For example, the list MAY include, but is not limited to:

- Passwords obtained from previous breach corpuses.
- Dictionary words.
- Repetitive or sequential characters (e.g. 'aaaaaa', '1234abcd').
- Context-specific words, such as the name of the service, the username, and derivatives thereof.

If the chosen secret is found in the list, the CSP or verifier SHALL advise the subscriber that they need to select a different secret, SHALL provide the reason for rejection, and SHALL require the subscriber to choose a different value.

Passwords: Recomendaciones (III)

> Verifiers SHALL store memorized secrets in a form that is resistant to offline attacks. Memorized secrets SHALL be salted and hashed using a suitable one-way key derivation function. Key derivation functions take a password, a salt, and a cost factor as inputs then generate a password hash. Their purpose is to make each password guessing trial by an attacker who has obtained a password hash file expensive and therefore the cost of a guessing attack high or prohibitive.

La realidad nos muestra que no siempre se siguen estas indicaciones:
Descubierta una vulnerabilidad en Pegasus Airlines que filtró 6,5 TB de datos alojados en AWS

AES: Advanced Encryption Standard (I)

El 12 de septiembre de 1997 el departamento de comercio del National Institute of Standars and Technology (NIST), antiguo NBS, hace un llamamiento público para la presentación de algoritmos candidatos al Advanced Encryption Standard (AES).

Requisitos mínimos

1. El algoritmo debe ser de clave secreta (simétrico).
2. El algoritmo debe ser un algoritmo de bloque.
3. El algoritmo debe ser capaz de soportar las combinaciones clave-bloque de los tamaños 128-128, 192-128 y 256-128.

AES: Advanced Encryption Standard (y II)

Criterios de evaluación

1. Seguridad: Es el factor más importante a la hora de evaluar los candidatos.
2. Coste:
   1. El algoritmo debe ser accesible a todo el mundo y de libre distribución.
   2. El algoritmo debe ser computacionalmente eficiente tanto en *hardware* como en *software*.
   3. El algoritmo debe utilizar la menor memoria posible tanto en *hardware* como en *software*.
3. Características de implementación del algoritmo:
   1. El algoritmo debe ser fácilmente implementable en distintas plataformas tanto en *hardware* como en *software*.
   2. El algoritmo debe acomodarse a diferentes combinaciones clave-bloque además de las mínimas requeridas.
   3. El algoritmo debe ser de diseño simple.

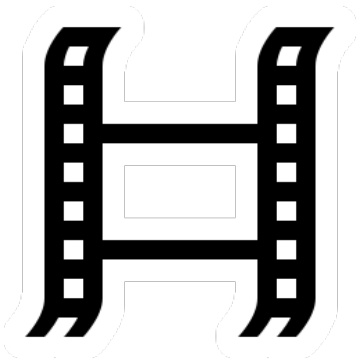
Candidatos al AES

1. **CAST-256**, Entust Technologies, Inc. (C. Adams).
2. **CRYPTON**, Future Systems, Inc. (Chae Hoon Lim).
3. **DEAL**, L. Knudsen, R. Outerbridge.
4. **DFC**, CNRS-Ecole Normale Superiere (S. Vaudenay).
5. **E2**, NTT *Nippon Telegraph and Telephone Corporation* (M. Kanda).
6. **FROG**, TecApro International S.A. (D. Georgoudis, Leroux, Chaves).
7. **HPC**, R. Schoeppel.
8. **LOKI97**, L. Brown, J. Pieprzyk, J.Seberry.
9. ~~**MAGENTA**~~, Deutshe Telekom AG (K. Huber).
10. **MARS***, IBM (N. Zunic).
11. **RC6***, RSA Laboratories (Rivest, M. Robshaw, Sidney, Yin).
12. **RIJNDAEL***, J. Daemen¶, V. Rijmen.
13. **SAFER+**, Cylink Corporation (L. Chen).
14. **SERPENT***, R. Anderson, E. Biham, L. Knudsen.
15. **TWOFISH***, B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson.

¶También participó en el diseño de SHA3.

