# Criptografía binaria
## block ciphers y funciones de hash

Carlos Sarraute

*Core Security Technologies*

Jornadas de Criptografía y Códigos Autocorrectores

20 al 24 de noviembre 2006 – Mar del Plata

# Agenda

- Introducción
  - como definir la seguridad de un sistema de encripción?
- Block ciphers
  - tipos de ataques
  - modos de operación
  - redes de substituciones y componentes lineales
- Funciones de hash
  - meta construcción de Merkle-Damgård
  - MD5

# Introducción : el secreto perfecto

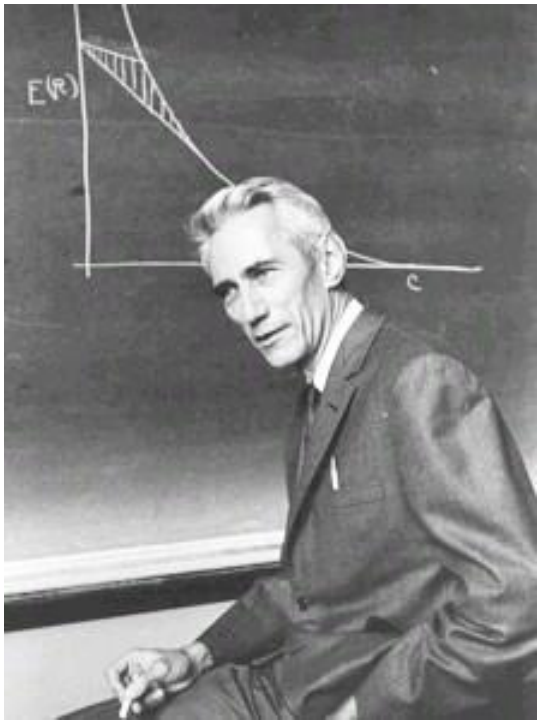
# Que es la seguridad?

- Acotemos la pregunta a la seguridad de un sistema de encripción
  - Que quiere decir un sistema de encripción sea perfectamente seguro?

- Respuesta de Claude Shannon en 1949

  "Communication Theory of Secrecy Systems"

  define lo que llama "perfect secrecy"



# Sistema de encripción general

- P = espacio de textos planos

  C = espacio de textos cifrados

  K = espacio de claves

- $\{ E_k : k \in K \}$ = funciones de encripción

  $\{ D_k : k \in K \}$ = funciones de desencripción

# Sistema de encripción simétrico

- Alice y Bob intercambian una clave k usando un canal seguro

- Alice y Bob se comunican por un canal inseguro
  - Alice calcula $c = E_k(p)$
  - Se lo manda a Bob
  - Bob calcula $p = D_k(c)$
  - Se cumple $D_k( E_k (p) ) = p$

- Distribución de probabilidades en los espacios P y K
  - Pr(p), Pr(k)
  - suponemos $Pr(p) > 0$ para todo $p \in P$

# Secreto perfecto o secreto de Shannon

- Para todo $p \in P$ y para todo $c \in C$

  $Pr( p \mid c ) = Pr( p )$

- Teorema: la cota de Shannon (1949)
  - Si un sistema mantiene el secreto perfecto entonces

    $| K | \geq | P |$

- Demostración por el absurdo. Fijamos un $c \in C$.
  - Sup que existe $p_0 \in P$ tal que para toda $k \in K$, $p_0 \neq D_k(c)$
  - Entonces $Pr( p_0 \mid c ) = 0 < Pr( p_0 )$
  - Luego para toda $p \in P$, existe $k \in K$ tal que $p = D_k(c)$

# El One-time Pad

- Conclusión: la clave tiene que ser por lo menos tan larga como el mensaje

- Existe un sistema de encripcion asi:

  $c = p \oplus k$

  $p = c \oplus k$

- No sirve en la práctica
  - aunque se usó durante la guerra fría
  - que ataques se les ocurre?

# Block ciphers

# Operaciones booleanas

- Vector booleano = vector cuyas coordenadas son bits
- XOR = suma modulo 2
  - notado a ^ b  a ⊕ b
- AND = producto modulo 2
  - notado a & b  a . b
- OR
  - notado a | b
- instrucciones muy rápidas

# Funciones booleanas

- función booleana
  - $f : \{0,1\}^m \rightarrow \{0,1\}^n$
- permutación booleana
  - $f : \{0,1\}^n \rightarrow \{0,1\}^n$ inversible
- un block cipher de n bits, con claves de k bits
  - $E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$

    tal que para cada $K \in \{0,1\}^k$

    $P \rightarrow E(P,K)$ es una permutación booleana
- n es el tamaño del bloque
- k es el tamaño de las claves

# Ejemplos de tamaños

- DES
  - bloques de n = 64 bits
  - claves de k = 56 bits
    - » especificada como 64 bits, pero con 8 bits de redundancia

- AES = Rijndael
  - bloques de n = 128 bits
  - claves de k = 128, 164 o 192 bits

# Block ciphers

- cada block cipher es una familia de $2^k$ permutaciones

- para cada $K \in \{0,1\}^k$
  - $E(P,K) = E_K(P)$ = función de encripción
  - la inversa $D_K = E_K^{-1}$ = función de desencripción

- cual es el mejor block cipher de n bits que se puede construir?

# Block ciphers

- cada block cipher es una familia de $2^k$ permutaciones

- para cada $K \in \{0,1\}^k$
  - $E(P,K) = E_K(P)$ = función de encripción
  - la inversa $D_K = E_K^{-1}$ = función de desencripción

- cual es el mejor block cipher de n bits que se puede construir?
  - true random cipher
  - implementa las $(2^n)!$ permutaciones de $\{0,1\}^n$ en $\{0,1\}^n$
  - estimación usando la fórmula de Stirling :

$$(2^n)! \approx \sqrt{2\pi 2^n}\left(\frac{2^n}{e}\right)^{2^n} \geq (2^{n-2})^{2^n}$$

# Cantidad de block ciphers

- Dado un tamaño de bloque n y un tamaño de clave k
  cual es la cantidad de block ciphers de esas dimensiones?

$$\left(2^n!\right)^{2^k}$$

- Para valores prácticos (n y k mayores que 60), el subconjunto de block ciphers con vulnerabilidades explotables forma una minoridad neglible

# Seguridad de un block cipher

- Las suposiciones básicas son:

- el principio de Kerckhoff
  - el atacante conoce todos los detalles de la función E
  - (salvo la clave)

- el atacante tiene acceso a toda la información que se transmite por el canal de comunicación.

- “security through obscurity” es tratar de esconder los detalles de las funciones usadas
  - se sigue viendo en la industria

# Ataques estandar

- ciphertext-only
  - el atacante no dispone de información adicional

- known-plaintext
  - el atacante dispone de pares $(m_i,c_i)$ de plaintext y ciphertext
  - los reportes metereológicos que los alemanes mandaban a sus submarinos sirvieron como known-plaintext para el equipo de Bletchley Park que analizaba la Enigma.
  - si encripto archivos que comienzan por un header fijo y no uso ningún tipo de random padding, un atacante dispone automáticamente de pares $(m_i,c_i)$.
  - el criptoanálisis lineal supone que disponemos de una gran cantidad de pares $(m_i,c_i)$.

