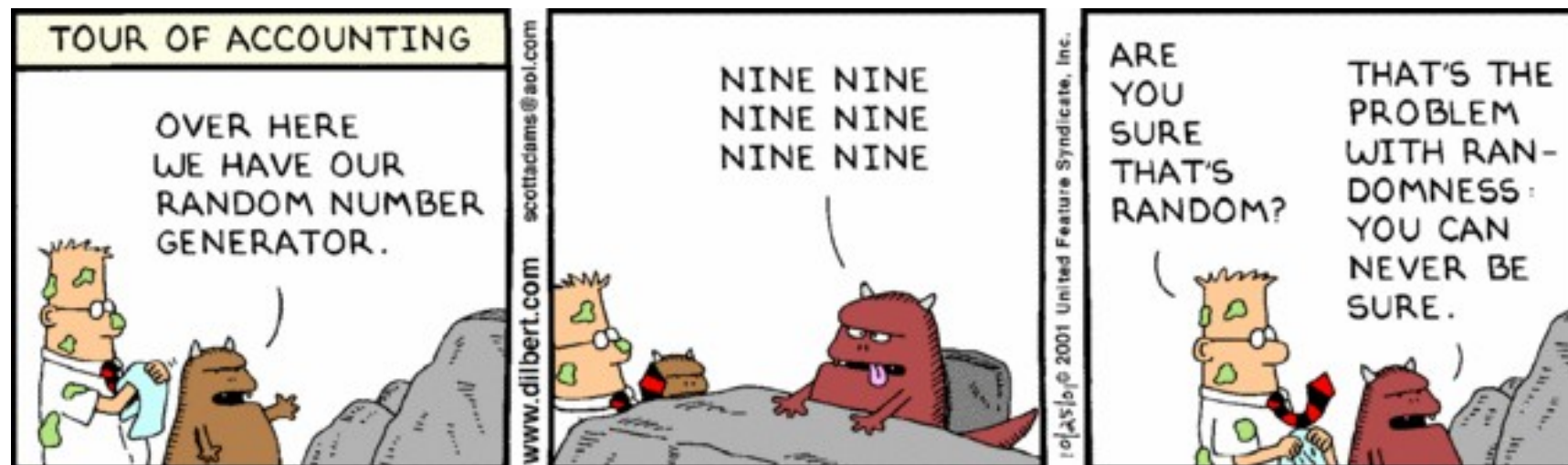


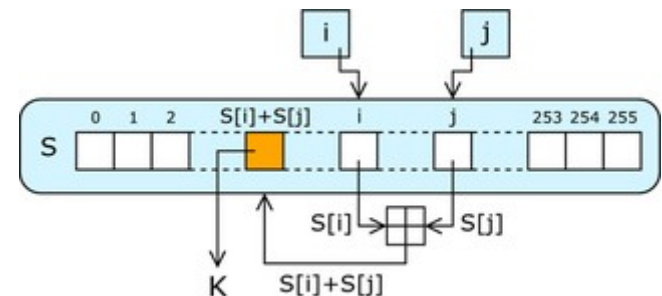
- Muy útiles para criptografía
  - Clave para cifrados de bloques (simétricos, asimétricos)
  - Flujo de clave para cifrado de flujo
  - Claves de sesión
  - Autenticación Challenge-response
- Muy difícil encontrar fuentes aleatorias verdaderas
  - no predecibles
  - no reproducibles

- Fuentes aleatorias verdaderas
  - Por ejemplo: ruido físico (radioactividad, ruido térmico en un semiconductor, ruido de un micrófono), estado de una computadora (reloj, posición del mouse, actividad de la red).
  - Combinar varias fuentes y usar una función resumen.
  - Métodos lentos (producen pocos bits).
- Fuentes pseudo-aleatorias
  - Generados por un algoritmo determinístico.
  - Parecen estadísticamente aleatorios.
  - Secuencia inicializada por una semilla.



- Diseñado por Ron Rivest (1987).
- Es uno de los cifradores de flujo más utilizados.
- Se encuentra incorporado en TLS/SSL y WEP.
- Marca registrada de RSA Security, apareció en Internet en 1994 (Arcfour).
- Opera con bytes y genera bytes pseudoaleatorios.
- Opera de modo sincrónico.
- Es particularmente eficiente en implementaciones de software.

- RC4 genera un flujo pseudo-aleatorio de bits (un keystream) que, para cifrado, se combina con el texto plano usando la función XOR. La fase de descifrar el mensaje se realiza del mismo modo.
- Para generar el keystream, el algoritmo de cifrado tiene un estado interno secreto y utiliza dos funciones.
  - Un algoritmo de programación de claves (Key scheduling algorithm o KSA) (se inicializa con una clave de entre 40 y 256 bits)
  - Un algoritmo de generación pseudo-aleatoria (pseudo-random generation algorithm o PRGA).



# DES - Data Encryption Standard

En 1973 el National Bureau of Standards (USA) llama a concurso para buscar un algoritmo criptográfico estándar.

En 1974 la NSA National Security Agency (USA) declara desierto el primer concurso, publica una segunda especificación y elige Lucifer con algunas variaciones.

En 1976 se adopta DES como estándar y se autoriza para ser usado en las comunicaciones no clasificadas del gobierno (USA).

# Características

## Cifrado en bloque

- cifra bloques de 64 bits utilizando una clave de 64 bits (8 bytes con paridad, o sea 7 bytes útiles).
- produce bloques cifrados de 64 bits.