RIJNDAEL-AES: Advanced Encryption Standard

Cifrado simétrico de bloque de 128 bits y clave de 128, 192 o 256 bits.[‖]



FIPS 197 Advanced Encryption Standard (AES)

Shay Gueron (2010). Intel® Advanced Encryption Standard (AES) New Instructions Set

[‖] RIJNDAEL permite también el uso de bloques de 192 o 256 bits.

RIJNDAEL-AES: Advanced Encryption Standard

Opera con bytes, $\mathrm{GF}(2^8)$ (polinomio irreducible $x^8+x^4+x^3+x+1$), y con palabras de 4 bytes, polinomios con coeficientes en $\mathrm{GF}(2^8)$.

$\mathrm{GF}(2^8)$ generado por 03=$x+1$

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

RIJNDAEL-AES: Advanced Encryption Standard

- N_b número de bits del bloque dividido por 32.
  N_k número de bits de la clave dividido por 32.
- El número de rondas, N_r, depende de la longitud de la clave y del bloque.

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

- Las diferentes transformaciones actúan sobre un resultado intermedio, `State`, formado por una matriz $4 \times N_b$ de bytes:

$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$\ldots$
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$\ldots$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$\ldots$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$\ldots$

RIJNDAEL-AES: Descripción del algoritmo de cifrado a alto nivel

1. `AddRoundKey(State, RoundKey`$_0$`)`
2. `Round(State, RoundKey`$_i$`)`, $i = 1, \ldots, N_r - 1$:
   1. `ByteSub(State)`
   2. `ShiftRow(State)`
   3. `MixColumn(State)`
   4. `AddRoundKey(State, RoundKey`$_i$`)`
3. `FinalRound(State, RoundKey`$_{N_r}$`)`:
   1. `ByteSub(State)`
   2. `ShiftRow(State)`
   3. `AddRoundKey(State, RoundKey`$_{N_r}$`)`

RIJNDAEL-AES: ByteSuB

Transformación no lineal de sustitución de bytes (S-box).

1. Toma el inverso multiplicativo en $\mathrm{GF}(2^8)$.
2. Aplica la transformación afín sobre $\mathrm{GF}(2)$:

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

RIJNDAEL-AES: `ShiftRow`

Las filas de `State` se desplazan cíclicamente, la primera no sufre desplazamiento, la segunda se desplaza C_1 posiciones, la tercera C_2 y la cuarta C_3:

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$\ldots$
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$\ldots$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$\ldots$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$\ldots$

$\Longrightarrow$

$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$\ldots$
$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$\ldots$	$m_{1,0}$
$m_{2,2}$	$m_{2,3}$	$\ldots$	$m_{2,0}$	$m_{2,1}$
$m_{3,3}$	$\ldots$	$m_{3,0}$	$m_{3,1}$	$m_{3,2}$

RIJNDAEL-AES: `MixColumn`

Las columnas de `State` son consideradas polinomios sobre $\mathrm{GF}(2^8)$ y multiplicadas módulo x^4+1 por el polinomio:

$$c(x) = \texttt{0x03}\, x^3 + \texttt{0x01}\, x^2 + \texttt{0x01}\, x + \texttt{0x02}.$$

Si $b(x) = c(x) \otimes a(x)$,

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

RIJNDAEL-AES: `AddRoundKey`

Consiste en un `XOR` entre `State` y `RounKey`.

$$\texttt{State} \oplus \texttt{RounKey}$$

RIJNDAEL-AES: Generación de subclaves (I)

- La clave se extiende a una lista de palabras de 4 bytes que llamaremos W y que contiene $N_b(N_r+1)$ palabras.
- Los primeros N_k elementos de W corresponden a la clave.
- El resto de los elementos de W se definen recursivamente utilizando la función `SubByte`, desplazamientos cíclicos y $\oplus$.
- Usa la función `RotByte` que devuelve una palabra cuyos bytes se han desplazado cíclicamente una posición.
- Utiliza unas constantes cuyos valores son

$$\texttt{Rcon}[i] = (RC[i], \texttt{0x00}, \texttt{0x00}, \texttt{0x00}),$$

siendo $RC[i]$ un elemento de $\mathrm{GF}(2^8)$ definido por

$$RC[1] = \texttt{0x01}, \qquad RC[i] = \texttt{0x02} \bullet RC[i-1].$$

RIJNDAEL-AES: Generación de subclaves (y II)