# Ataques estandar

- chosen-plaintext
  - el atacante dispone de los ciphertexts correspondientes a un conjunto de plaintexts que el mismo eligió.
  - el criptoanálisis diferencial supone que disponemos de una gran cantidad de pares $(m_i,c_i)$ debidamente elegidos
  - normalmente se requiere de un block cipher que resista por lo menos a un chosen-plaintext attack.

- adaptive chosen-plaintext
  - donde la elección de los plaintexts puede depender de los pares $(m_i,c_i)$ anteriores.

- chosen-ciphertext
- adaptive chosen-ciphertext

# Complejidad de un ataque

- complejidad de datos
  - cantidad esperada de inputs que va a requerir el ataque
  - por ejemplo, cantidad de ciphertexts, o cantidad de pares plaintext-ciphertext).
  - medidad de complejidad llamada pasiva (no depende del atacante).
- complejidad de almacenamiento
  - la cantidad esperada de unidades de almacenamiento requeridas.
  - rainbow tables
- complejidad de procesamiento
  - la cantidad esperada de operaciones requeridas.
- medidas activas, dependen de los recursos del atacante

# Crackeando DES por fuerza bruta

- Para crackear DES por fuerza bruta
  - examinando las $2^{56}$ posibles claves
- la Electronic Frontier Foundation construyó en 1998 una máquina de $ 250,000
- le permitió desencriptar un ciphertext en menos de tres dias.
- En 1999, ya lograban hacerlo en menos de 22 horas.

# Noción de K-seguridad

- Considerando el conjunto de block ciphers de dimensiones n y k

$$(2^n!)^{2^k}$$

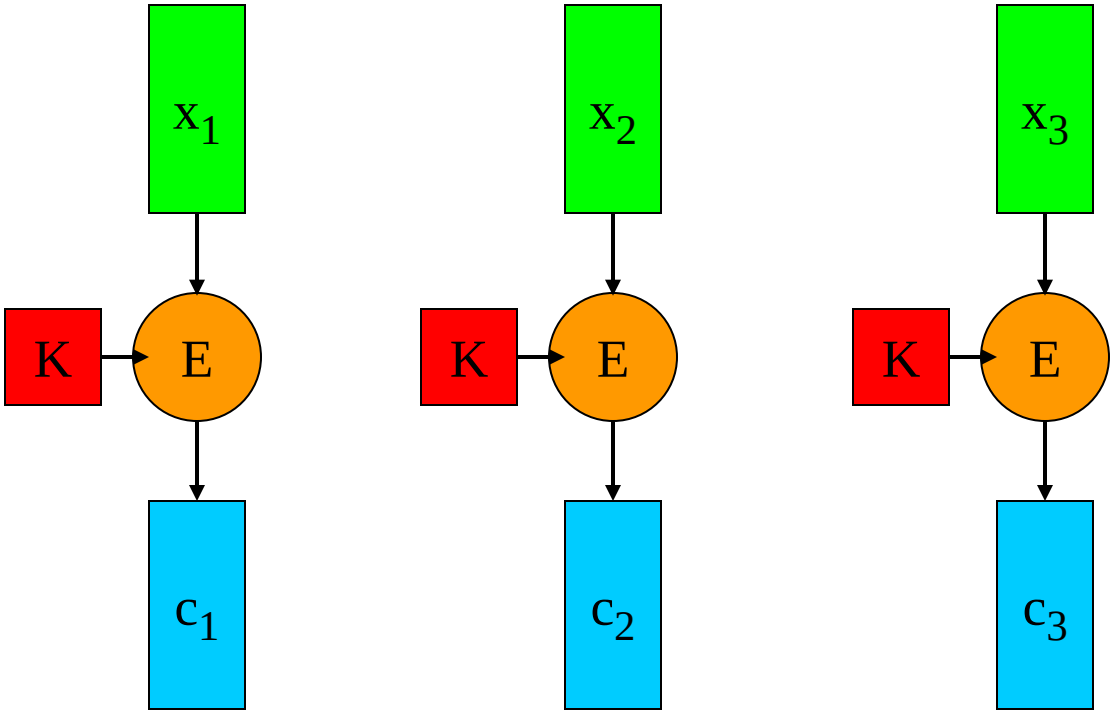
- Un block cipher se dice K-seguro
  - si todas las posibles estrategias de ataque tienen los mismos requerimientos de procesamiento y almacenamiento que para la mayoria de los block ciphers de mismas dimensiones.
  - esto debe ser el caso para todos los posibles modos de acceso del adversario:

    known / chosen / adaptively chosen plaintext / ciphertext, ...
  - y para cualquier distribución de las claves.

  - Noción relativa: se puede construir un block cipher K-seguro con bloques y claves de 5 bits, pero obviamente no es muy seguro.

