Números aleatorios

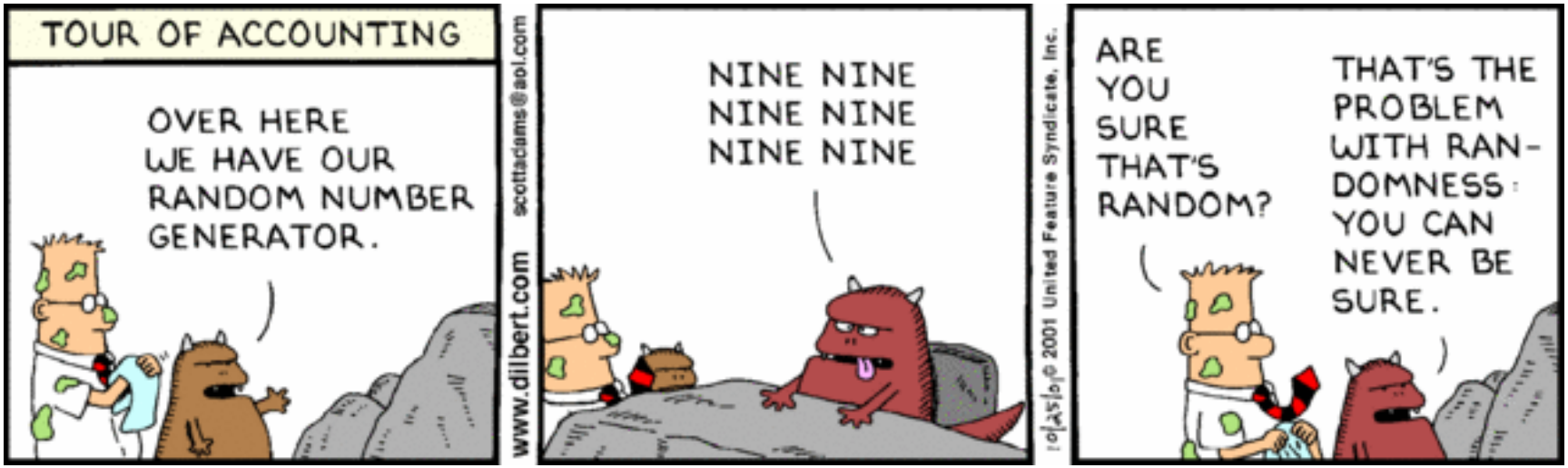
DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

- 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

Números aleatorios

- 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.


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
www.dilbert.com scottadams@aol.com
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
© 2001 United Feature Syndicate, Inc.

LFSR - Registros de desplazamiento con retroalimentación lineal

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

- Genera números pseudo-aleatorios, que pueden ser utilizados para cifradores de flujo.

- Un *n*-stage LFSR consiste en:

  - Un registro de *n* bits $r = r_0 \ldots r_{n-1}$
  - Una secuencia de *n* bits $t = t_0 \ldots t_{n-1}$
  - Funcionamiento:
    - Usar r_{n-1} como bit clave
    - Calcular $x = r_0 t_0 \oplus \ldots \oplus r_{n-1} t_{n-1}$
    - Desplazar *r* un bit a la derecha, descartando r_{n-1}, *x* pasa a ser r_0

LFSR - Registros de desplazamiento con retroalimentación lineal

4-stage LFSR; t = 1001

r	k_i	*cálculo del nuevo bit*	*nuevo r*
0010	0	01⊕00⊕10⊕01 = 0	0001
0001	1	01⊕00⊕00⊕11 = 1	1000
1000	0	11⊕00⊕00⊕01 = 1	1100
1100	0	11⊕10⊕00⊕01 = 1	1110
1110	0	11⊕10⊕10⊕01 = 1	1111
1111	1	11⊕10⊕10⊕11 = 0	0111

……

La clave tiene un período igual a 15 (010001111010110)

LFSR - Registros de desplazamiento con retroalimentación lineal

- La secuencia generada depende del estado inicial
- El mismo estado genera la misma secuencia
- Existen 2^n estados posibles
- Periodo máximo de 2^n-1 (sólo bajo ciertas condiciones)

NLFSR - Registros de desplazamiento con retroalimentación no lineal

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

- Un *n*-stage NLFSR consiste en:
  - Un registro de *n* bits $r = r_0 \ldots r_{n-1}$
  - Funcionamiento:
    - Usar r_{n-1} como bit clave
    - Calcular $x = f(r_0, \ldots, r_{n-1})$; *f* es cualquier función
    - Desplazar *r* un bit a la derecha, descartando r_{n-1}, *x* pasa a ser r_0
- El funcionamiento es el mismo que LFSR pero la función de reemplazo de bits es más general.