$N_k \leq 6$

```
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 = SubByte(RotByte(temp))^Rcon[i/Nk];


   W[i] = W[i - Nk] ^ temp;
  }
}
```

$N_k > 6$

```
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 = SubByte(RotByte(temp))^Rcon[i/Nk];
   else if (i % Nk == 4)
       temp = SubByte(temp);
   W[i] = W[i - Nk] ^ temp;
 }
}
```

RIJNDAEL-AES: Difusión de cambios

Difusión de cambios.

RIJNDAEL-AES: Un algoritmo de descifrado (I)

1. AddRoundKey(State, InvRoundKey_{N_r})
2. Round(State, InvRoundKey_i), $i = N_r - 1, \ldots, 1$:
   1. InvByteSub(State)
   2. InvShiftRow(State)
   3. InvMixColumn(State)
   4. AddRoundKey(State, InvRoundKey_i)
3. FinalRound(State, InvRoundKey_{N_0}):
   1. InvByteSub(State)
   2. InvShiftRow(State)
   3. AddRoundKey(State, InvRoundKey_{N_0})

RIJNDAEL-AES: Un algoritmo de descifrado (II)

- Las funciones `InvByteSub`, `InvShiftRow`, `InvMixColumn` son las inversas de `ByteSub`, `ShiftRow`, `MixColumn` respectivamente.
- Las subclaves `InvRoundKey` vienen dadas por:
  - $\texttt{InvRoundKey}_0 = \texttt{RoundKey}_0$
  - $\texttt{InvRoundKey}_i = \texttt{InvMixColumn}(\texttt{RoundKey}_i)$, $i = 1, \ldots, N_r - 1$
  - $\texttt{InvRoundKey}_{N_r} = \texttt{RoundKey}_{N_r}$

RIJNDAEL-AES: Un algoritmo de descifrado (y III)

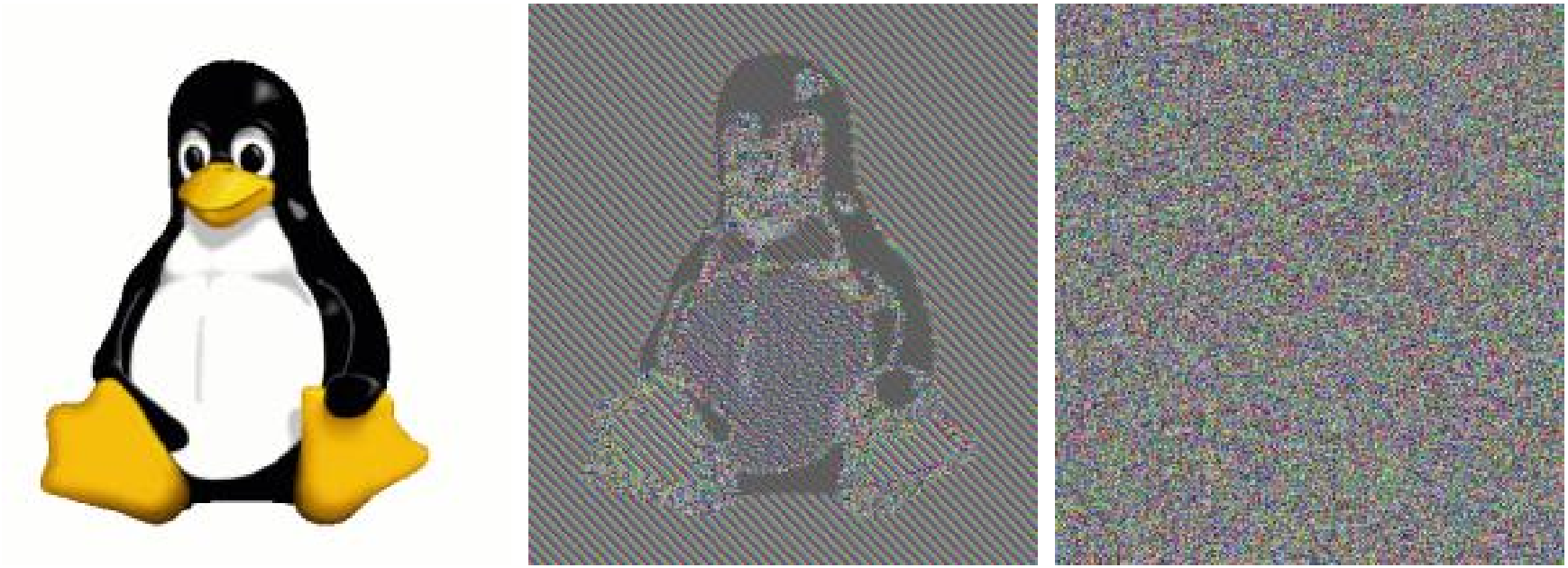
- **InvByteSub**. Si $\mathbf{y} = \mathbf{Ax} + \mathbf{B}$, invirtiendo $\mathbf{x} = \mathbf{A}^{-1}(\mathbf{y} - \mathbf{B})$. Después se ha de sustituir cada byte diferente de `0x00` por su inverso en $GF(2^8)$.