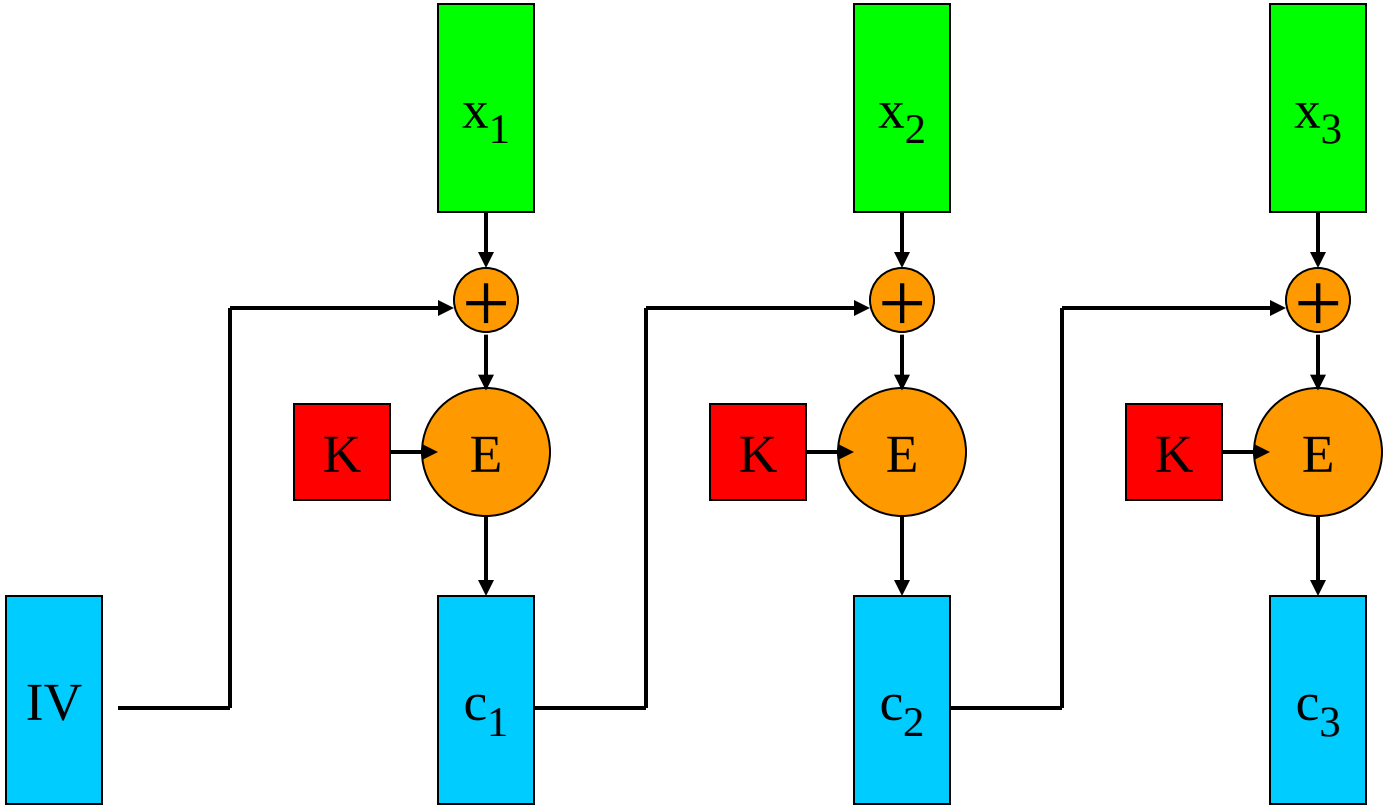
# Modos de operación

- Modo ECB = Electronic Code Book



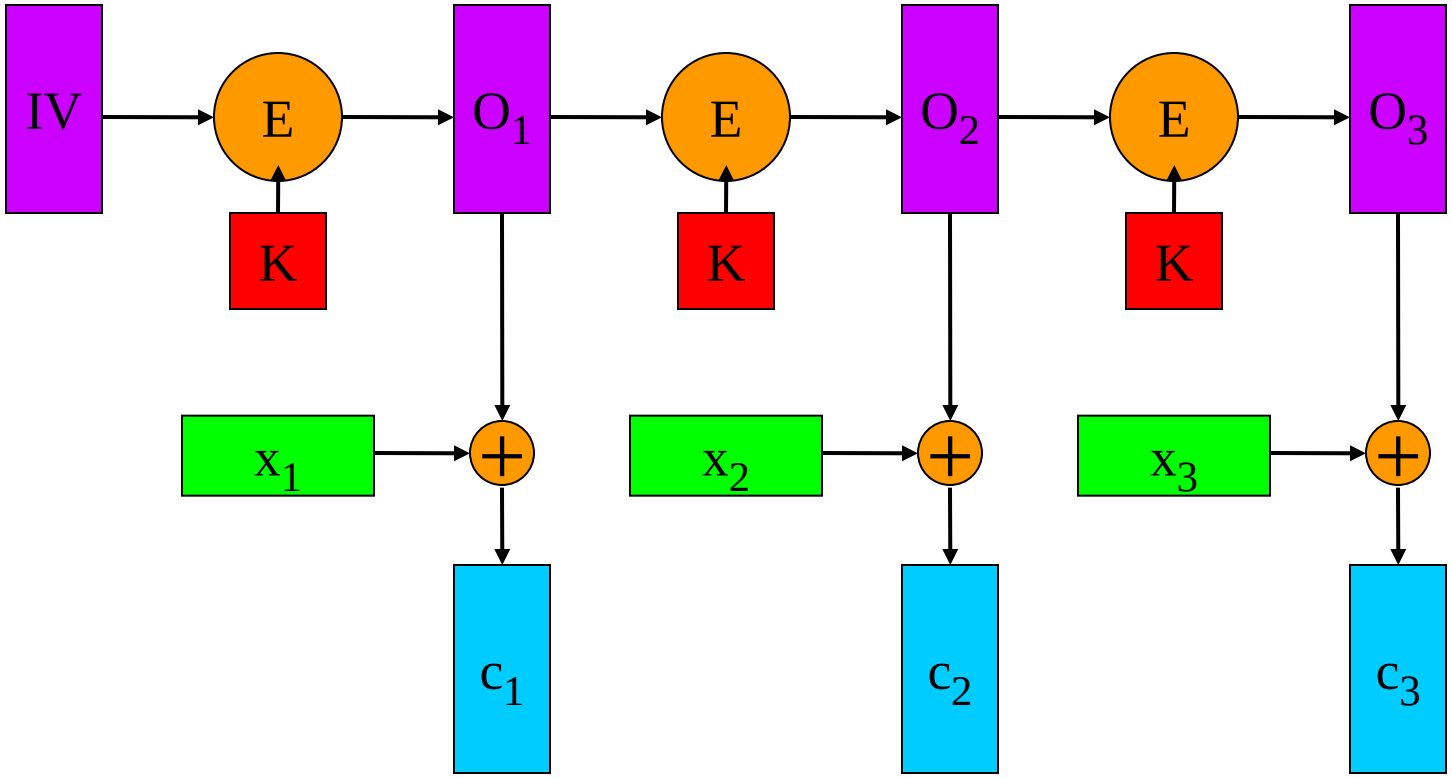
# Modos de operación

- Modo CBC = Cipher Block Chaining



# Modos de operación

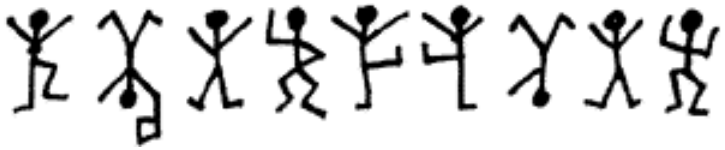
- Modo OFB = Output Feed Back
- resulta un stream cipher



# Camino hacia DES y Rijndael

- En 1973, Horst Feistel publicó en la Scientific American "Cryptography and Computer Privacy"
  - cuenta el camino desde los ciphers históricos (con lápiz y papel) hasta DES.

- Toma de los ciphers antiguos dos ideas básicas:
- el substitution cipher
  - cada letra se reemplaza por otra letra, o por cualquier otro simbolo raro (como los hombres que bailan que criptoanalizó Sherlock Holmes)



- los ciphers basados en permutaciones de las letras.

# S-box = substitution box

- Al substitution cipher corresponde el substitution box = S-box

- un aparato que transforma una entrada de m bits en una salida de m bits en forma arbitraria
  - estable una biyección entre las $2^m$ entradas y las $2^m$ salidas posibles.
  - componente es no lineal.

- El problema de las S-boxes es que su construcción es complicada
- esto limita el tamaño de m
  - una S-box con m=1024 resistiría cualquier análisis, pero es tecnológicamente imposible.
- por software, se representa con una tabla de $2^m$ entradas

# L-box = componente lineal

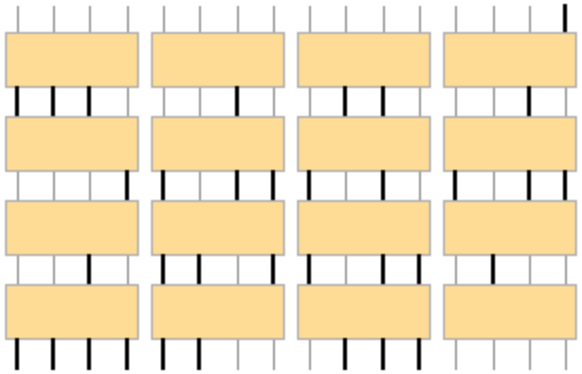
- Es muy fácil construir un cableado que conecta cada bit de entrada con un bit de la salida
  - al mejor estilo del “cerebro mágico”
- permuta las posiciones de los bits sin alterar su valor.

