



# **8636 Criptografía y Seg Informática**

**Elementos de Criptografía  
Hash, MAC**



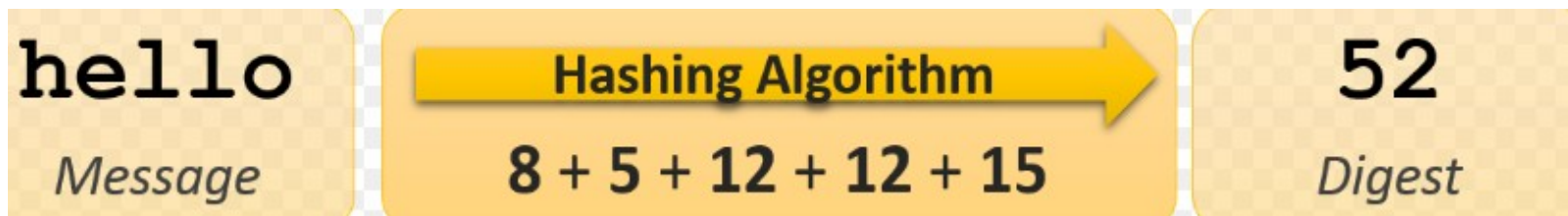
# Funciones de Hash

Autenticación de Mensajes



# Que es una función de HASH?

- Las funciones hashing (o hash) juegan un rol fundamental en la Criptografía pues sirven esencialmente para verificar la integridad de los mensajes.
- Una función de hashing  **$h(m)$**  transforma cadenas de bits ( $m$ ) de longitudes arbitrarias pero finitas, en otras cadenas de longitud fija de  $n$ -bits (**Generalmente  $m \gg n$** ).



# Hash: PROPIEDADES BASICAS

- **Compresión:** A partir de un mensaje de cualquier longitud, el resumen  $h(M)$  debe tener una longitud fija. Lo normal es que la longitud de  $h(M)$  sea menor que el mensaje  $M$ .

```
(kali@kali)-[~/Desktop/hash]
$ ls -lh -ltr
total 216K
-rw-r--r-- 1 kali kali 214K Oct  5 16:05 Seguridad-informática.txt

(kali@kali)-[~/Desktop/hash]
$ openssl sha256 Seguridad-informática.txt
SHA256(Seguridad-informática.txt)= 35e63a5b61a7dbd9c091711e41d25dba5dd38243735d94fa6edde2544a4bf745
```

- **Facilidad de cálculo:** debe ser fácil calcular  $h(M)$  a partir de un mensaje  $M$ .



# Propiedades de las funciones hash

- **Unidireccionalidad:** Conocido un resumen  $h(M)$ , debe ser computacionalmente imposible encontrar  $M$  a partir de dicho resumen.
- **Difusión:** El resumen  $h(M)$  debe ser una función compleja de todos los bits del mensaje  $M$ : si se modifica un solo bit del mensaje  $M$ , el hash  $h(M)$  debería cambiar la mitad de sus bits aproximadamente.

```
(kali㉿kali)-[~/Desktop/hash]
$ echo "este mensaje es secreto" > secret.txt

(kali㉿kali)-[~/Desktop/hash]
$ openssl sha256 secret.txt
SHA256(secret.txt)= 00cc5da103ce54cee9fa1f578a33f62ebc30a4c1772b577afdfceeeef0f9796f0

(kali㉿kali)-[~/Desktop/hash]
$ echo "este mensaje es secreto0" > secret.txt

(kali㉿kali)-[~/Desktop/hash]
$ openssl sha256 secret.txt
SHA256(secret.txt)= 9cb1e717b86aaeaab917a7c18f465259744a057446b3b96d5ce991bcb547c698
```



# Propiedades de las funciones hash

---

## **RESISTENCIA A LA PREIMAGEN:**

Es computacionalmente imposible encontrar un  $x$  (desconocido) tal que  $h(x)=y$  (conocido)

## **RESISTENCIA A LA 2° PREIMAGEN:**

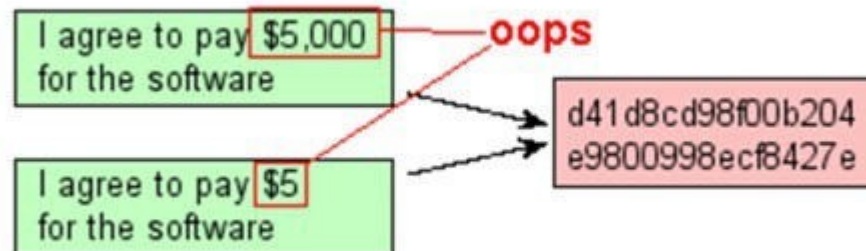
Dados  $x$  y su correspondiente  $y=h(x)$ , Es computacionalmente imposible encontrar otro  $x'$  distinto a  $x$  tal que  $y'=h(x')=y$

