Seguridad en Sistemas de Información

Curso en Tamaulipas [Q2 2014]

Luis J. Dominguez Perez

Cinvestav, Mayo 26 de 2014 - L2

Contenido, sección 1

Criptografía simétrica

- DES
- Modos de operación
- AES

DES - Data Encryption Standard

Historia:

- Las compañías financieras necesitaban de un mecanismo de protección que tuviese el visto bueno del gobierno norteamericano.
- El primer llamado para el concurso fue en mayo de 1973, seguido de un segundo llamado en 1974
- No hubo muchas propuestas. IBM presentó LUCIFER.
- El gobierno trabajó con la IBM para rediseñar el algoritmo

DES - Data Encryption Standard

Cronograma:

- 1973 - el NBS (El Buró nacional de estándares de los EEUU, ahora NIST) solicita propuestas de criptosistemas para documentos "no-clasificados"
- 1974 - el NBS repite el requerimiento.
  - IBM responde con una modificación de LUCIFER.
    LUCIFER está basado en una familia de cifradores creados por Horst Feistel a finales de los 1960s.
  - NBS le pide a la NSA (National Security Agency) que lo evalúe.
    Por esas fechas la NSA negaba su propia existencia.
  - La NSA convence a la IBM que reduzca el tamaño estándar de la clave del LUCIFER de 128-bits a 56-bits, esto posibilitaría los ataques por fuerza bruta.
  - . . .

DES

- . . .
  - LA NSA escogió algunos de los parámetros del sistema del cifrador.
    En particular pidió que se agregara soporte a un tipo de ataques en particular. Estos ataques se descubrirían en los 1990s, y se llamarían ataques diferenciales. Se desconoce si la NSA sabía de su existencia, o si sólo sospechaban.
  - IBM obtiene la patente del DES (Data Encryption Standar).
- 1975 se publican los detalles del algoritmo, excepto algunos criterios de funciones internas. La discusión pública comienza. La gente tenía la incertidumbre si la NSA le había metido mano para su beneficio (introducción de trap-doors).
- . . .

DES

- 1976 se adopta como un estándar para todos los doumentos gubernamentales “no-clasificados”.
  Data Encryption Standard - FIPS PUB 46.
- Se hace estándar para hardware en 1977
- ANSI X3.92-1981 (hardware + software)
- ANSI X3.106-1983 (modes of operation)
- Australia AS2805.5-1985
- Se reafirmó como estándar hasta 1999 cuando se substituyó por el AES - Advanced Encryption Standard.

Confusión y difusión

De acuerdo con el teorista de la información Claude Shannon, hay dos operaciones primitivas con las cuales los algoritmos de cifrado fuerte pueden ser construídos:

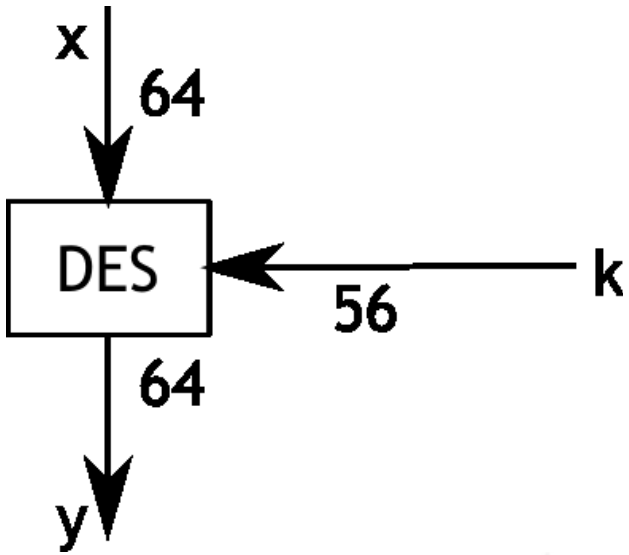
- **Confusión**. Operación de cifrado en la que la relación entre la llave y el texto cifrado se ocultan.
- **Difusión**. Operación de cifrado en la que la influencia de un símbolo del texto en claro se reparte sobre muchos del texto cifrado para ocultar las propiedades estadísticas del texto en claro; i.e., permutaciones y mezcla columnas.