NLFSR - Registros de desplazamiento con retroalimentación no lineal

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

4-stage NLFSR; $f(r_0, r_1, r_2, r_3) = (r_0 \,\&\, r_2) \mid r_3$

r	k_i	*cálculo del nuevo bit*	*nuevo r*
1100	0	(1 & 0) | 0 = 0	0110
0110	0	(0 & 1) | 0 = 0	0011
0011	1	(0 & 1) | 1 = 1	1001
1001	1	(1 & 0) | 1 = 1	1100
1100	0	(1 & 0) | 0 = 0	0110
0110	0	(0 & 1) | 0 = 0	0011
0011	1	(0 & 1) | 1 = 1	1001

La clave tiene un periodo igual a 4 (0011)

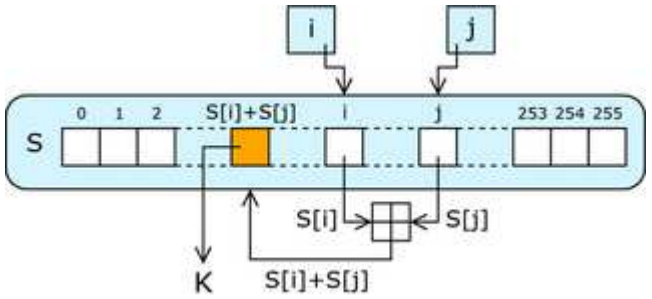
- Los NLFSR se usan menos porque es difícil diseñarlos de manera tal que produzcan un periodo largo.

RC4

- 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 aleatorios.
- Opera de modo sincrónico.
- Es particularmente eficiente en implementaciones de software.

RC4

- 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 Bureaux 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.

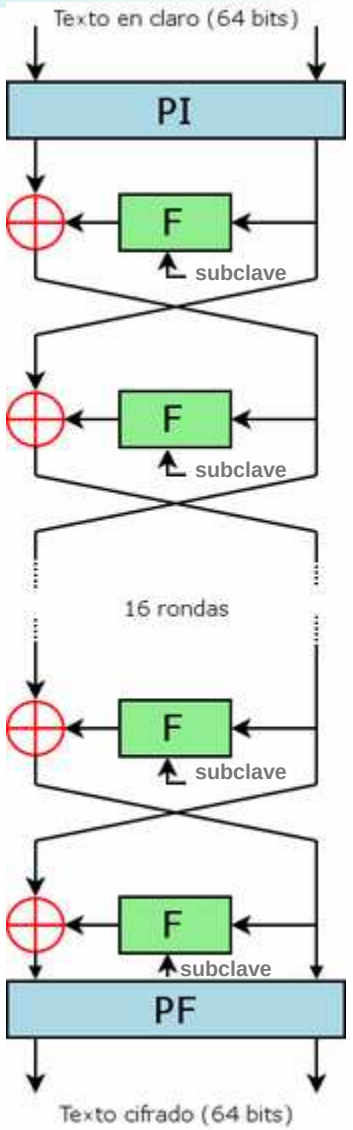
Cifrado por producto

- 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.