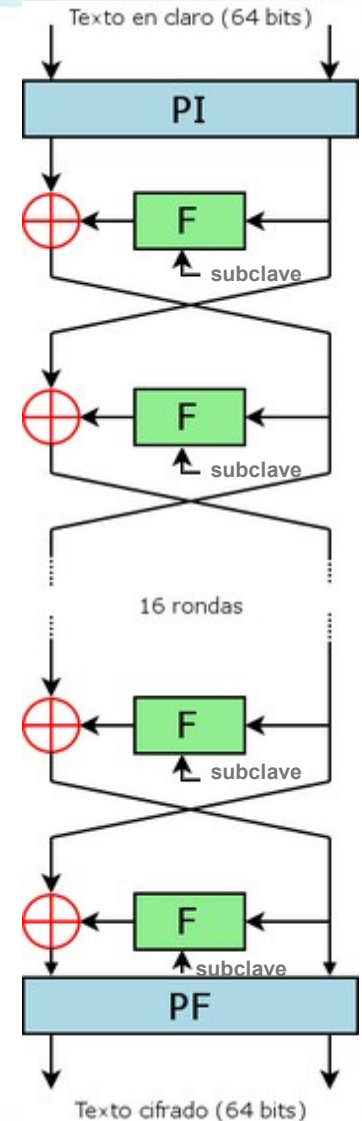
## Cifrado por producto

- la unidad básica es el bit.
- realiza sustituciones y transposiciones (permutaciones) sobre los bits.

Consiste de 16 rondas (iteraciones) y en cada una utiliza una clave generada a partir de la clave suministrada originalmente.

# Esquema general

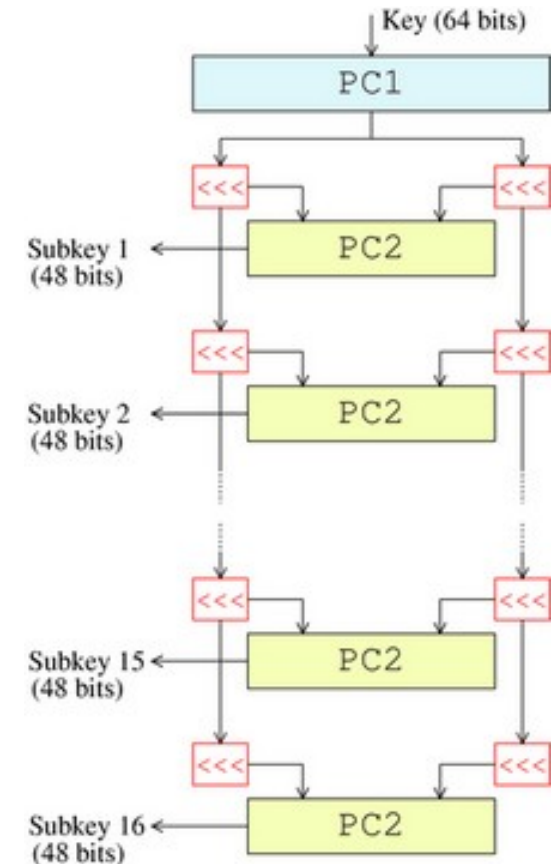
- Existe una permutación inicial y una final ( $PI$  y  $PF$ )
- Hay 16 rondas que:
  - Dividen el bloque en dos mitades.
  - Pasan una mitad por una función  $F$  de Feistel.
  - Mezclan ambas mitades con una operación XOR.
  - Rotan ambas mitades.





# Generación de subclaves

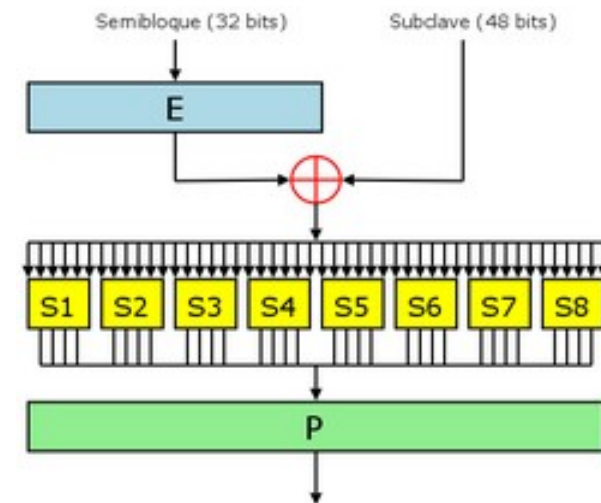
- Se permutan y eligen 56 bits de la clave inicial (*PC1*).
- Los 8 bits restantes se descartan o se usan como paridad.
- Los 56 bits se dividen en dos mitades de 28 bits.
- En cada ronda ambas mitades se desplazan hacia la izquierda uno o dos bits (dependiendo de la ronda), y se seleccionan los 48 bits de la subclave permutando cada mitad (*PC2*) y seleccionando 24 bits en cada una.