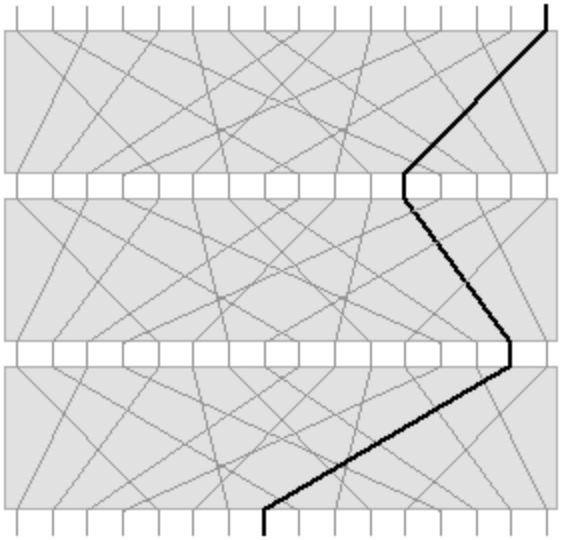
- En el artículo original de Feistel, los llama permutation box, o P-box.
  - lo llamamos L-box porque nos importa que sea lineal

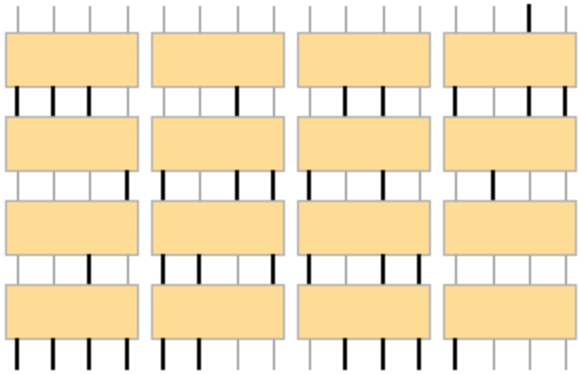
- Podemos ver al L-box como una transformación booleana lineal
  - $L : \{0,1\}^n \rightarrow \{0,1\}^n$
  - cuya matriz es una permutación de la identidad $I_n$.

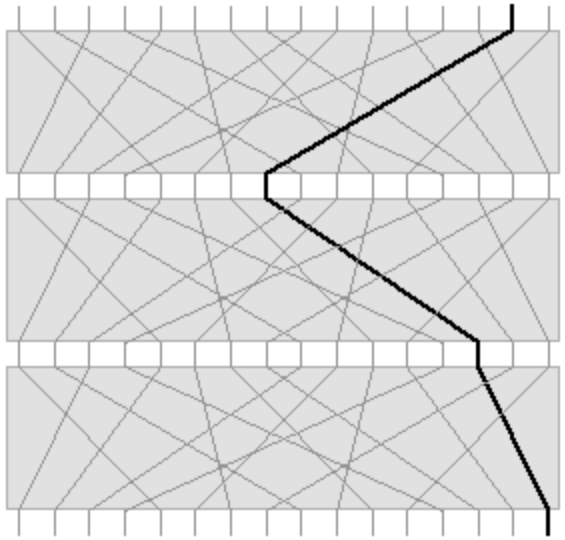
- Implementación en software : tabla con n entradas