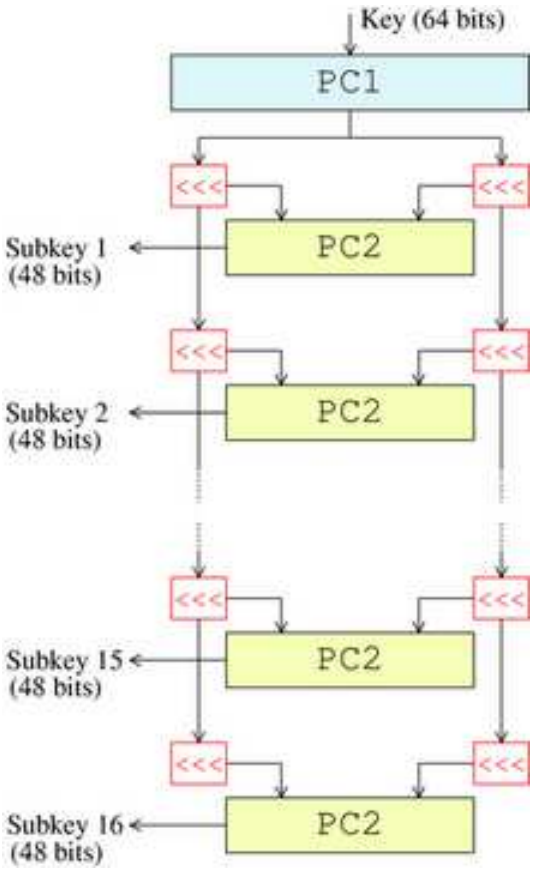
Esquema general

- 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 ambos mitades con una operación XOR.
  - Rotan ambas mitades.



Generación de subclaves

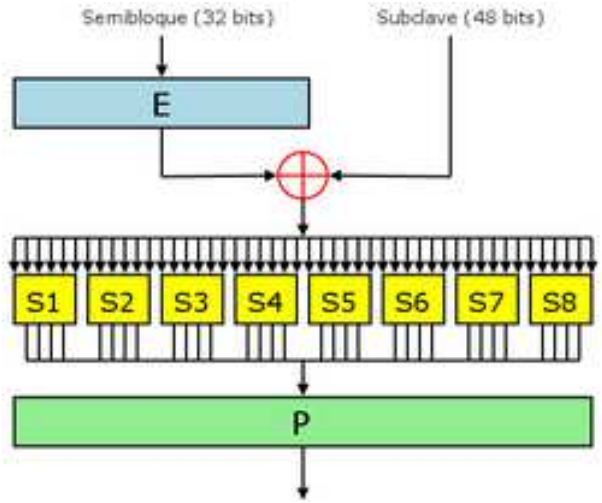
- Se permutan y eligen 56 bits de la clave inicial (*PC1*).
- Los 8 bits restantes se descartan o se usan como paridad.
- Los 56 bits se dividen en dos mitades de 28 bits.
- En cada ronda ambas mitades se desplazan hacia la izquierda uno o dos bits (dependiendo de la ronda), y se seleccionan los 48 bits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