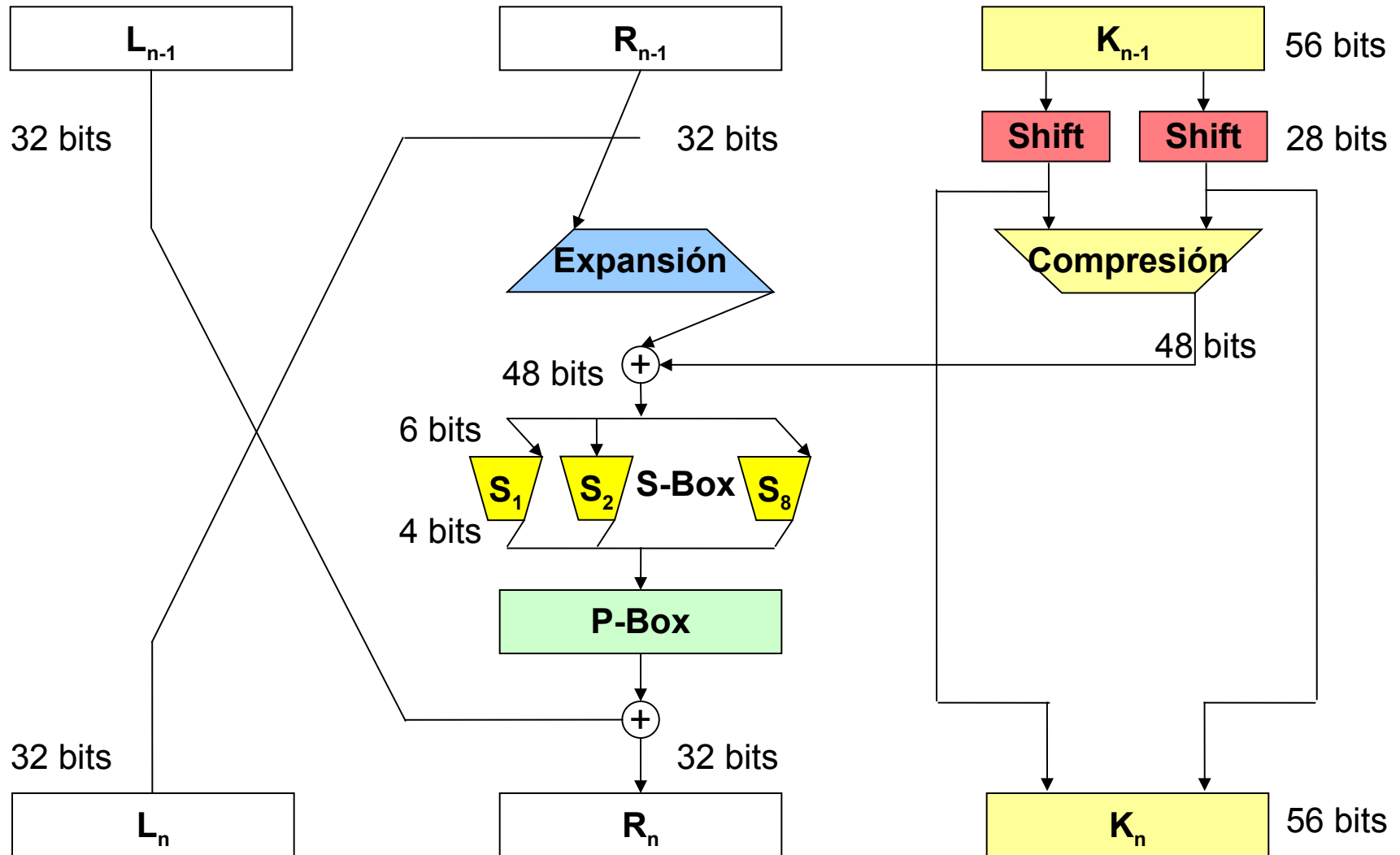
# Función $F$

La función  $F$  trabaja con bloques de 32 bits y consta de 4 pasos:

- *Expansión*: el bloque de 32 bits se expande a 48 bits mediante *permutación de expansión*, duplicando algunos de los bits.
- *Mezcla*: el resultado se combina con una *subclave* utilizando una operación XOR.
- *Sustitución*: el bloque es dividido en ocho trozos de 6 bits antes de ser procesados por las cajas de sustitución. Cada una de las ocho S-Box reemplaza sus seis bits de entrada con cuatro bits de salida, de acuerdo con una transformación no lineal, especificada por una tabla de búsqueda.
- *Permutación*: finalmente, las 32 salidas de las S-Box se reordenan de acuerdo a una permutación fija; la P-Box.



# Funcionamiento



DES posee propiedades no deseadas:

- Tiene 4 claves débiles
  - Son su propia inversa
- Tiene 6 pares de claves semi débiles
  - Sus inversas son otras claves
- Propiedad complementaria
  - $DES_k(m) = c \Rightarrow DES_k(m') = c'$
- Las S-Box tiene propiedades irregulares
  - La distribución de los números pares e impares no es aleatoria.
  - La salida de la cuarta S-Box depende de la entrada de la tercera S-Box.

## Fuerza bruta

- Consiste en probar una por una cada posible clave.

## Criptografía diferencial (finales de los '80)

- Consiste en usar pares de textos en claro elegidos y analizar como evolucionan a medida que se ejecutan las rondas cuando se cifran con la misma clave.
- Requiere  $2^{47}$  textos en claro elegidos.

## Criptografía lineal (1993)

- Requiere  $2^{43}$  textos en claro elegidos.