$$\mathbf{A^{-1}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

- **InvShiftRow**. Se han de rotar las filas el mismo número de posiciones pero en sentido contrario.
- **InvMixColumn**. Se multiplica por el polinomio inverso del anterior $d(x) = \texttt{0x0B}\, x^3 + \texttt{0x0D}\, x^2 + \texttt{0x09}\, x + \texttt{0x0E}$. Matricialmente:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \texttt{0x0E} & \texttt{0x0B} & \texttt{0x0D} & \texttt{0x09} \\ \texttt{0x09} & \texttt{0x0E} & \texttt{0x0B} & \texttt{0x0D} \\ \texttt{0x0D} & \texttt{0x09} & \texttt{0x0E} & \texttt{0x0B} \\ \texttt{0x0B} & \texttt{0x0D} & \texttt{0x09} & \texttt{0x0E} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

Modos de operación



(a) Original[1] (b) ECB (c) Otro

*Tux the Penguin. Created in 1996 by Larry Ewing with The GIMP.

Modos de operación[††] (I)

ECB **Electronic CodeBook**

$$c_i = E_k(m_i); \qquad m_i = D_k(c_i).$$

CBC **Cipher Block Chaining** Se inicializa c_0 aleatorio,

$$c_i = E_k(m_i \oplus c_{i-1}); \qquad m_i = D_k(c_i) \oplus c_{i-1}.$$

1. c_0 (o `IV` o `Nonce`) puede ser público. Variar c_0 permite obtener diferentes resultados cifrando el mismo mensaje con la misma clave.
2. Bloques iguales van a criptogramas diferentes.
3. Un error en la transmisión afecta al descifrar a dos bloques del mensaje.
4. Puede usarse como MAC[**], variar un bit en un bloque del mensaje afecta al resto de los bloques cifrados.

[**] https://csrc.nist.gov/publications/detail/sp/800-38b/final

[††] NIST SP 800-38A: Recommendation for Block Cipher Modes of Operation

Modos de operación (II)

CFB **Cipher FeedBack** Se inicializa c_0 aleatorio,

$$c_i = m_i \oplus E_k(c_{i-1});$$

$$m_i = c_i \oplus E_k(c_{i-1}).$$

Además de las propiedades del modo CBC:

1. Se utiliza la misma función para cifrar y descifrar.
2. No es necesario *padding*.

OFB **Output FeedBack** Se inicializa s_0 aleatorio,

$$c_i = m_i \oplus E_k(s_{i-1}), \qquad s_i = E_k(s_{i-1}));$$

$$m_i = c_i \oplus E_k(s_{i-1}).$$

1. Un error en la transmisión afecta al descifrar a un bloque del mensaje.
2. Se utiliza la misma función para cifrar y descifrar.
3. Se puede usar como cifrado de flujo.