# S-boxes y L-boxes solos no sirven



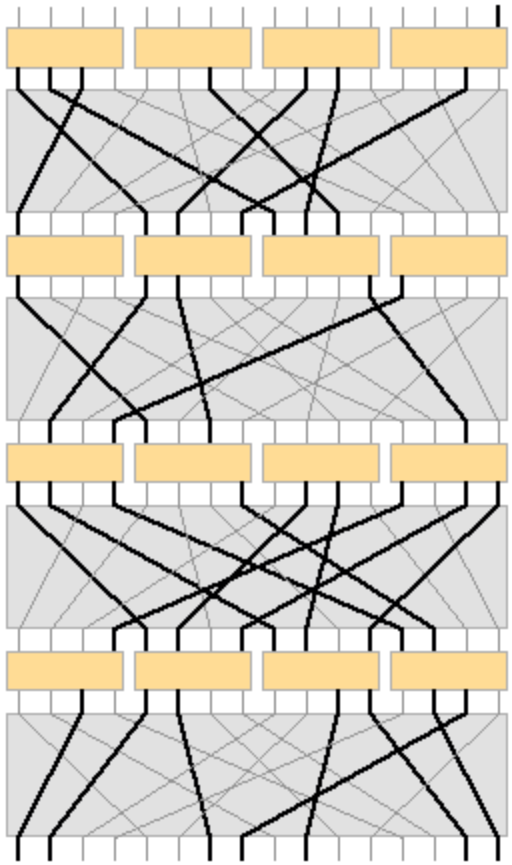


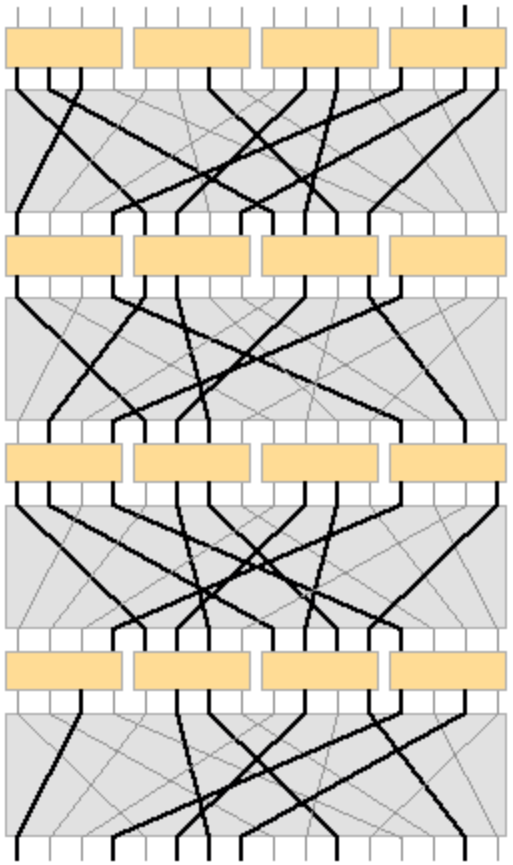




# Red de substituciones y componentes lineales

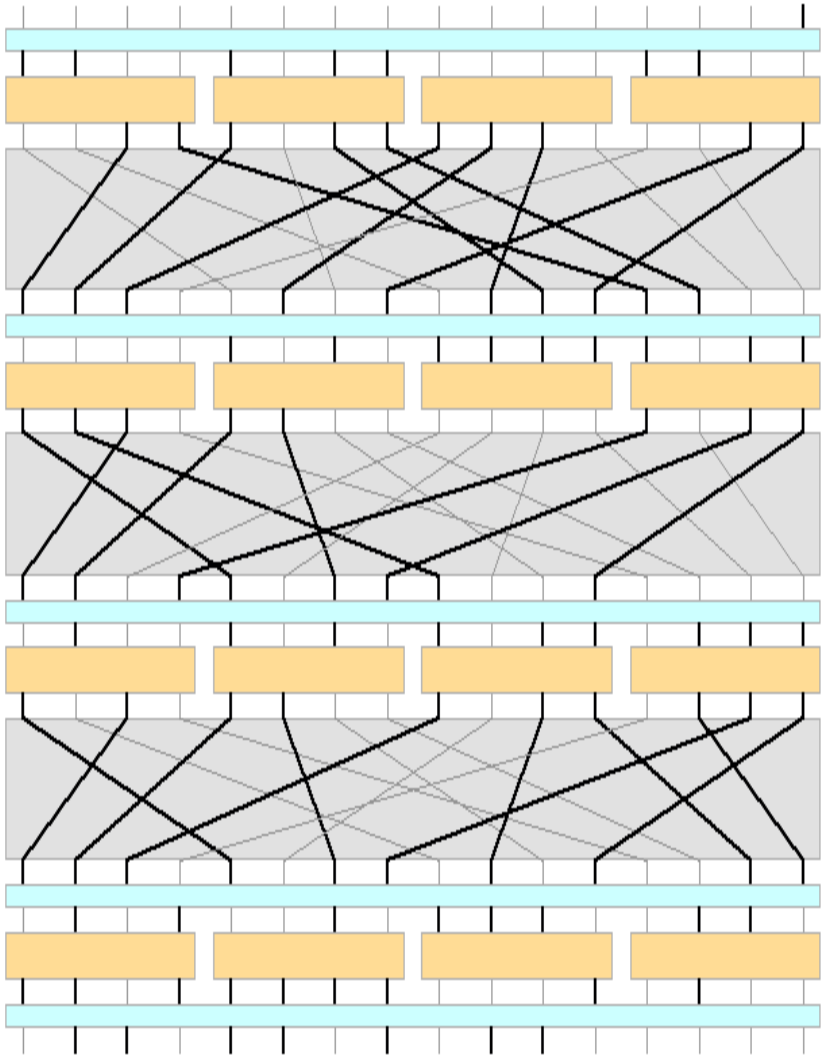
- Block cipher iterado de 4 rondas
- confusión y difusión





# Red de S-box y L-box con clave

- Agregamos un XOR con las claves en cada ronda
- Key schedule = como generar claves para cada ronda a partir de la clave K

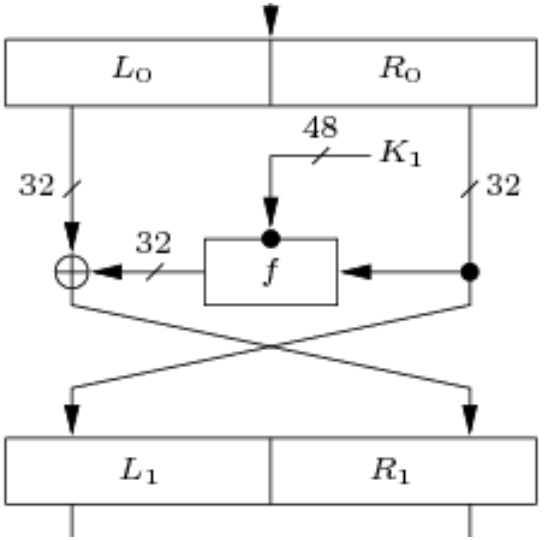


# DES y Rijndael

- Rijndael sigue este esquema con más S-boxes y más rondas
  - detalles en el curso sobre AES

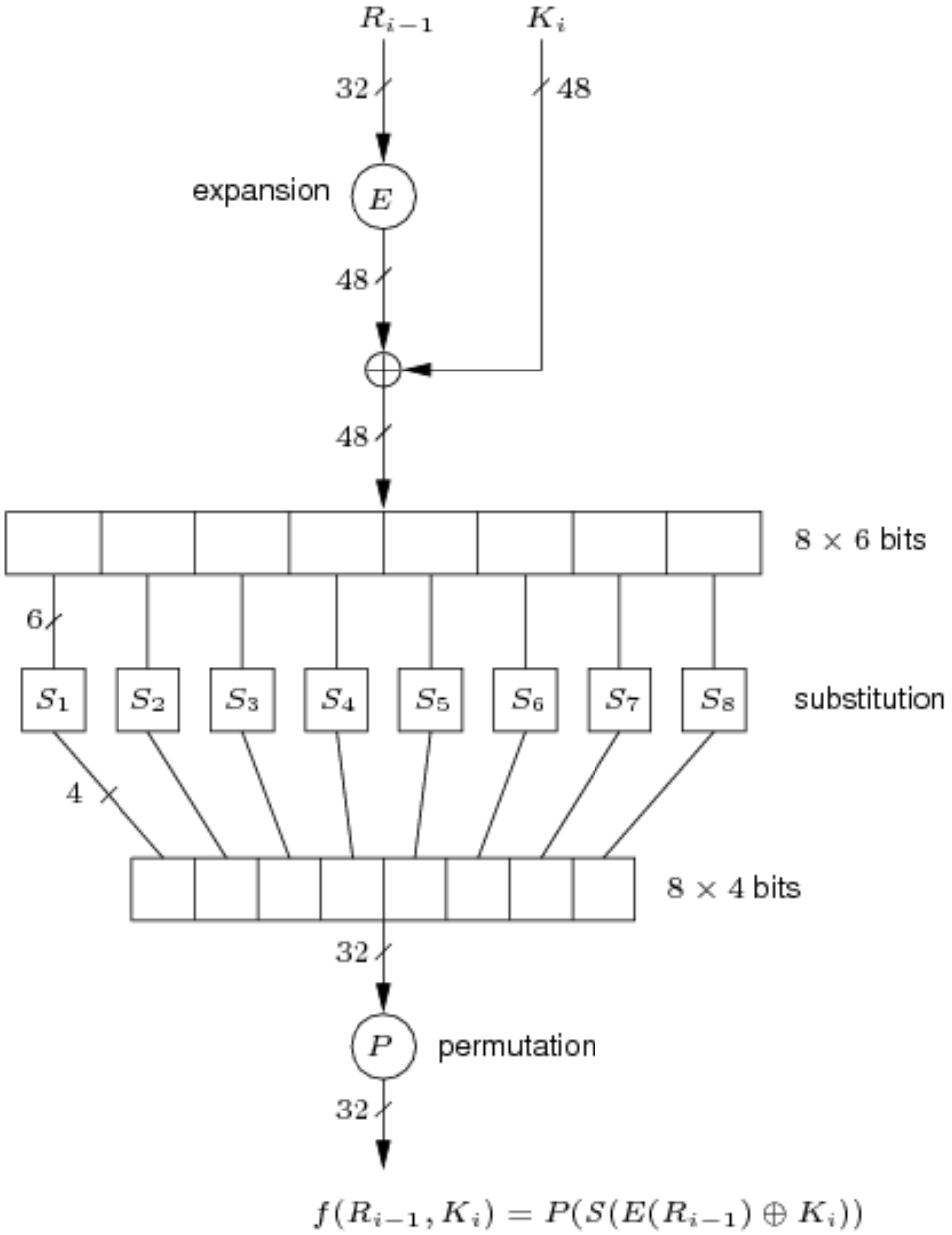
- DES sigue un esquema un poco más enredado

- Tiene estructura de Feistel network
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f( R_{i-1}, K_i )$