# Modos de operación

## Electronic Code Book (ECB)

Cifrar cada bloque de manera independiente

## Cipher Block Chaining (CBC)

XOR de cada bloque con el texto cifrado anterior

## Encrypt-Decrypt-Encrypt (EDE)

Utiliza dos claves:  $k, k'$ , tamaño de clave 112. Si  $k = k'$  es igual a DES.

$$c = \text{DES}_k(\text{DES}_{k'}^{-1}(\text{DES}_k(m)))$$

## Encrypt-Encrypt-Encrypt (TripleDES)

Utiliza tres claves:  $k, k', k''$ , tamaño de clave 168.

$$c = \text{DES}_k(\text{DES}_{k'}(\text{DES}_{k''}(m)))$$

# Historia de DES

- El NIST certifica al DES en 1987 y luego en 1993.
- Entre 1997 y 1999 el DES se enfrenta a tres desafíos (DES Challenge) que impulsa y promociona la compañía RSA.
- En 1997 NIST no certifica DES y llama concurso internacional para buscar un nuevo algoritmo de cifrado.
- En 2001 NIST selecciona al algoritmo Rijndael como sucesor de DES. Lo llama Advanced Encryption Standard (AES).
  - Es diseñado para soportar los ataques que fueron exitosos sobre DES.

# AES - Advanced Encryption Standard

Es un cifrador de bloque y un cifrador por producto.

Opera con bloques y claves de longitudes variables, que pueden ser especificadas independientemente a 128, 192 ó 256 bits (las 9 combinaciones son posibles), extendiéndose fácilmente a múltiplos de 32 bits, lo que fortalece aún más su uso futuro.

Permite una implementación eficiente y rápida en software y hardware. Esto lo hace adecuado para la casi totalidad de las aplicaciones actuales.



## Problemas:

- Distribución de las claves:
  - Es necesario que el receptor conozca la clave que se va a utilizar
  - No es posible utilizar medios inseguros para la comunicación de las claves.
- Complejidad en la gestión de las claves

La invención del concepto de criptografía de clave pública se atribuye a Diffie y a Hellman (1976).

Nace como una solución al problema de distribución de claves, permitiendo que dos partes puedan acordar una clave en común sobre canales inseguros.

La criptografía asimétrica propone un mecanismo en el cual el descifrado de la información se realiza con una clave distinta a la clave de cifrado.

Se basa en la existencia de dos claves:

- *Clave pública* disponible para todos.
- *Clave privada* conocida solo por un individuo.

Idea:

- Confidencialidad: cifrar usando la *clave pública del destinatario*, descifrar usando su *clave privada*.
- Integridad/Autenticación: cifrar usando la *clave privada del emisor*, descifrar usando su *clave pública*.

Todo criptosistema de clave pública debe cumplir las siguientes condiciones:

- Dada la clave apropiada debe ser computacionalmente fácil cifrar y descifrar un mensaje.
- Debe ser computacionalmente imposible derivar la clave privada a partir de la clave pública.
- Debe ser computacionalmente imposible determinar la clave privada a partir de un ataque de texto en claro elegido.

Alice y Bob eligen dos números  **$p$**  y  **$g$**  tal que  $p$  es primo y  $g$  es un entero tal que  $2 \leq g \leq p-1$ .

Alice elige un entero  **$a$** , y envía a Bob el resultado de:  
 **$g^a \bmod p$**

Bob elige un entero  **$b$** , y envía a Alice el resultado de:  
 **$g^b \bmod p$**

Alice calcula  **$k = (g^b \bmod p)^a \bmod p$**

Bob calcula  **$k' = (g^a \bmod p)^b \bmod p$**

$k$  y  $k'$  son iguales a  $g^{ab} \bmod p$

# Funcionamiento (conceptual)

Tomamos  $p = 23$ ,  $g = 3$

Alice elige:

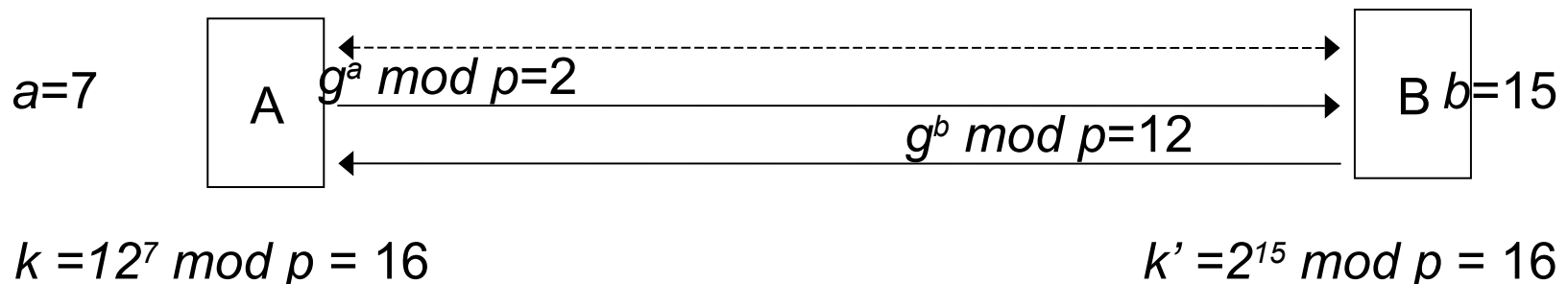
$a = 7$  y envia  $g^a \bmod p$ , o sea  $3^7 \bmod 23 = 2$

Bob elige:

$b = 15$  y envia  $g^b \bmod p$ , o sea  $3^{15} \bmod 23 = 12$

Alice calcula  $k = (12)^a \bmod p = (12)^7 \bmod 23 = 16$

Bob calcula  $k' = (2)^b \bmod p = (2)^{15} \bmod 23 = 16$



El algoritmo Diffie-Hellman se basa en operaciones matemáticas de exponenciación y en el problema de obtener el logaritmo discreto. Requiere el uso de números muy grandes.