Confusión y difusión

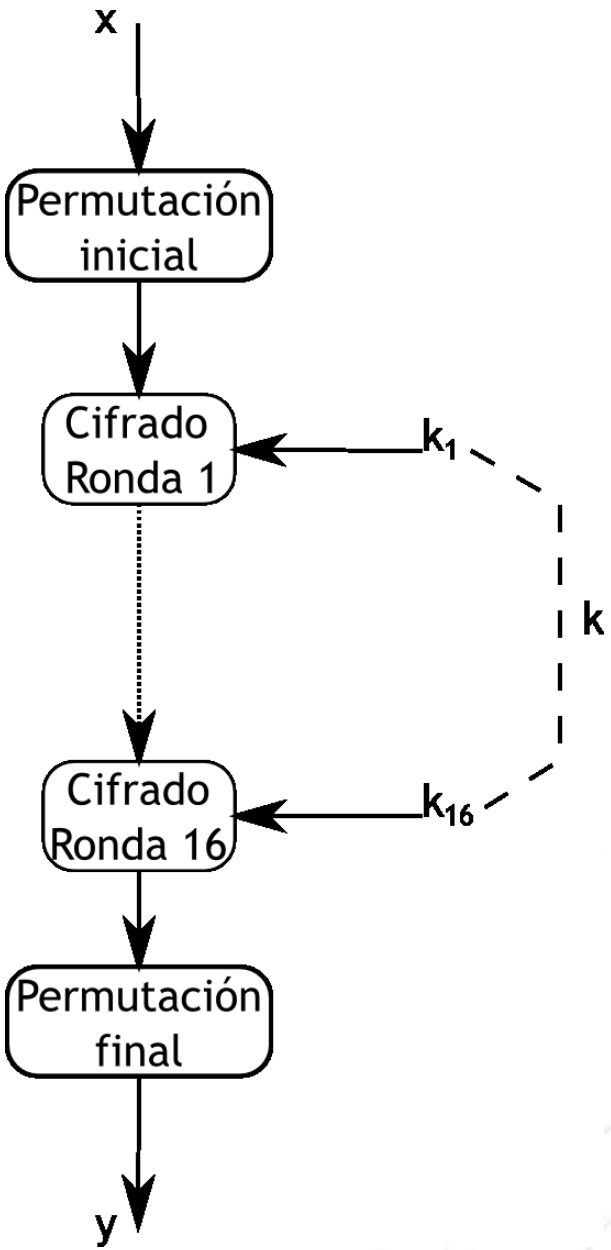
- Utilizar sólo confusión o sólo difusión resulta en cifradores que no son seguros.
- A través de la concatenación de dichas primitivas se construyen cifradores fuertes; estos se conocen como cifradores de producto.
- Todos los cifradores modernos y seguros son cifradores de producto.
- Una secuencia de operaciones de confusión y difusión se le conoce como ronda.
- Las rondas se repiten n veces sobre el texto en claro o el resultante de la ronda anterior. Esto provoca un efecto avalancha.

DES caja negra

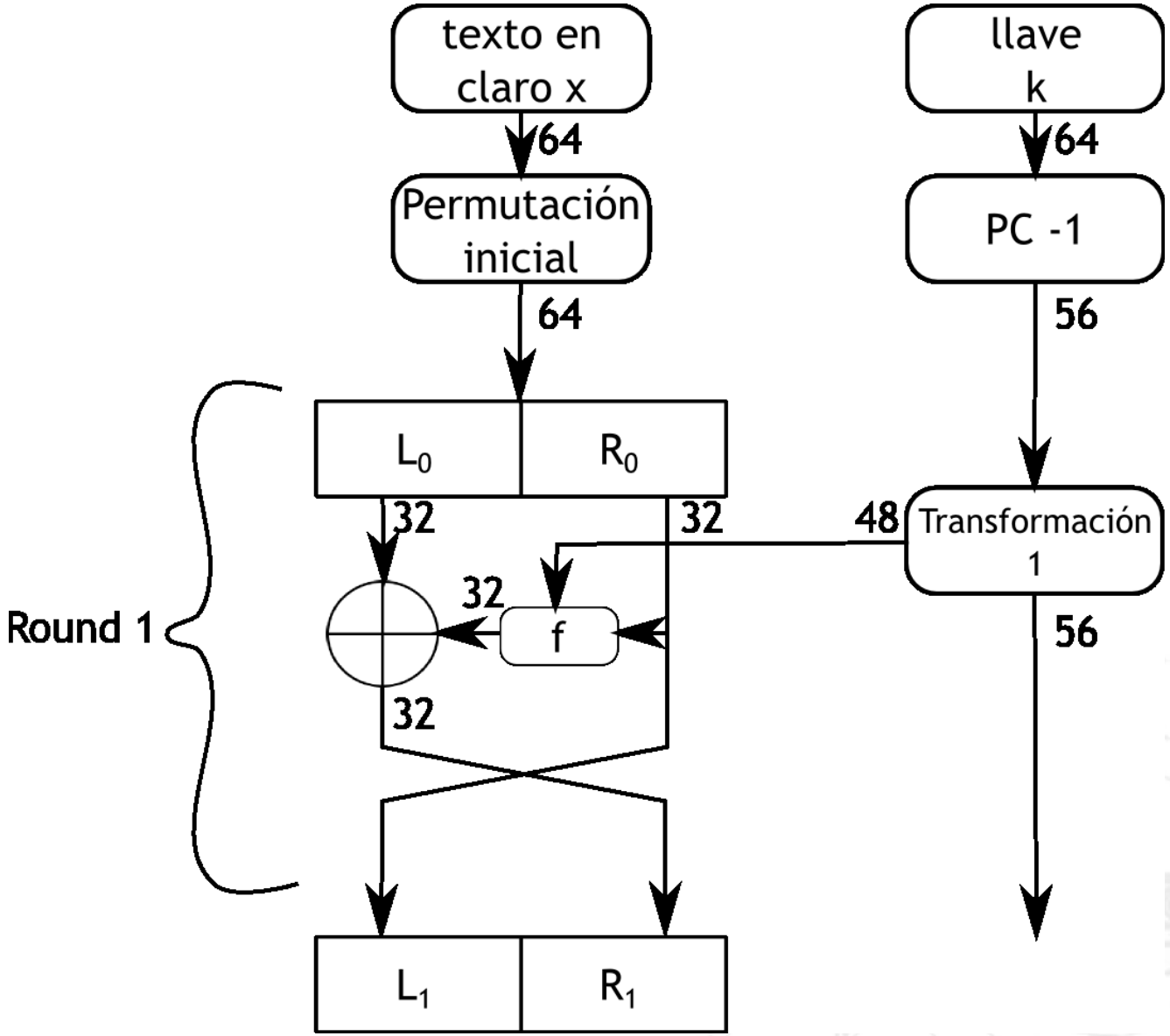
El DES es un cifrador que toma bloques de 64-bits de longitud con una llave de 56-bits.