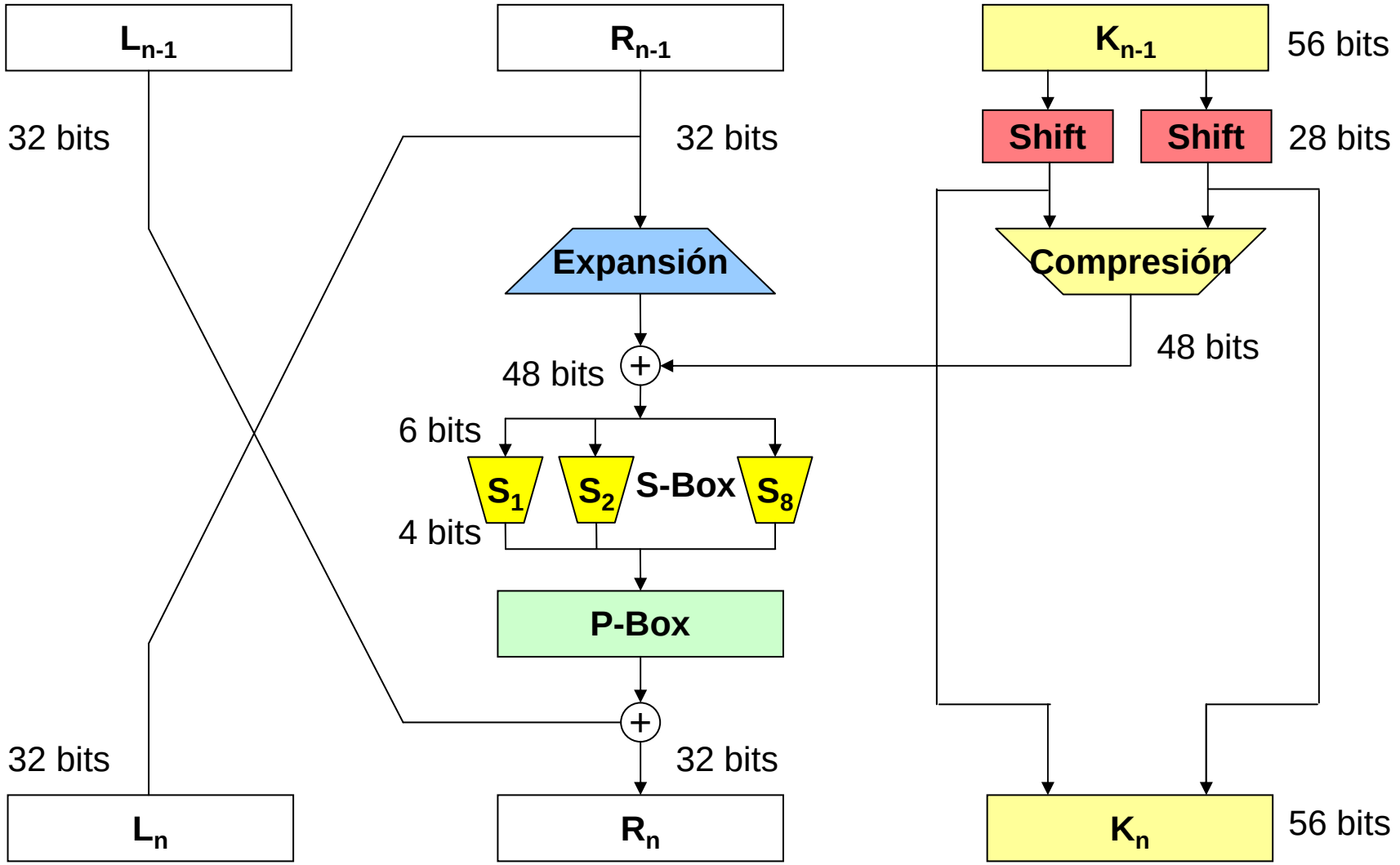
Función *F*

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 trasformación 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



Propiedades no deseadas

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.

Ataques

Fuerza bruta

- Consiste en probar una por una cada posible clave.

Criptoanálisis 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.

Criptoanálisis linear (1993)

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

Modos de operación

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

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.

$c = DES_k(DES_{k'}^{-1}(DES_k(m)))$

Encrypt-Encrypt-Encrypt (TripleDES)

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

$c = DES_k(DES_{k'}(DES_{k''}(m)))$

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.

Criptografía simétrica

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

Criptografía asimétrica

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.

Criptografía asimétrica

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*.

Condiciones

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.

Diffie-Hellman

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

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

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

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

Alice calcula $\boldsymbol{k = (g^b \bmod p)^a \bmod p}$
Bob calcula $\boldsymbol{k' = (g^a \bmod p)^b \bmod p}$

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

Funcionamiento

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

Alice elige:

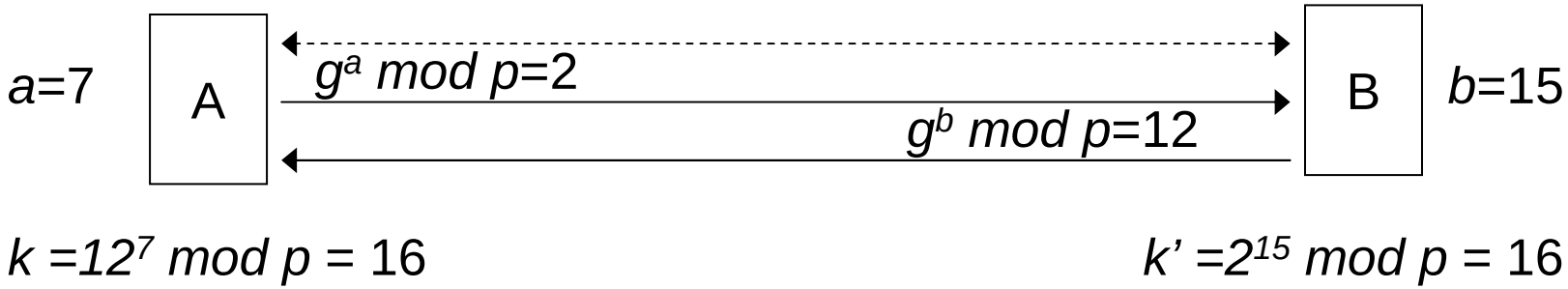
a = 7 y envia $g^a \bmod p$, o sea $3^7 \bmod 23 = 2$

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



Características

El algoritmo Diffie-Hellman se basa en operaciones matemáticas de exponenciación y en el problema de obtener el logaritmo discreto.