- resolver  $r = g^a \bmod p$ , donde  $a$  es la incognita.

Solo  $a$  y  $b$ , y  $g^{ab} = g^{ba}$  se mantienen en secreto.

No provee autenticación y es vulnerable a un ataque Man-in-the-middle.

Más Información: RFC 2631

Algoritmo creado en 1977 por Ron Rivest, Adi Shamir y Len Adleman del MIT.

Puede ser usado para cifrado y firma de mensajes.

Se basa en el problema de la factorización de números muy grandes.

Es un cifrador exponencial.

Más información: RFC 2313 (RSA v1.5)



- Función  $\phi(n)$ 
  - Número de enteros positivos menores que  $n$  que son coprimos con  $n$ 
    - *Coprimo = que no tiene factores en común con  $n$*
- Por ejemplo:  $\phi(10) = 4$ 
  - 1, 3, 7, 9 son coprimos con 10
- Otro ejemplo:  $\phi(21) = 12$ 
  - 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20 son coprimos con 21

Consiste en elegir dos números primos grandes  $p$ ,  $q$  tal que  $p \neq q$ .

- Sea  $n = pq$ ; se puede demostrar que

$$\phi(n) = (p-1)(q-1)$$

- Elegir  $e < n$  tal que  $e$  es coprimo con  $\phi(n)$ .

- Calcular  $d$  la solución de la ecuación

$$ed \bmod \phi(n) = 1$$

dicha solución existe y es única.

Clave pública: **( $e$ ,  $n$ )**

Clave privada:  **$d$**

Dado un mensaje  $m$ , se lo divide en bloques de longitud menor a  $n$ .

Para **cifrar** el mensaje se calcula lo siguiente:

$$c_i = m_i^e \bmod n$$

Para **descifrar** el mensaje se toma cada bloque cifrado y se calcula:

$$m_i = c_i^d \bmod n$$

Tomamos  $p = 7$ ,  $q = 11$

Entonces:

$$n = p * q = 7 * 11 = 77$$

$$\phi(n) = (p-1) * (q-1) = 6 * 10 = 60$$

Se elige  $e = 17$ , y por lo tanto  $d = 53$  que cumple  
 $e^d \bmod \phi(n) = 1$

Alice toma como clave privada  **$d = 53$**  y como clave publica  **$(e, n) = (17, 77)$**

Bob quiere enviarle a Alice el mensaje secreto HELLO  
(07 04 11 11 14)

Para cifrar usa la clave pública de Alice ( **$e$** ,  **$n$** ), y calcula  
 **$c = m^e \bmod n$** , es decir  **$c = m^{17} \bmod 77$** .

- $07^{17} \bmod 77 = 28$
- $04^{17} \bmod 77 = 16$
- $11^{17} \bmod 77 = 44$
- $11^{17} \bmod 77 = 44$
- $14^{17} \bmod 77 = 42$

Enviando 28 16 44 44 42

Alice recibe 28 16 44 44 42

Usa su clave privada,  $d = 53$ , y calcula  $m = c^d \bmod n$ , es decir  $m = c^{53} \bmod 77$  para descifrar el mensaje:

- $28^{53} \bmod 77 = 07$  (H)
- $16^{53} \bmod 77 = 04$  (E)
- $44^{53} \bmod 77 = 11$  (L)
- $44^{53} \bmod 77 = 11$  (L)
- $42^{53} \bmod 77 = 14$  (O)

Nadie que no sea Alice puede leer el mensaje que Bob le envió, ya que solo ella conoce su clave privada.

Tomamos  $p = 7$ ,  $q = 11$

Entonces:

$$n = p * q = 7 * 11 = 77$$

$$\phi(n) = (p-1) * (q-1) = 6 * 10 = 60$$

Se elige  $e = 17$ , y por lo tanto  $d = 53$  que cumple  
 $e^d \bmod \phi(n) = 1$

Alice toma como clave privada  **$d = 53$**  y como clave publica  **$(e, n) = (17, 77)$**

Alice quiere enviar el mensaje HELLO (07 04 11 11 14) de tal forma que Bob sepa que el mensaje recibido es de Alice y que no sufrió cambios durante el envío.

Usa su clave privada,  $d = 53$ , y calcula  $c = m^d \bmod n$ , es decir  $c = m^{53} \bmod 77$  para cifrar el mensaje:

- $07^{53} \bmod 77 = 35$
- $04^{53} \bmod 77 = 09$
- $11^{53} \bmod 77 = 44$
- $11^{53} \bmod 77 = 44$
- $14^{53} \bmod 77 = 49$

Enviando 35 09 44 44 49



Bob recibe 35 09 44 44 49

Usa la clave pública de Alice (**e**, **n**), y calcula  **$m = c^e \bmod n$** , es decir  **$m = c^{17} \bmod 77$**  para descifrar el mensaje:

- $35^{17} \bmod 77 = 07$  (H)
- $09^{17} \bmod 77 = 04$  (E)
- $44^{17} \bmod 77 = 11$  (L)
- $44^{17} \bmod 77 = 11$  (L)
- $49^{17} \bmod 77 = 14$  (O)

Bob sabe que Alice envió el mensaje ya que solo ella conoce la clave privada que se utilizó para cifrarlo.

Si los bloques del mensaje (letras) hubiesen sido alterados el mismo no se descifraría de manera correcta.

# Todo junto

Alice quiere enviarle a Bob el mensaje HELLO cifrado y autenticado.