# La f de DES

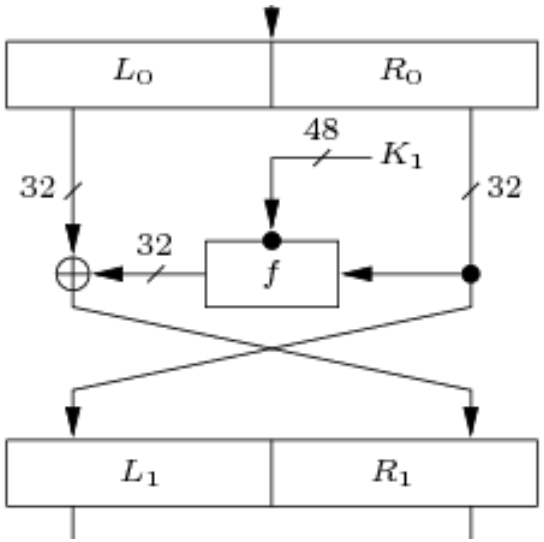
- f no es inversible

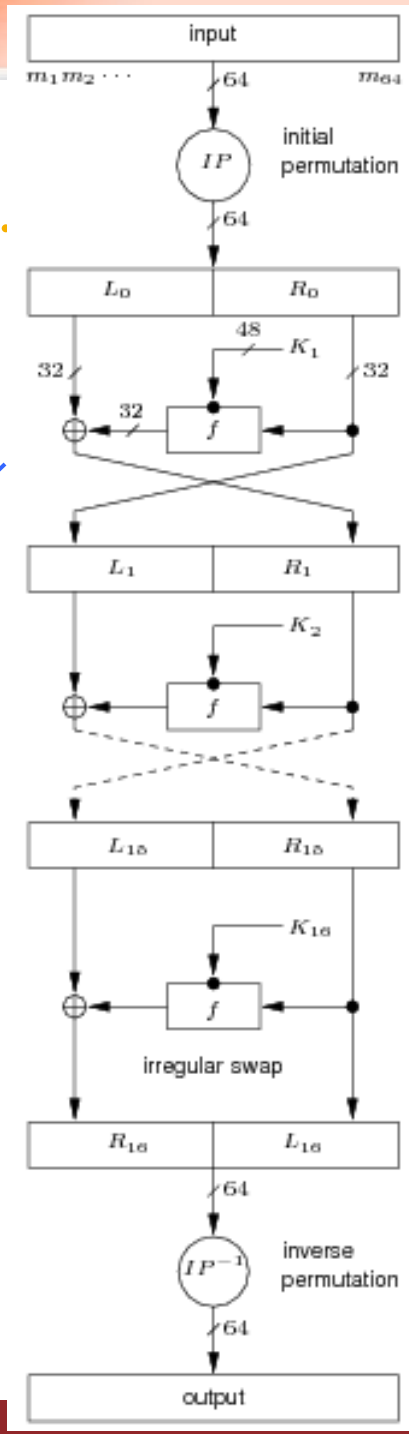


# Feistel network

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

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



# Funciones de hash

# Funciones de hash

- una función de hash h tiene las propiedades
- compresión:
  - toma una entrada x de tamaño arbitrario y produce una salida h(x) de tamaño fijo
- fácil de computar:
  - dados h y x, es fácil calcular h(x)
- sirve como una representación compacta del input
- las colisiones son inevitables

# Funciones de hash criptográficas

- resistente a calculo de preimagen = one way
  - dado y = h(x), no se puede (computacionalmente) calcular un x'
    tal que h(x´) = y  sin conocer x

- resistente a calculo de segunda preimagen
  - dado x, no se puede encontrar una segunda preimagen x'
    tal que h(x) = h(x´)
  - funcion de hash one way débil

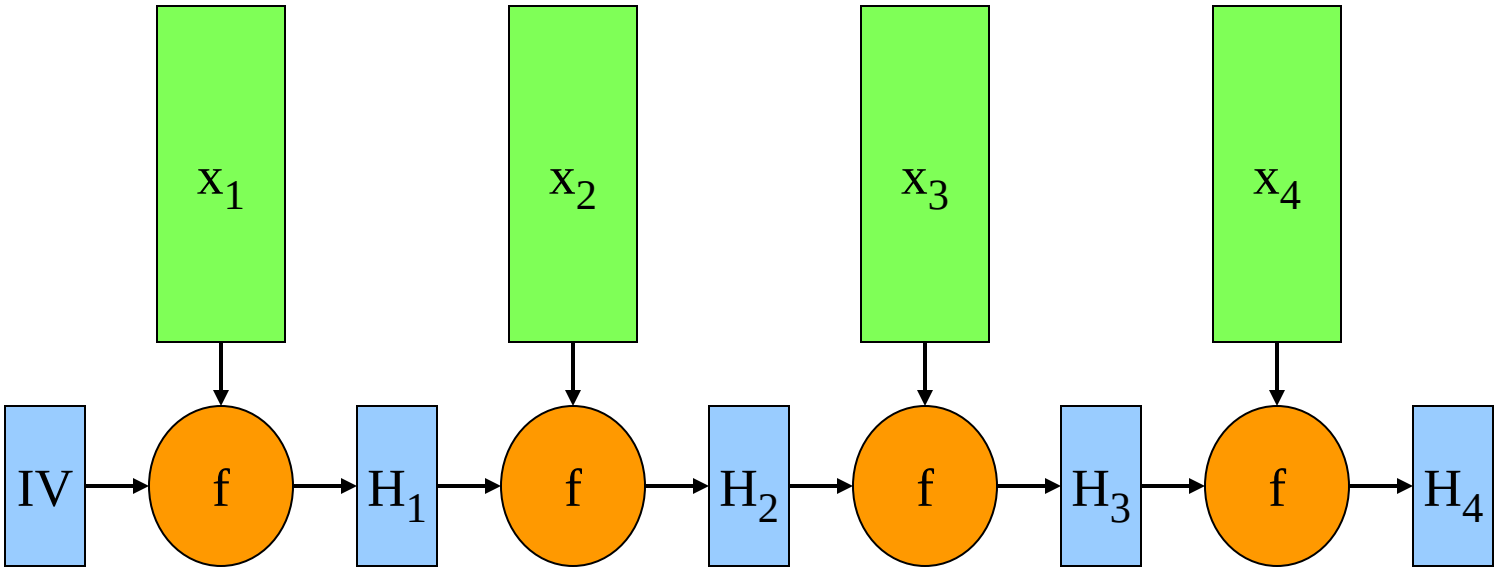
- resistente a colisiones
  - no se puede encontrar x ≠ x' que tengan el mismo hash
    o sea tales que h(x) = h(x´)
  - funcion de hash one way fuerte