- 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

RSA

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)

Fundamentos

- 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

Funcionamiento

Consiste en elegir dos números primos grandes *p, q* tal que *p≠q.*

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

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

- Elegir $e < n$ talque 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***

Funcionamiento

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:

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

Confidencialidad

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)**

Confidencialidad

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

Confidencialidad

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 envio, ya que solo ella conoce su clave privada.

Integridad / Autenticación

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)**

Integridad / Autenticación

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 envio.

Usa su clave privada, $\boldsymbol{d = 53}$, y calcula $\boldsymbol{c = m^d \bmod n}$, es decir $\boldsymbol{c = m^{53} \bmod 77}$ 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

Integridad / Autenticación

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

Ataques

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.

Performance

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Equipo de prueba: 2.16GHz Intel Core 2

	16 bytes	**64 bytes**	**256 bytes**	**1024 bytes**	**8192 bytes**
des cbc	48478.10k	49616.16k	49765.21k	50106.71k	50034.01k
des ede3	18387.39	18631.02k	18699.26k	18738.18k	18718.72k
blowfish cbc	79577.50k	83067.03k	84676.78k	84850.01k	85063.00k
aes-128 cbc	58751.94k	94443.86k	111424.09k	116704.26k	117997.57k
aes-192 cbc	53451.79k	82076.22k	94609.83k	98496.85k	99150.51k
aes-256 cbc	49225.21k	72779.84k	82266.88k	85054.81k	85762.05k

(1000s de bytes por segundo)

	sign	**verify**	**sign/s**	**verify/s**
rsa 512 bits	0.000522s	0.000042s	1915.8	23969.9
rsa 1024 bits	0.002321s	0.000109s	430.8	9191.1
rsa 2048 bits	0.012883s	0.000329s	77.6	3039.6
rsa 4096 bits	0.079055s	0.001074s	12.6	931.3

Checksums

Usados para detección de errores:

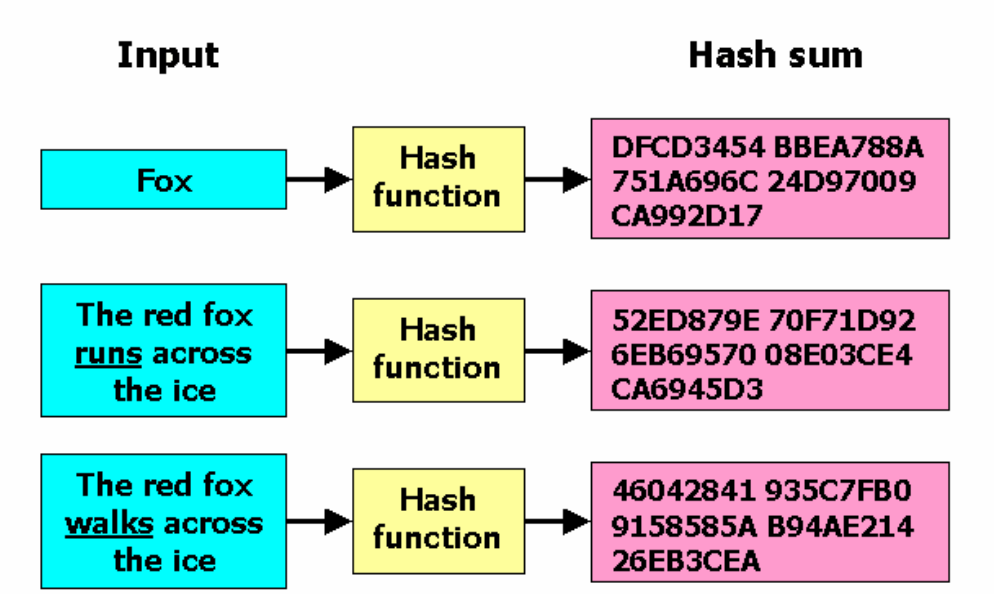
- Bit de Paridad: Digito binario que indica si la cantidad de bits en 1 en un conjunto de bits precedente es par o impar. Ejemplo: DES usa 8 bytes para la clave, pero el 8 bit de cada byte se usa para paridad.