- Alice: clave pública (17, 77); clave privada 53
- Bob: clave pública (37, 77); clave privada 13

Alice cifra HELLO (07 04 11 11 14) con su clave privada, luego lo cifra con la clave pública de Bob:

- $(07^{53} \bmod 77)^{37} \bmod 77 = 07$
- $(04^{53} \bmod 77)^{37} \bmod 77 = 37$
- $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
- $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
- $(14^{53} \bmod 77)^{37} \bmod 77 = 14$

Envia 07 37 44 44 14

La seguridad de RSA depende completamente del problema de factorizar números grandes.

Requiere la utilización de un esquema de padding.

Tipos de ataques:

- Texto cifrado elegido (firma de texto aleatorios)
- Módulo común (exponentes diferentes coprimos)
- Exponente de cifrado bajo.
- Exponente de descifrado bajo.

# Criptografía de curva elíptica (ECC)

Es una variante de criptografía asimétrica, que suele requerir claves de menor tamaño.

Se basa en el problema de los logaritmos discretos.

## Equipo de prueba: Intel I5-7200

	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
des cbc	69715.93k	71967.38k	72461.53k	72399.19k	72662.22k
des ede3	26534.38k	26933.45k	26862.93k	26939.08k	27025.38k
aes-128 cbc	123492.28k	137312.26k	140304.61k	141960.31k	142415.19k
aes-256 cbc	91444.94k	98985.89k	99997.18k	100822.56k	101212.16k

(1000s de bytes por segundo)

	sign	verify	sign/s	verify/s
rsa 1024 bits	0.000103s	0.000006s	9665.7	153958.0
rsa 2048 bits	0.000667s	0.000019s	1499.9	51859.6
rsa 4096 bits	0.004479s	0.000068s	223.2	14609.2

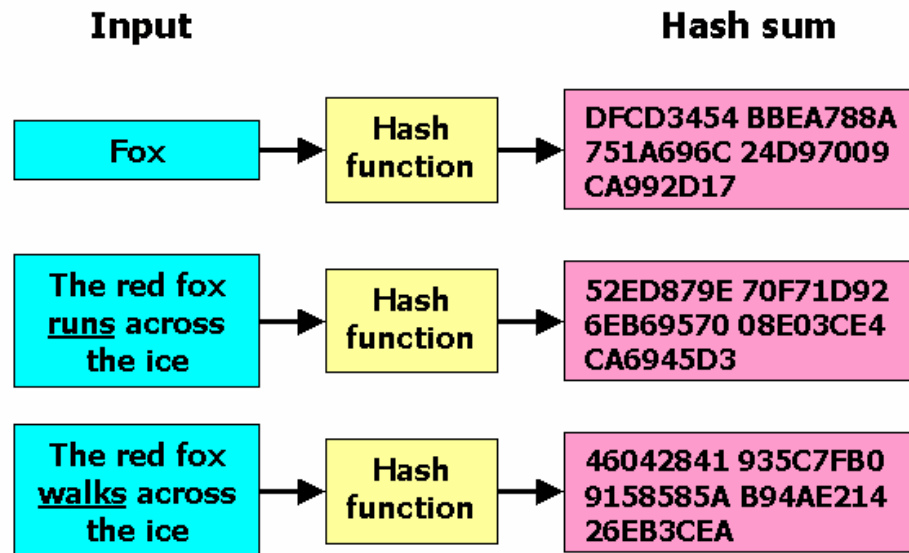
Usados para detección de errores:

- Bit de Paridad: Dígito binario que indica si la cantidad de bits en 1 en un conjunto de bits precedente es par o impar. Ejemplo: DES usa 8 bytes para la clave, pero el 8 bit de cada byte se usa para paridad.
- Cyclic Redundancy Check: Se basan en la idea de dividir polinomios y utilizar el resto.

Por ejemplo: CRC-16:  $g(x)=1+x^2+x^{15}+x^{16}$

# Funciones de hashing

La función de hash transforma un mensaje de entrada  $M$  de longitud variable en una cadena de salida de longitud fija  $H(M)$ . Debe ser fácil de computar.



Propiedades de las funciones de hashing criptográficas:

1. Resistencia a preimágenes

Dado  $z$  es computacionalmente no factible hallar algún  $x$  tal que  $h(x) = z$

2. Resistencia a segundas preimágenes

Dado  $x$  es computacionalmente no factible hallar  $x' \neq x$  tal que  $h(x) = h(x')$

3. Resistencia a colisiones

Es computacionalmente no factible hallar  $x, x'$  (con  $x \neq x'$ ) tal que  $h(x) = h(x')$