DES estructura general



DES estructura Feistel



DES en la práctica

- El DES fué utilizado ampliamente por el gobierno de los EEUU y por la industria para proteger las comunicaciones.
- Una aplicación importante es en las transacciones bancarias. Por ejemplo, el DES se utilizaba para cifrar los números NIP y las operaciones de los cajeros automáticos (ATM)

DES en la práctica

- Aunque la descripción del algoritmo del DES es larga, se puede implementar eficientemente en la práctica: la únicas operaciones aritméticas utilizadas son sumas módulo 2 (que es un XOR). La función de expansión E, las cajas S-box, las permutaciones fijas, y los cálculos con las llaves pueden ser implementadas mediante tablas de búsqueda (lookup tables) en software, o mediante circuitería fija en hardware.
- DES puede ser hasta 500 veces más rápida que su equivalente en sistemas asimétricos (de hasta 512-bits).

Controversia DES

Longitud de la llave. Se tiene un gran problema con la longitud de la llave.

- DES fue desarrollado a partir de LUCIFER, el cual utilizaba una llave de 128-bits. La longitud fue reducida a 56-bits posibilitando los ataques de conocimiento de texto en claro por búsqueda exhaustiva.

Ataques

- 1977 - Diffie y Hellman sugirieron un diseño de un chip VLSI que puede probar 10^6 llaves/seg. Un equipo con 10^6 de estos circuitos podría romper la clave en cuestión de 10 horas. Costo: $\$20$ mill. USD
- 1990 - Eli Biham y Adi Shamir sugirieron un criptoanálisis diferencial
- 1993 - Mitsuru Masui sugirió un criptoanálisis linear

Más controversias

Diseño S-box. Todo proceso de cifrado, exceptuando el cómputo de las cajas S-Box es linear, y los sistemas lineares no son muy seguros.

- La seguridad de DES depende principalmente de las cajas S-box. Cuando el Buró Nacional de Estándares de los EEUU publicó el concurso del DES en 1973, especificó ciertos criterios, entre los cuales se encuenta que las cajas S-box fueran una permutación de $0, 1, \ldots, 15$, pero *no* que fueran lineares o una función afín a las entradas.

- Sin embargo, el criterio de diseño de las cajas S-box es oscuro. Se ha sugerido que contienen *trapdoors* escondidas que permiten a la NSA (Agencia de Seguridad Nacional de los EEUU) descifrar secretamente mensajes.
- Por otro lado, no hay evidencia que soporte estas afirmaciones, aun y cuando gente ha pasado mucho tiempo buscándolas.

Variante

Cifrado múltiple.

- Mucha gente pensó que DES se hubiera podido hacer más seguro al utilizar doble cifrado con dos llaves.
- Esto es obviamente no más seguro que DES.
- Existe una variante, Triple DES (utilizando DES con diferentes llaves) sí es más seguro que DES.

Descripción DES

- DES cifra una cadena de bits de un texto en plano de 64 bits, obteniendo un texto cifrado de 64 bits.
  1. Dada una cadena x de texto en plano, se aplica una permutación fija IP a x:
  $$x_0 = \mathsf{IP}(x) = L_0 R_0,$$
  donde L_0 es la parte baja de 32 bits, y R_0 la parte alta.
  2. Se calculan 16 iteraciones de una función. Calcule $L_i R_i$ para $1 \leq i \leq 16$ de acuerdo a:
  $$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1,K_i}), \end{aligned}$$
  en donde $\oplus$ denota adición módulo 2. $K_1, K_2, \ldots, K_{16}$ son las cadenas de 48 bits de la función de calendarización de la llave K.
  3. Se aplica la función de permutación inversa IP^{-1} a la cadena $R_{16}L_{16}$, obteniendo el texto cifrado:
  $$y = \mathsf{IP}^{-1}(R_{16}L_{16}).$$

Descripción DES

La función f

- la función f tiene como entrada dos cadenas de bits A y J de 32 y 48 bits respectivamente. La salida de $f(A, J)$ es una cadena de 32 bits, y la función es como sigue:
  1. Se expande el argumento A a una cadena de 48 bits utilizando una función de expansión E. La función $E(A)$ consiste en una permutación de los 32 bits de A junto con 16 repetidos (una vez).
  2. Se calcula $E(A) \oplus J$, y se escribe el resultado como una concatenación de ocho cadenas de 6 bits: $B = B_1 B_2 \ldots B_8$.