- 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.



Funciones de hashing criptográficas

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 $h(x) = h(x')$

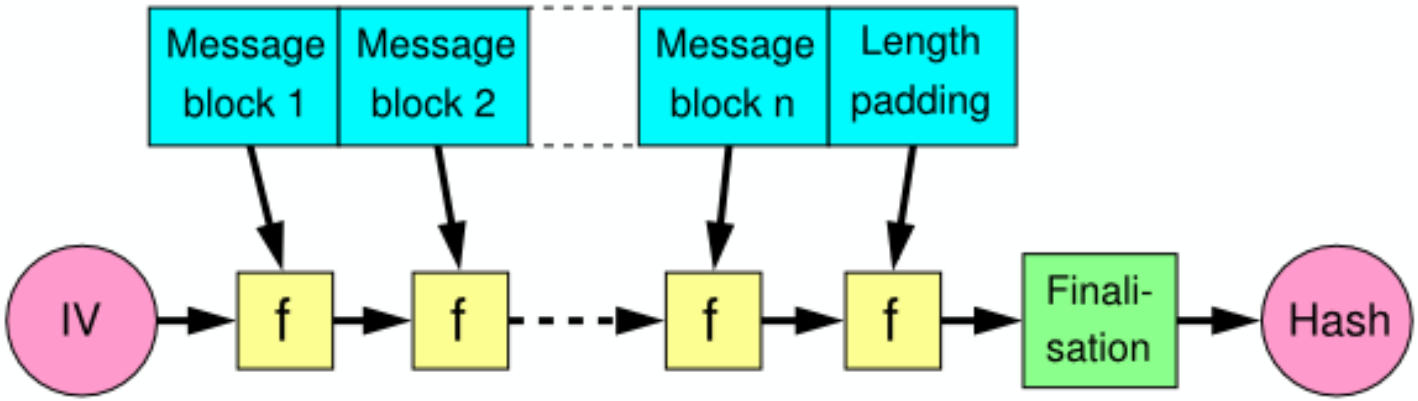
Si cumple 1. y 2. se denomina *función de hashing de una vía* o *función débil de una vía* (OWHF).

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.



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

Funciones de hashing criptográficas

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

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

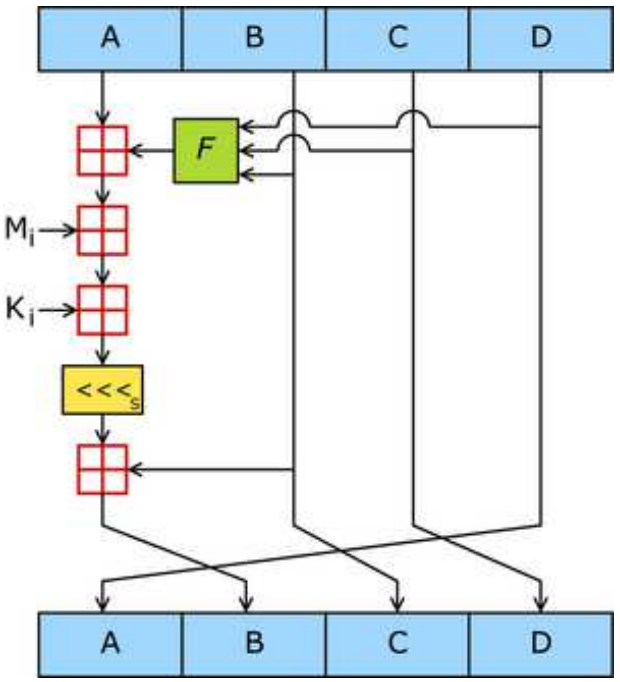
Funciones de hashing criptográficas

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

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.



$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

Confianza en funciones de hash

- En la actualidad, no todas las funciones de hash se consideran segura.
- Como se verá en los ejemplos, no se debe usar MD5.
- SHA-1 también presenta algunos ataques teóricos, por lo que es mejor utilizar SHA-2 (sha-256,sha-384,sha-512).
- Competencia SHA-3. Se eligió la función Keccak como nuevo standard (http://keccak.noekeon.org/). Luego de algunas discusiones, fue aprobado como standard en agosto de 2015 (FIPS 202)

Usos y mal usos de funciones de Hash

- 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).