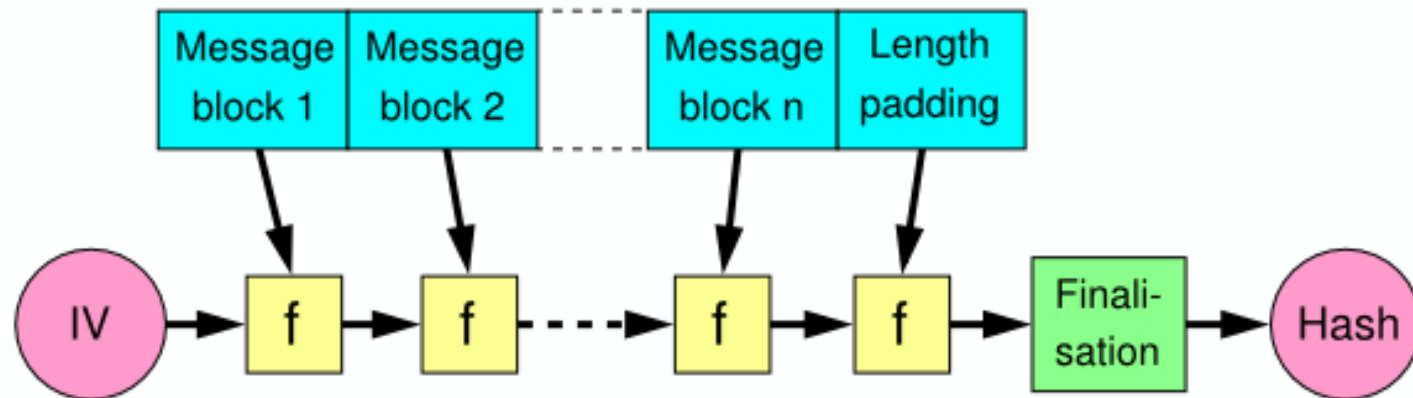
Si cumple 1. y 2. se denomina *función de hashing de una vía o función débil de una vía* (OWHF).

Si cumple 2. y 3. (y eventualmente 1.) se denomina *resistente a colisiones o función fuerte de una vía* (CRHF).



# Funciones de hashing criptográficas

La longitud ( $n$ ) es típicamente: 128, 160, 256 o 512 bits.  
Las funciones más difundidas son: MD5, SHA, SHA-1, RIPEMD160, SHA-256, SHA-512, SHA-3.



Merkle–Damgård hash construction: MD5, SHA1, SHA2.

## MD5: Modo de operación

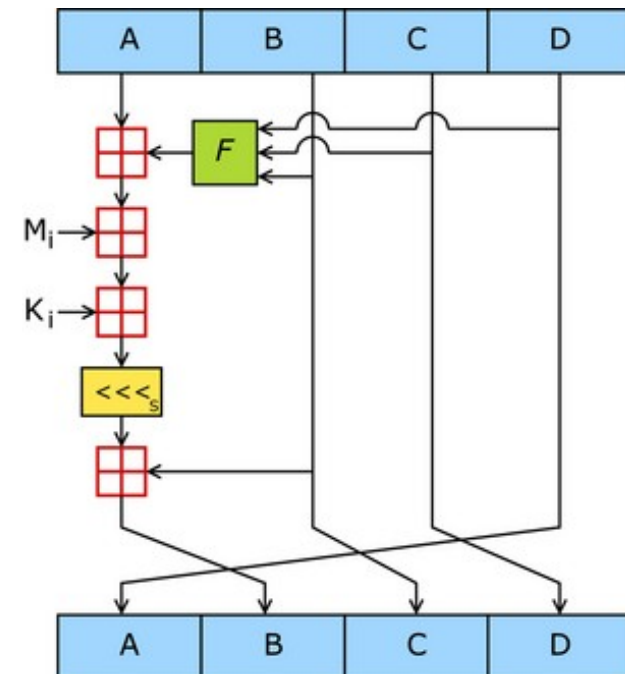
- Recibe un mensaje de longitud variable.
- Devuelve una cadena de 128 bits.
- El mensaje de entrada es cortado en bloques de 512 bits.
- El mensaje es rellenado para completar el último bloque.
- El relleno se hace colocando un bit en 1 y a continuación tantos 0 como sea necesario, dejando 64 bits al final para colocar la longitud del mensaje original.

Más información: RFC 1321

## MD5: Modo de operación

- El algoritmo opera con un estado de 128 bits, dividido en 4 words de 32 bits ( $A$ ,  $B$ ,  $C$  y  $D$ ).
- $A$ ,  $B$ ,  $C$  y  $D$  se inicializan con valores constantes.
- Cada bloque de 512 bits modifica el estado.
- El procesamiento de cada bloque tiene 4 etapas llamadas *rounds*.
- Cada vuelta se compone de 16 operaciones similares utilizando:
  - Suma modulo 32
  - Rotación a izquierda de  $s$  bits
  - Una función no lineal  $F$  diferente en cada *round*.

$F$  es una función no lineal; se usa una función distinta en cada vuelta.  $M_i$  denota un bloque de 32-bit del mensaje de entrada, y  $K_i$  denota una constante de 32-bit, diferente para cada operación.



$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z)$$

- En la actualidad, no todas las funciones de hash se consideran seguras.

Como se verá en los ejemplos, no se debe usar MD5 ni SHA1 para nuevas aplicaciones (ver <https://shattered.io/>).

- Es común usar alguna variante de sha-2 (sha-256, sha-384, sha-512).
- Competencia SHA-3. Se eligió la función Keccak como nuevo standard (<http://keccak.noekeon.org/>) en agosto de 2015 (FIPS 202)

- Uso correcto:
  - Utilizar un hash cuando se puede distribuir de manera segura  $H(x)$  y se desea verificar que un valor  $x'$ , recibido de manera insegura, es de hecho igual a  $x$ .
- Uso incorrecto:
  - Distribuir  $H(x)$  y  $x$  por el mismo medio: si modifico  $x$  también puedo modificar  $H(x)$ .
  - Utilizar  $H(x)$  como una firma: cualquiera puede calcular  $H(x)$ .