Descripción DES

3. El siguiente paso utiliza ocho S-boxes $S_1, S_2, \ldots, S_8$. Cada S_i es un areglo de 4×16, con enteros del $0 - 15$. Dada una cadena de bits $B = b_1 b_2 b_3 b_4 b_5 b_6$, se calcula $S_j(B_j)$ como sigue: sea $r = b_1 b_6$ $(0 \leq r \leq 3)$, y $c = b_2 b_3 b_4 b_5$ $(0 \leq c \leq 15)$. Entonces $C_j = S_j(B_j)$ para $j = 1, 2, \ldots, 8$, es definida como la entrada $S_j(r, c)$ como una cadena de 4 bits.
4. Finalmente, se permuta la cadena de bits $C = C_1 C_2 \ldots C_8$ de 32 bits de acuerdo a una permutación fija P, y este es el resultado de $f(A, J)$.

Descripción DES

Calendarización de la llave K

- La llave K es una cadena de 64 bits, de los cuales 56 son la llave y 8 son de paridad (para detección de errores). Los bits en las posiciones $8, 16, \ldots, 64$ están definidos para que los grupos de 8 bits (de los cuales son el último) contengan un número impar de 1's.
  Así que un error se puede detectar para cada grupo de 8 bits.

Descripción DES

Calendarización de la llave K

- Dada la llave K, e ignorando los bits de paridad, se permitan los bits de K de acuerdo a una permutación fija P_1. La función $P_1(K) = C_0 D_0$ da una salida de 56 bits, siendo C_0 la parte baja de 28 bits, y D_0 la parte alta.
- Para $i = 1, 2, \ldots 16$ se calcula

$$C_i = \mathsf{LS}_i(C_{i-1})$$
$$D_i = \mathsf{LS}_i(D_{i-1})$$
$$K_i = \mathsf{P}_2(C_i D_i)$$

en donde LS_i representa un shift circular hacia la izquierda de una o dos posiciones, dependiendo del valor de i: una posición si $i \in [1, 2, 9, 16]$, y dos posiciones de otra manera. La función de permutación P_2 es otra permutación fija.

Utilización con PyCrypto

Hash

```
1 from Crypto.Hash import SHA256
2 h = SHA256.new()
3 h.update(b'Hello')
4 print h.hexdigest()
```

Code taken from PyCrypto Documentation.

Utilización con PyCrypto

DES

```
1 from Crypto.Cipher import DES3
2 from Crypto import Random
3 key = b'Sixteen byte key'
4 iv = Random.new().read(DES3.block_size)
5 cipher = DES3.new(key, DES3.MODE_OFB, iv)
6 plaintext = b'sona si latine loqueris '
7 msg = iv + cipher.encrypt(plaintext)
```

Code taken from PyCrypto Documentation.

Modos de operación

Hemos dicho que DES trabaja con bloques de 64-bits, ¿qué pasa si requerimos cifrar más de 64 bits? Agarramos de 64 bits en 64 bits?

Modos de operación

Hemos dicho que DES trabaja con bloques de 64-bits, ¿qué pasa si requerimos cifrar más de 64 bits? Agarramos de 64 bits en 64 bits? (no)

Modos de bloques

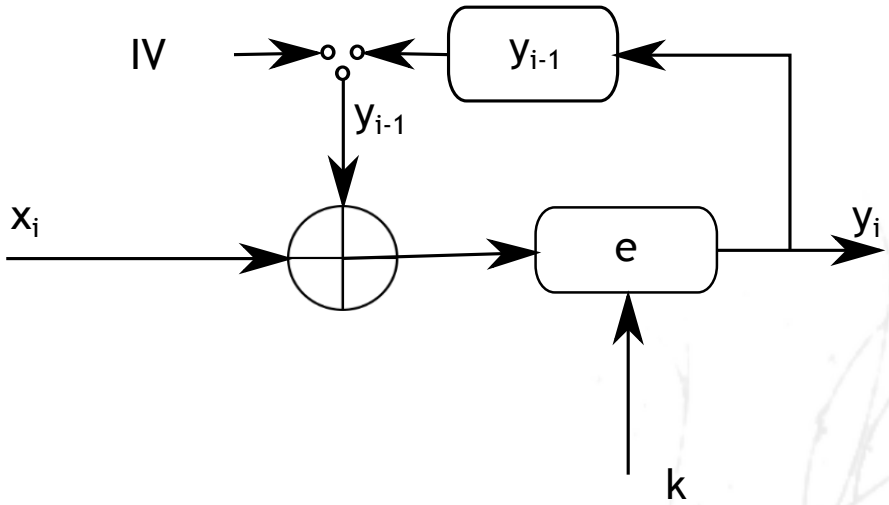
- ECB - Electronic Codebook Block
- CBC - Cipher Block Chaining