Performance y soporte

Equipo de prueba: 2.16GHz Intel Core 2.

Software: openssl. Soporte para sha-256 y sha-512 agregado en Julio de 2005. Aun no posee soporte para sha-3

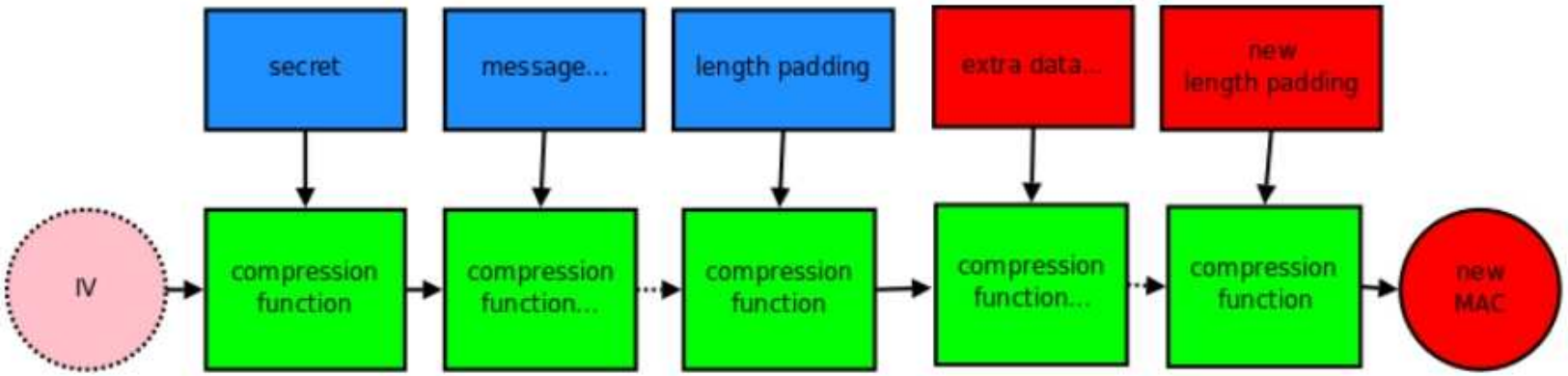
type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
md5	16807.01k	58256.45k	160439.13k	287183.53k	375220.91k
sha1	16774.59k	55500.39k	142628.69k	233247.74k	288382.98k
rmd160	13854.71k	40271.23k	87613.95k	124333.06k	141781.67k
sha256	9359.24k	22510.83k	40963.75k	51710.29k	56014.17k
sha512	7026.78k	28121.32k	54330.79k	86190.76k	104270.51k

(1000s de bytes por segundo)

Ataques a funciones de hash

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

- Length Extension Attack:



- Funciones de hash que utilizan la construcción Merkle–Damgård son vulnerables.

- Solución: double-hashing h(h(m))

HMAC

- 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 (longitud mayor a la del hash), obtenemos un MAC, que podemos utilizar para detectar manipulación de contenido.
- Es un mecanismo muy rápido.

HMAC

$HMAC_k(m) = h((k \oplus opad) \| h((k \oplus ipad) \| m))$

donde opad = 0x5c5c5c…5c, ipad = 0x363636…36

Ejemplos: HMAC-MD5, HMAC-SHA1

Utilizados en SSL, IPSEC para detectar cambios.

Más información: RFC 2104

Usos y mal usos de funciones HMAC

- Garantizar de 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

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

- ¿ Por qué usar
  $HMAC_k(m) = h((k \oplus opad) \| h((k \oplus ipad) \| m))$?

- ¿ Si uso $HMAC_k(m) = h(k \| m)$?
  - ¡Length extension attack! (Ej. API de Flickr, http://netifera.com/research/flickr_api_signature_forgery.pdf)

- ¿ Si uso $HMAC_k(m) = h(m \| k)$?
  - Solucioné Length Extension Attack
  - ¿Qué pasa si puedo encontrar una colisión?
    - Sea m' / h(m) = h(m'), entonces h(m||k)=h(m'||k)
    - En MD5 encontrar una colisión de prefijo-elegido toma menos de 1 minuto.