Modos de operación (III)

CTR **Counter Mode** Dados $T_1, \ldots, T_n$

$$c_i = m_i \oplus E_k(T_i);$$

$$m_i = c_i \oplus E_k(T_i).$$

1. Se utiliza la misma función para cifrar y descifrar.
2. No es necesario *padding*.
3. Los T_i deben ser diferentes para cada bloque y mensaje.
4. Los T_i se pueden generar como se quieran (sin repeticiones).
5. Se pueden precalcular los valores $E_k(T_i)$.
6. Se puede usar como cifrado de flujo.
7. Se pueden descifrar bloques del mensaje de forma independiente.
8. Es paralelizable.

Modos de operación (y IV)

GMAC **Galois Message Authentication Code**[‡‡]

Sea $y_0 = 0 \dots 0$ un bloque de 128 bits y H la clave de autentificación.

$$y_i = (m_i \oplus y_{i-1}) \bullet H$$

El símbolo $\bullet$ representa la multiplicación en $\mathrm{GF}(2^{128})$ (polinomio irreducible $x^{128} + x^7 + x^2 + x + 1$).

[‡‡] Asociado al CTR como método de autentificación.

Lightweight Cryptography

https://csrc.nist.gov/projects/lightweight-cryptography

There are several emerging areas (e.g. sensor networks, healthcare, distributed control systems, the Internet of Things, cyber physical systems) in which highly-constrained devices are interconnected, typically communicating wirelessly with one another, and working in concert to accomplish some task. Because the majority of current cryptographic algorithms were designed for desktop/server environments, many of these algorithms do not fit into constrained devices.

Lightweight Cryptography

- **AEAD Requirements** Authenticated decryption, also known as decryption-verification, shall be supported: it shall be possible to recover the plaintext from a valid ciphertext (i.e., a ciphertext that corresponds to the plaintext for a given associated data, nonce, and key), given associated data, nonce and key. Plaintext shall not be returned by the decryption-verification process if the ciphertext is invalid.[...]

  The family shall include one primary member that has a key length of at least 128 bits, a nonce length of at least 96 bits, and a tag length of at least 64 bits. The limits on the input sizes (plaintext, associated data, and the amount of data that can be processed under one key) for this member shall not be smaller than 2^{50}-1 bytes.
- **Hash Function Requirements** Hash functions shall accept all byte-string inputs that meet the specified maximum length of messages. Submissions shall include justification for any length limits. The family shall include one primary member that has an output size of at least 256 bits. The limit on the message size for this member shall not be smaller than 2^{50}-1 bytes.

Lightweight Cryptography

- **NIST Released Draft NISTIR 8114,** August 11, 2016. Report on Lightweight Cryptography is available for public comment.
- **Lightweight Crypto Draft Requirements and Evaluation Criteria**, May 14, 2018.
- **Request for Nominations for Lightweight Cryptographic Algorithms**, August 27, 2018.
- **Round 1 Candidates** In March 2019, NIST received 57 submissions to be considered for standardization. After the initial review of the submissions, 56 were selected as Round 1 Candidates.
- **Round 2 Candidates** Of the 56 Round 1 candidates, 32 were selected to continue to Round 2. August 2019.
- **Finalists** On March 29, 2021, NIST announced ten finalists as ASCON, Elephant, GIFT-COFB, Grain128-AEAD, ISAP, Photon-Beetle, Romulus, Sparkle, TinyJambu, and Xoodyak
- **NIST Selects Ascon** February 07, 2023.

  NIST IR 8454: Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process

Ascon

- Ascon: Lightweight Authenticated Encryption & Hashing: Ascon is a family of authenticated encryption and hashing algorithms designed to be lightweight and easy to implement, even with added countermeasures against side-channel attacks. Ascon has been selected as new standard for lightweight cryptography in the NIST Lightweight Cryptography competition (2019–2023). Ascon has also been selected as the primary choice for lightweight authenticated encryption in the final portfolio of the CAESAR competition (2014–2019).
- Ascon: Specification