Modos de flujo

- CFB - Cipher Feedback
- OFB - Output Feedback

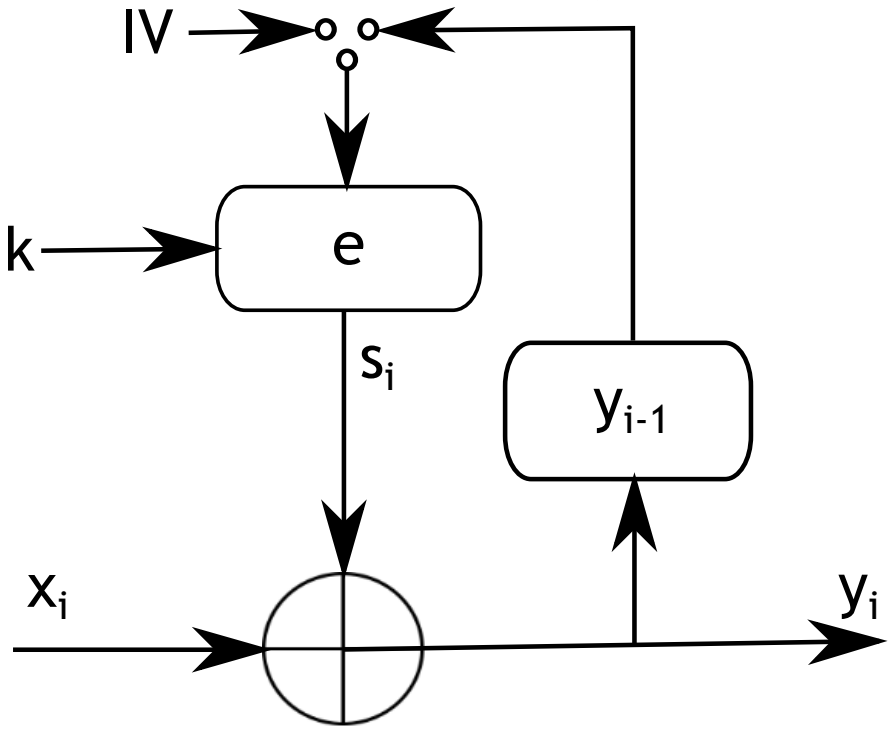
Por bloques

- ECB - El mensaje se rompe en bloques de 64-bits (se rellena con ceros).
- CBC - Hace un xor de la salida anterior con el bloque a cifrar (requiere vector de inicialización).

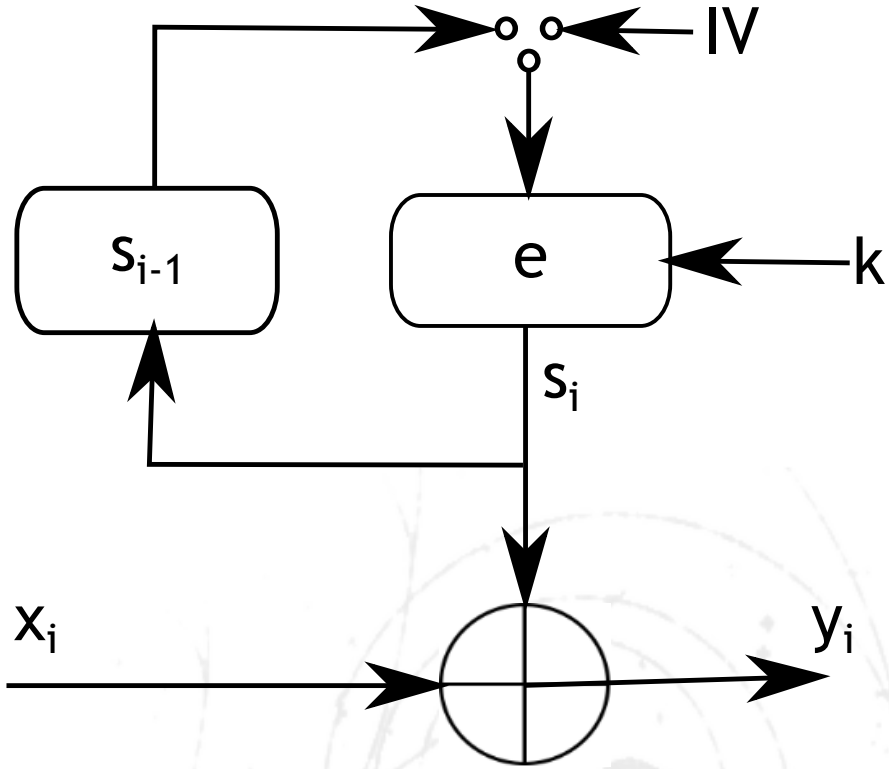


Por flujo

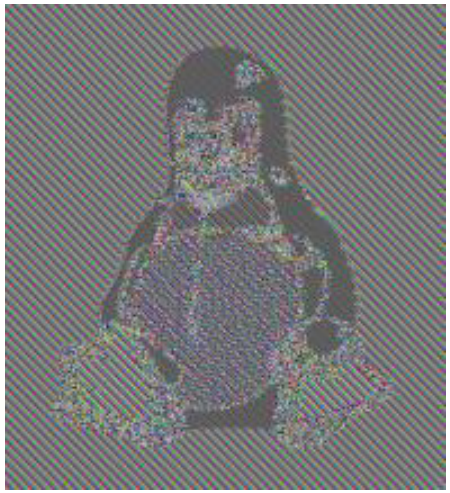
CFB - Se hace un xor de la salida anterior con el mensaje