# Usos de las funciones de hash

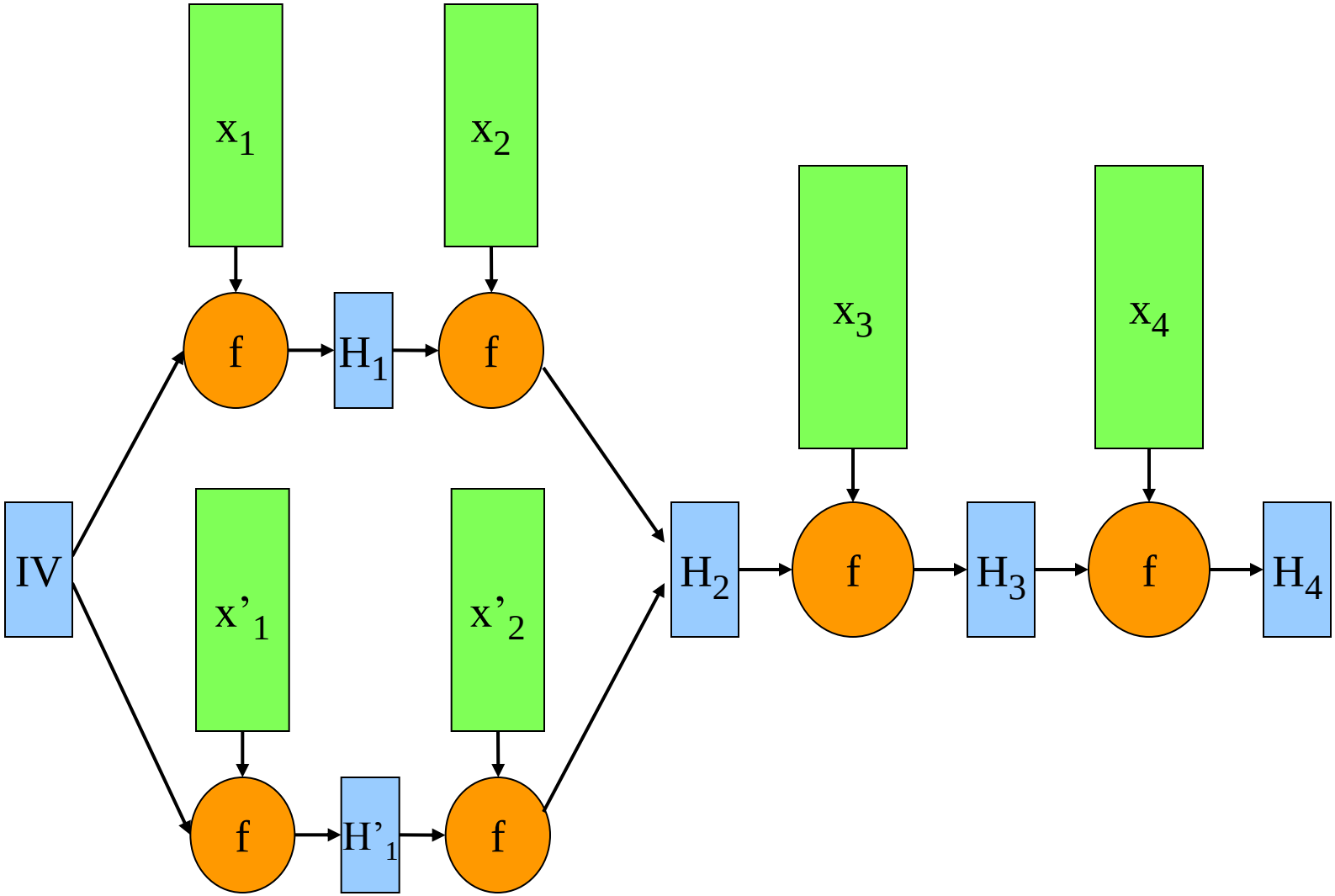
- integridad de datos
  - calcular el hash de los archivos a proteger
  - compara con el hash guardado para verificar que el archivo no fue modificado
  - para esto se usa MD5
  - firmas digitales con RSA (SSL)
- MAC = message authentication code
  - junto con una clave
  - verificar la fuente del mensaje y su integridad
- pseudo random number generators
  - para mezclar fuentes de entropía en un random pool

# Meta construcción de Merkle Damgård

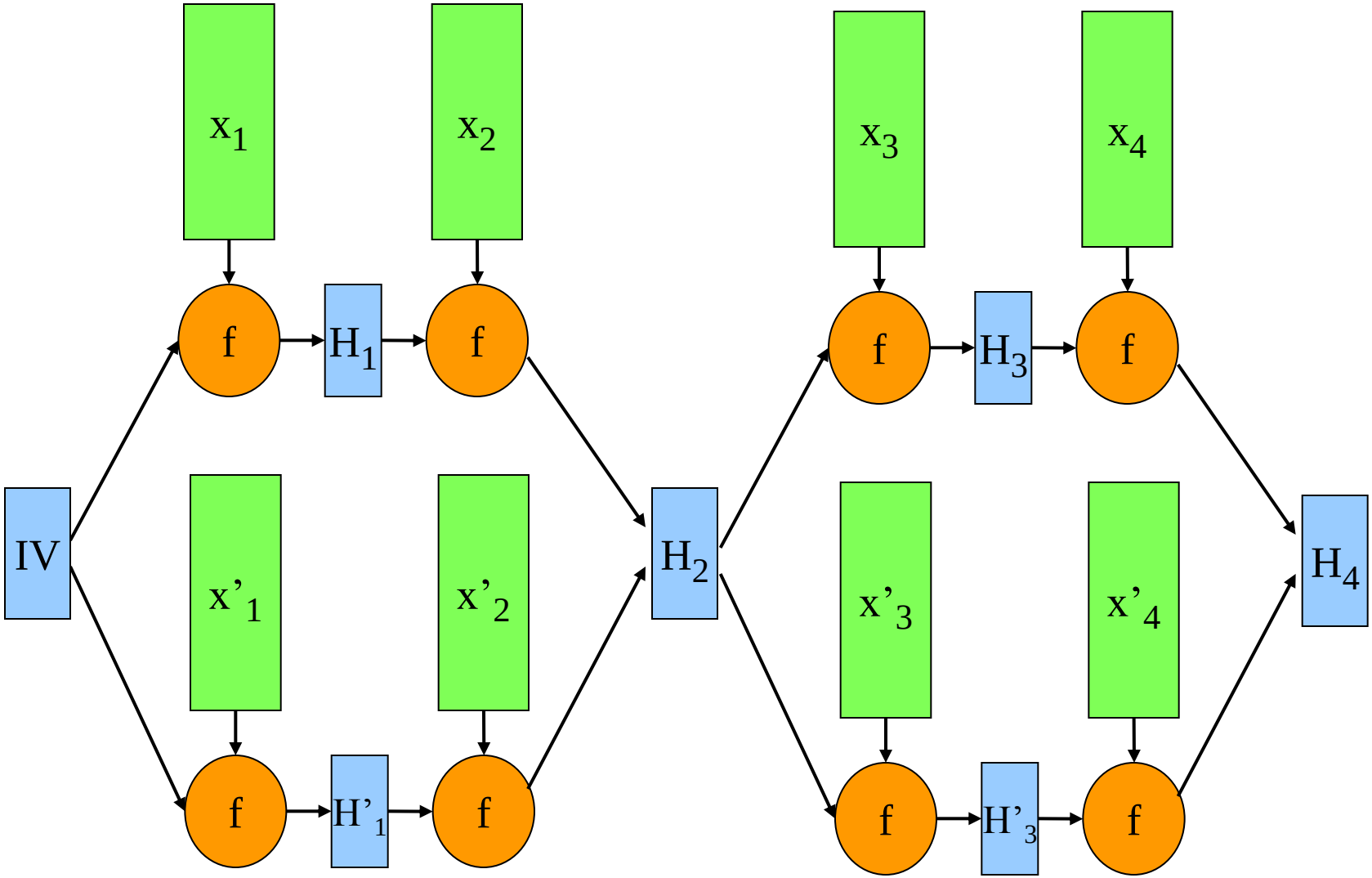
- Para producir un hash de tamaño n
- Usa una función de compresión f : f ($x_i$ , $H_{i-1}$) = $H_i$
  - $|x_i| = r \quad |H_i| = |H_{i-1}| = n$
  - ejemplo de MD5 : r = 512 bits,  n = 128 bits