Equipo de prueba: Intel I5-7200

Software: openssl

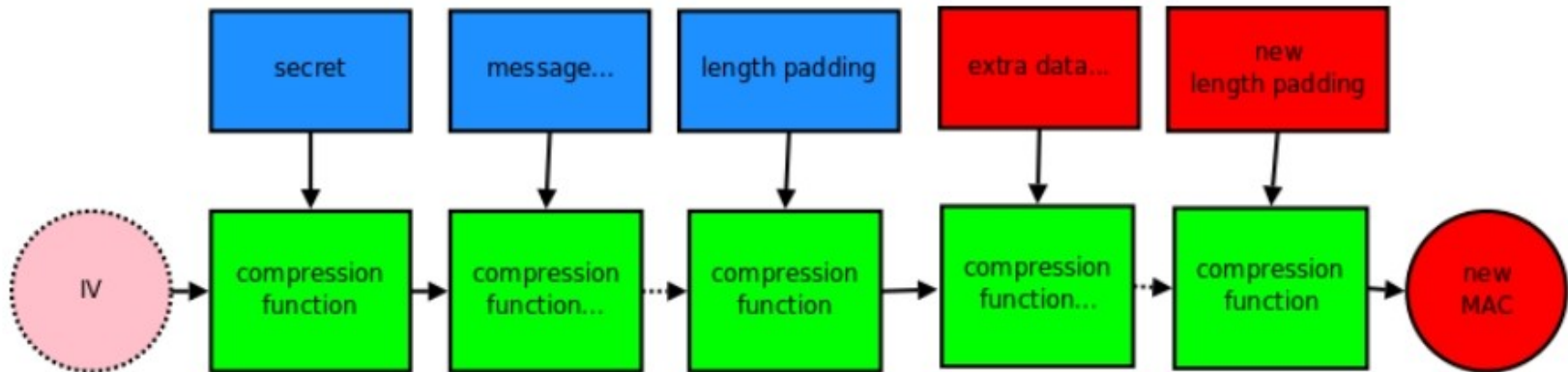
type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
md5	61226.15k	177574.63k	382561.88k	526483.46k	610956.63k
sha1	74659.35k	210613.57k	492783.62k	737642.50k	861978.62k
rmd160	41477.38k	97536.64k	174338.22k	215652.01k	234651.65k
sha256	69876.08k	159183.17k	296403.03k	372265.30k	401047.55k
sha512	50480.67k	200609.02k	342099.29k	509153.28k	594537.13k

(1000s de bytes por segundo)

- Las funciones de hash vistas (MD5, SHA-1, etc.) no han sido diseñadas para la autenticación al carecer de clave secreta.
- Un Message Authentication Code (MAC)  $MAC_k()$  ideal utiliza una clave para mapear entradas de tamaño arbitrario en salidas de tamaño fijo ( $n$ ) de forma tal que toma tiempo  $2^n$  generar un par  $(x, MAC_k(x))$  aun si se proveen pares válidos.
- A diferencia de una función de hash, conocer  $MAC_k(x)$  no permite computer  $MAC_k(y)$  para algún otro  $y$ .
- Al combinar hash y clave secreta, obtenemos un MAC, que podemos utilizar para detectar manipulación de contenido.
- Es un mecanismo muy rápido.

# Ataque a funciones de hash

- Length Extension Attack:



- Funciones de hash que utilizan la construcción Merkle–Damgård son vulnerables. No afecta a sha-3
- Solución: double-hashing  $h(h(m))$ ?



$$\text{HMAC}_k(m) = h((k \oplus \text{opad}) \parallel h((k \oplus \text{ipad}) \parallel m))$$

donde  $\text{opad} = 0x5c5c5c\dots5c$ ,  $\text{ipad} = 0x363636\dots36$

Ejemplos: HMAC-MD5, HMAC-SHA1

Utilizados en SSL, IPSEC para detectar cambios.

Más información: RFC 2104

# Usos y mal usos de funciones HMAC

Garantizar que no haya dos mensajes diferentes que resulten en la misma entrada a ser protegida por el HMAC.

- Ej: AWS Signature V1 firma el Query String HTTP de la siguiente forma:
  - Partir el query string en los caracteres '&' y '=' para obtener una serie de clave-valor.
  - Ordenar los pares según las claves.
  - Concatenar: key1 + value1 + key2 + value2 + ...
  - HMAC-sha1 de la concatenación.

<http://www.daemonology.net/blog/2008-12-18-AWS-signature-version-1-is-insecure.html>

- Tener cuidado con los side-channels relacionados con el tiempo a la hora de verificar la firma. Si la función de comparación aborta al encontrar que el primer byte donde la firma difiere se pueden ir adivinando iterativamente los bytes.

# HMAC: No reinventar la rueda

- ¿ Por qué usar  
$$\text{HMAC}_k(m) = h((k \oplus \text{opad}) \parallel h((k \oplus \text{ipad}) \parallel m)) ?$$
- ¿ Si uso  $\text{HMAC}_k(m) = h(k \parallel m) ?$   
¡Length extension attack! (Ej. API de Flickr,  
[https://ia902205.us.archive.org/15/items/pdfy-HaRR7XMfTOb\\_RrU0/flickr\\_api\\_signature\\_forgery.pdf](https://ia902205.us.archive.org/15/items/pdfy-HaRR7XMfTOb_RrU0/flickr_api_signature_forgery.pdf))
- ¿ Si uso  $\text{HMAC}_k(m) = h(m \parallel k) ?$ 
  - Solucioné Length Extension Attack
  - ¿Qué pasa si puedo encontrar una colisión?
    - Sea  $m' / h(m) = h(m')$ , entonces  $h(m \parallel k) = h(m' \parallel k)$
    - En MD5 encontrar una colisión de prefijo-elegido toma menos de 1 minuto.