OFB - El feedback es independiente del mensaje actual



Comparativa



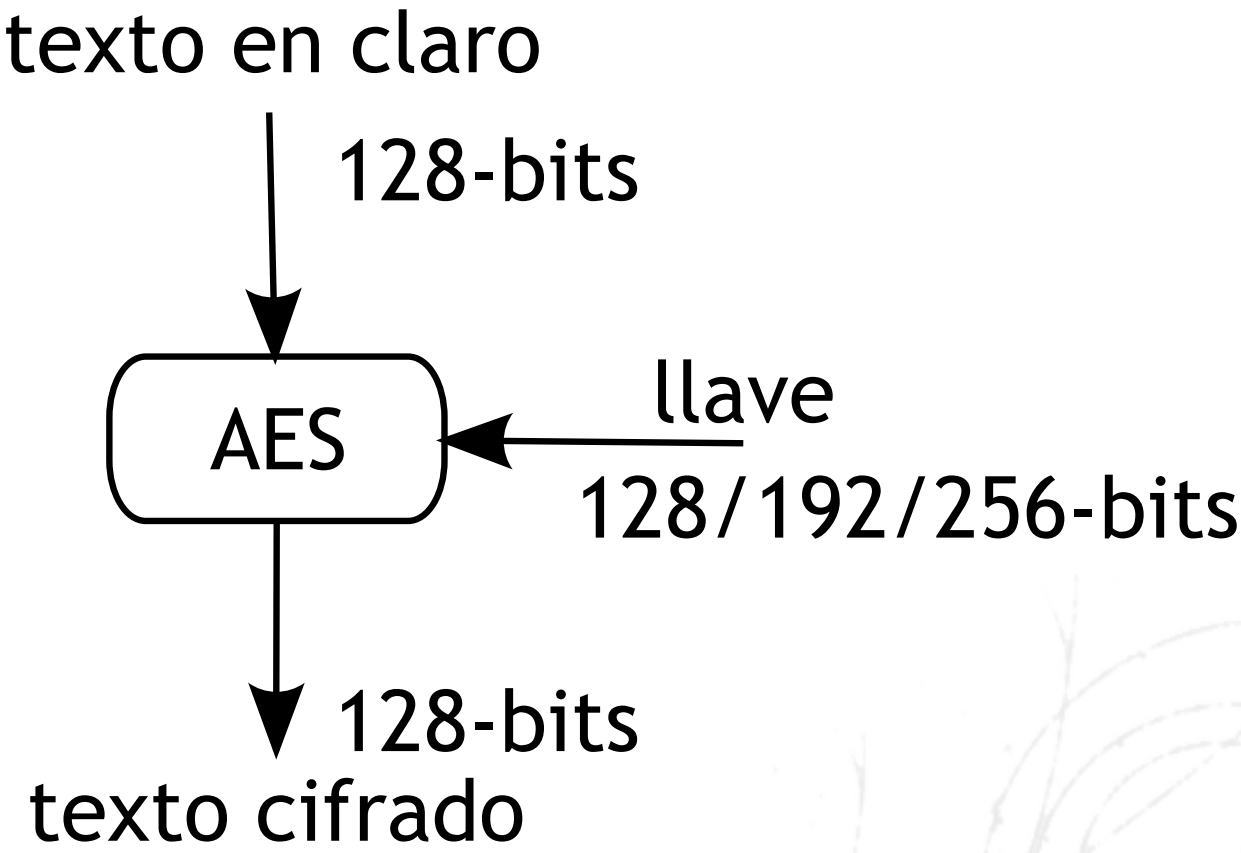
https://doegox.github.io/ElectronicColoringBook/

AES - Advanced Encryption Standard

¿Porqué un estándar nuevo?

- Ataques de fuerza bruta
- La solución (Triple DES) lo hace el triple de lento
- DES es eficiente solo para hardware
- Nuevos tipos de ataques
- Utilizar bloques de 64-bits no es útil para todos los escenarios