# Una colisión se puede extender...



# Una colisión con IV libre se puede multiplicar...



# Objetivos de seguridad

- Para una función de hash de n bits
- resistencia a preimagen : $2^{n}$
- resistencia a colisiones : $2^{n/2}$
  - debido al ataque del cumpleaños

# Problema del cumpleaños

- Probabilidad de que entre k personas, dos cumplan el mismo día

- Probabilidad de que no hayan 2 que cumplan el mismo día
- $p(k) = 365/365 \cdot 364/365 \cdot \ldots (365 - (k-1)/365)$

- Si el año tiene n días
- $p(k) = 1 \cdot (1 - 1/n) \cdot (1 - 2/n) \cdot \ldots (1 - (k-1)/n)$

- Usamos el desarrollo de Taylor : $e^x = 1 + x + x^2/2 + \ldots$
- $p(n) \approx e^0 e^{-1/n} e^{-2/n} \ldots e^{-(k-1)/n} = \exp( -(k-1)k / 2n ) \approx \exp( -k^2 / 2n )$

# Ataque del cumpleaños

- Queremos estimar k para que $p(k) \approx 1/2$

- $\exp( -k^2 / 2n ) = 1/2$
  $-k^2 / 2n = \ln (1/2)$
  $k^2 = \ln(2)\ 2n$
  $k = (2 \ln(2))^{1/2}\ n^{1/2}$

- Este es el ataque del cumpleaños : si hay $2^{128}$ hashes necesitamos calcular unos $2^{64}$ para encontrar una colisión

# Función de hash MD5

- Diseñado por Rivest, después de que se encontraran colisiones en MD4
- Función de compresión que opera sobre bloques de r = 512 bits produce un hash de n = 128 bits
- Pensado para una arquitectura de 32 bits
  - mensaje = 16 enteros de 32 bits
  - hash = 4 enteros de 32 bits
  - estados internos de la función = enteros de 32 bits

# MD5 : variables

- Mensaje : bloque de 16 enteros * 32 bits = 512 bits
  - $m_0 \ldots m_{15}$
- Estados internos de la función de hash :
  - 64 pasos = 4 rondas * 16 pasos por ronda
  - $Q_0 \ldots Q_{63}$
- Valores iniciales (IVs) :
  - $Q_{-4}$ , $Q_{-3}$ , $Q_{-2}$ , $Q_{-1}$
- Salida :
  - $Q_{64}$ , $Q_{65}$ , $Q_{66}$ , $Q_{67}$

# MD5 : funciones auxiliares

- 4 funciones, una para cada ronda :
  - $F_1(X, Y, Z) = (X \,\&\, Y) \oplus (\sim X \,\&\, Z)$
  - $F_2(X, Y, Z) = (Z \,\&\, X) \oplus (\sim Z \,\&\, Y)$
  - $F_3(X, Y, Z) = X \oplus Y \oplus Z$
  - $F_4(X, Y, Z) = Y \oplus (X \mid \sim Z)$

- $X >> s$ = rotar a la derecha s posiciones (X es un entero de 32 bits)
- $X << s$ = rotar a la izquierda s posiciones
  - $s_0 \ldots s_{63}$ son constantes de rotación

- $+$ = suma modulo $2^{32}$
  - $k_0 \ldots k_{63}$ son constantes aditivas

# Primer ronda de MD5

- Se usa la función auxiliar $F_1(X,Y,Z) = (X \,\&\, Y) \,\hat{}\, (\sim X \,\&\, Z)$
- $Q_i = Q_{i-1} + ( F_1(Q_{i-1} , Q_{i-2} , Q_{i-3} ) + Q_{i-4} + m_i + k_i ) << s_i$
- $Q_0 = Q_{-1} + ( F_1(Q_{-1} , Q_{-2} , Q_{-3} ) + Q_{-4} + m_0 + k_0 ) << s_0$

  …

  $Q_{15} = Q_{14} + ( F_1(Q_{14} , Q_{13} , Q_{12} ) + Q_{11} + m_{15} + k_{15} ) << s_{15}$
- Cada $Q_i$ depende de los cuatro Qs anteriores y de $m_i$
- Se usan 4 registros e instrucciones muy rápidas

# Rondas 2, 3, 4

- Se usan las funciones auxiliares $F_2$ , $F_3$ , $F_4$

- Se vuelven a usar todos los $x_i$ del mensaje en cada ronda pero en otro orden

  $Q_i = Q_{i-1} + ( F_r(Q_{i-1} , Q_{i-2} , Q_{i-3} ) + Q_{i-4} + m_{wi} + k_i ) << s_i$

- Por ejemplo

  $Q_{16} = Q_{15} + ( F_2(Q_{15} , Q_{14} , Q_{13} ) + Q_{12} + m_1 + k_{16} ) << s_{16}$

# Trabajar sobre el estado interno

- En la primer ronda

  $Q_0 = Q_{-1} + ( F_1(Q_{-1} , Q_{-2} , Q_{-3} ) + Q_{-4} + m_0 + k_0 ) << s_0$

- Se puede calcular $x_0$ conociendo $Q_0$

  $(Q_0 - Q_{-1} ) >> s_0 = F_1(Q_{-1} , Q_{-2} , Q_{-3} ) + Q_{-4} + m_0 + k_0$

  $m_0 = (Q_0 - Q_{-1} ) >> s_0 - F_1(Q_{-1} , Q_{-2} , Q_{-3} ) - Q_{-4} - k_0$

- Por otra parte $m_0$ vuelve a usarse en la segunda ronda

  $Q_{19} = Q_{18} + ( F_2(Q_{18} , Q_{17} , Q_{16} ) + Q_{15} + m_0 + k_{19} ) << s_{19}$

  luego

  $m_0 = (Q_{19} - Q_{18} ) >> s_{19} - F_2(Q_{18} , Q_{17} , Q_{16} ) - Q_{15} - k_{19}$

# SHA 1 = Secure Hash Algorithm

- También está basado en MD4
- Produce un hash de 5 * 32 = 160 bits
- Usa las funciones auxiliares de MD4
  - $F_1(X, Y, Z) = (X \& Y) \oplus (\sim X \& Z)$
  - $F_2(X, Y, Z) = (X \& Y) \mid (X \& Z) \mid (Y \& Z)$
  - $F_3(X, Y, Z) = X \oplus Y \oplus Z$
- mas rotaciones a la izquierda y sumar constantes
- 4 rondas de 20 pasos
- se pueden esperar ataques similares a los de MD5

# Conclusión

# Continuará...

- Presentamos las primitivas criptográficas mas usadas en la práctica:
  - block ciphers
  - funciones de hash

- Introducción a la charla de Gerardo Richarte sobre algoritmos para encontrar colisiones en MD5

# Información de contacto

- Preguntas? Dudas?

  escribime a < carlos at coresecurity com >

- Más información sobre nuestros proyectos:

  www.coresecurity.com/corelabs/