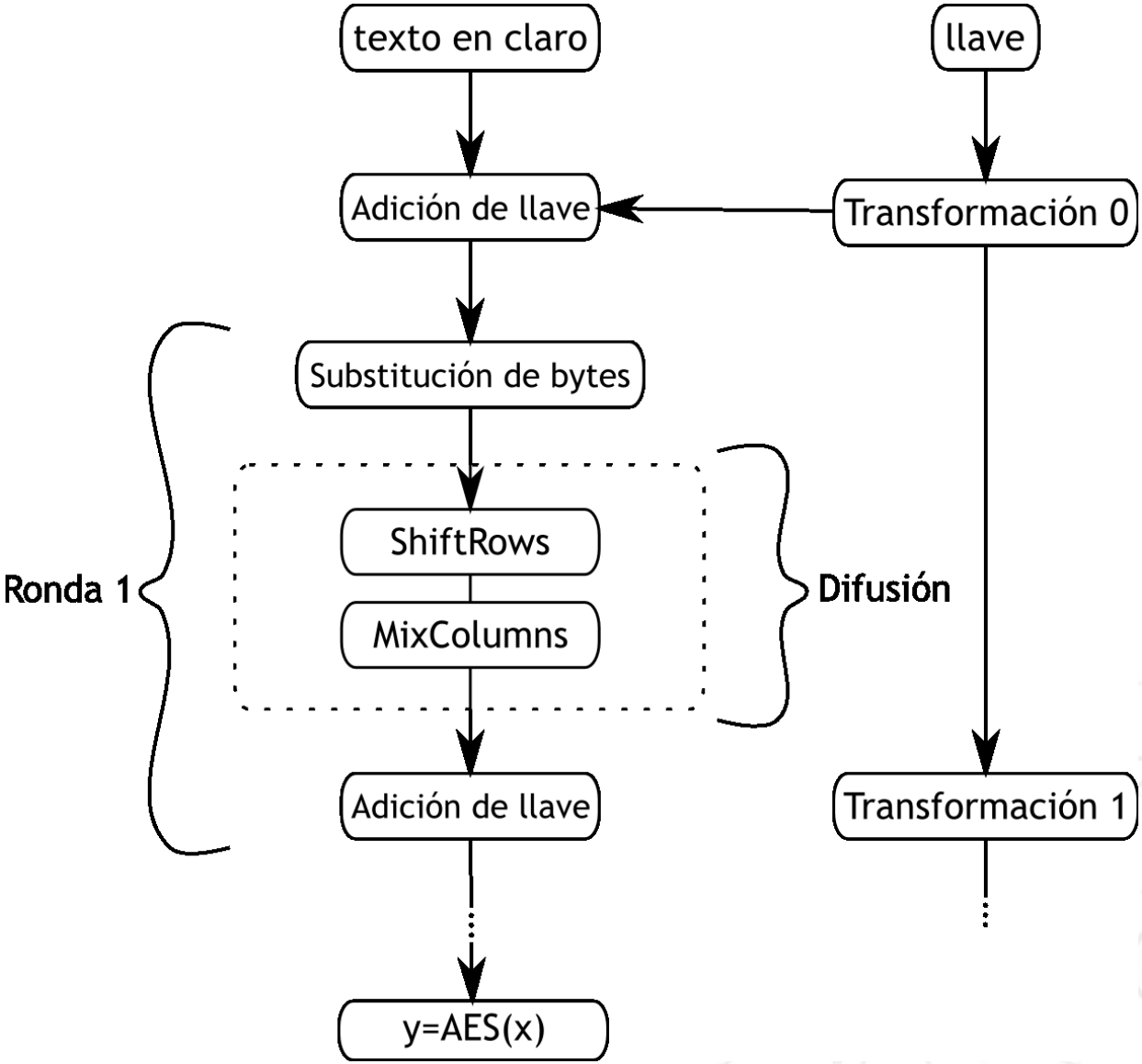
AES - Caja negra



AES - descripción

- AES no utiliza una función Feistel, se desea cifrar todo un bloque por ronda
- Se necesitan 10, 12 o 14 rondas para cifrar con claves de 128, 192 o 256 bits
- En cada ronda hay 3 capas: Adición de llave, de Sustitución de bytes, y de difusión
- la capa de difusión se subdivide en: ShiftRow, que permuta datos a nivel de byte; y MixColumn, que mezcla bloques de 4-bytes dentro de una matriz

AES - rondas


texto en claro
llave
Adición de llave
Transformación 0
Substitución de bytes
ShiftRows
MixColumns
Ronda 1
Difusión
Adición de llave
Transformación 1
y=AES(x)

AES - Detalle

- El algoritmo utiliza llaves de $128, 192,$ o 256 bits.
- AES-128 toma una entrada de 128 bits, hace un XOR con la llave original, y luego realiza 10 rondas con derivaciones de esta nueva llave.
- Cada ronda consiste de cuatro *capas*: *ByteSub* (BS), *ShiftRow* (SR), *MixColumn* (MC), y *AddRoundKey* (ARK).
- En la décima ronda se omite el paso de MC. Así que el cifrado Rijndael queda así:

Ronda0	ARK con la llave original
Ronda1	BS,SR,MC,ARK utilizando la llave de ronda1
$\ldots$	
Ronda9	BS,SR,ARK utilizando la llave de ronda9
Ronda10	BS,SR,ARK utilizando la llave de ronda10

La salida del algoritmo es de 128 bits.

AES - Detalle

- La entrada de 128 bits se parte en 16 grupos de 1 byte
- Los 16 bytes se arreglan en una matriz de 4×4
  - Los pasos de ShiftRow, y MixColumn operan en la matriz
  - Los pasos de ByteSub, y AddRoundKey operan sobre los bytes

Algunas de las operaciones utilizan el campo finito $\mathbb{F}_{2^8} = \mathbb{F}_{256}$. Las propiedades de este campo son que sus elementos se representan por un solo byte, y que podemos sumar y multiplicar los bytes, y obtener exactamente 1 byte.

AES - Detalle

- Cada byte, exceptuando el 0 tiene su inverso multiplicativo, esto es, dado un byte b, existe un b^{-1} tal que $bb^{-1} = 00000001$
- La construcción de este campo (y más imporantemente, la regla de multiplicación) dependen de un polinomio irreducible de grado 8. El polinomio utilizado por Rijndael es $x^8 + x^4 + x^3 + x + 1$
- Para utilizar el algoritmo, se necesita poder realizar multiplicación de elementos en $\mathbb{F}_{256}$. En la práctica, una tabla *lookup table* se puede utilizar.

AES - Detalle

1. ByteSub

- En este paso cada byte de la matriz se reemplaza por otro byte utilizando la S-box - una matriz de 16×16 cuyas líneas y columnas están enumeradas del 0 al 15, y cuyas entradas consisten de los $2^8 = 16^2$ diferentes posibles bytes.

AES - Detalle

- Considere cada byte con bits $abcdefgh$. Encuentre la entrada en la S-bis en la columna número $abcd$ y la columna $efgh$. Convierta la entrada a un byte en binario, y este byte reemplaza al original.
- (A diferencia de DES, la construcción de esta S-box es pública. para calcular el byte de reemplazo sin utilizar la S-bix, hay que ver el byte original como un elemento en $\mathbb{F}_{256} = \mathbb{F}_{2^8}$ buscando su inverso multiplicativo (en el caso del 0 es el 0). Considere este inverso como un vector de 8 bits y multiplíquelo por una matriz circular a la derecha de 8×8 ($C(10001111)$), y agregue (XOR) el vector fijo

AES - Detalle

2. ShiftRow

- Las líneas de la matriz de 4×4 bytes son rotadas cíclicamente a la izquierda como sigue: la primer línea no se rota, la segunda se rota 1 lugar, la tercera se rota 2 lugares, y la cuarta se rota 3.

AES - Detalle

3. Mixcolumn

- En este paso considere los valores de la matriz de 4×4 como elementos de:

$$\begin{matrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{matrix}$$

AES - Detalle

4. AddRoundKey

- In each round a different round key of 128 bits is generated from the original key. This round key is divided into 16 bytes, and each byte is XOR'ed with an entry of the 4×4 matrix we are working with.

AES - Detalle

Calendarización de la llave

- The original key is broken up into bytes and these are arranged in a 4×4 array.
- This matrix is expanded by adding 40 new columns so that the total number of columns is 44: additional columns are generated recursively. Let $W(j)$ denote column j (starting at 0), we have:
  - If $i \neq 0 \bmod 4 : W(i-4) \oplus W(i-1)$
  - If $i \equiv 0 \bmod 4 : W(i-4) \oplus T(W(i-1))$
- $W(i-1)$ is modified before XORing. To find $T(W(i1))$ we:
  1. Cyclically shift the elements of the column up by one.
  2. Using the S-box of the ByteSub step, replace the bytes in the column
  3. Compute the round constant, $r(i) = 00000010(i4)/4$ in GF(256) and add the result to the first byte of the column.

The the i-th round key for the i-th round is the key which consists of the columns $W(4i)$, $W(4i+1)$, $W(4i+2)$ and

AES - Detalle

http://www.swfcabin.com/open/1299977086

Codigo

```
1 from Crypto.Cipher import AES
2 from Crypto import Random
3 key = b'Sixteen byte key'
4 iv = Random.new().read(AES.block_size)
5 cipher = AES.new(key, AES.MODE_CFB, iv)
6 msg = iv + cipher.encrypt(b'Attack at dawn')
```

Code taken from PyCrypto Documentation.

Código para MIRACL

```
gcc -O2 -o testaes testaes.c mraes.c miracl.a
```

```
1  #include "stdlib.h"
2  #include "miracl.h"
3  #define NB 4
4
5  int main() {
6      int i,j,nk;
7      aes a;
8      MR_BYTE y,x,m;
9      char key[32];
10     char block[16];
11     char iv[16];
12     for (i=0;i<32;i++) key[i
           ]=0;
13     key[0]=1;
14     for (i=0;i<16;i++) iv[i]=i;
15     for (i=0;i<16;i++) block[i
           ]=i;
16     for (nk=16;nk<=32;nk+=8) {
17         printf("\nKey Size= %d bits\n",nk*8);
18         if (!aes_init(&a,MR_CBC,nk,key,iv)) {
19             printf("Failed to Initialize\n");
20             return 0;
21         }
22         printf("Plain=   ");
23         for (i=0;i<4*NB;i++) printf("%02x",
               block[i]);
24         printf("\n");
25         aes_encrypt(&a,block);
26         printf("Encrypt= ");
27         for (i=0;i<4*NB;i++) printf("%02x",(
               unsigned char)block[i]);
28         printf("\n");
29         aes_reset(&a,MR_CBC,iv);
30         aes_decrypt(&a,block);
31         printf("Decrypt= ");
32         for (i=0;i<4*NB;i++) printf("%02x",
               block[i]);
33         printf("\n");
34
35         aes_end(&a);
36     }
37     return 0;
38 }
```

Code taken from Certivox MIRACL.