## **Resistencia a las colisiones:**

Es computacionalmente imposible encontrar 2 entradas  $(x,x')$  que tengan la misma salida  $h(x)$

# Resistencia

- **RESISTENCIA A LA PREIMAGEN:**
- **RESISTENCIA A LA 2° PREIMAGEN**
- **Resistencia a las colisiones:**

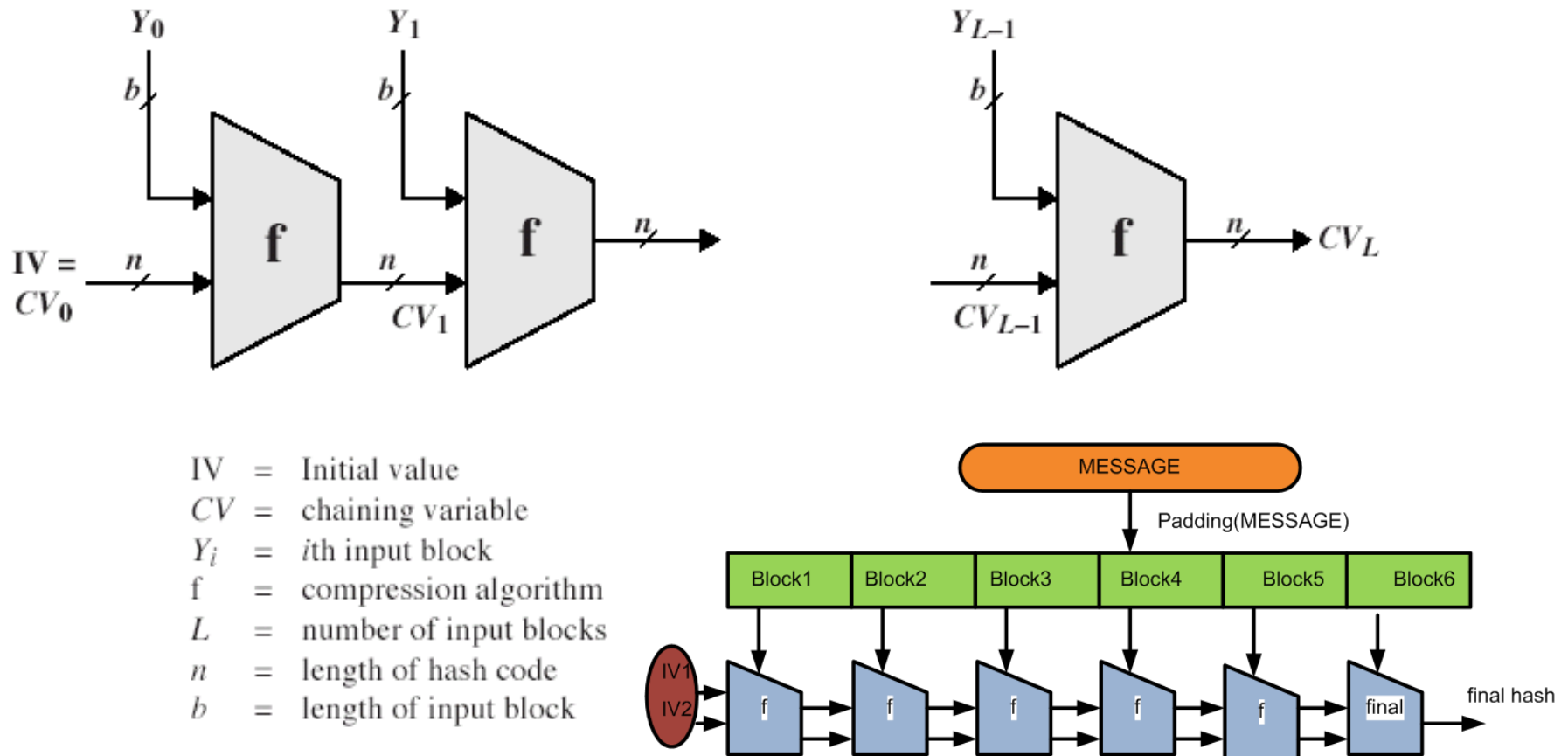




- **NO CORRELACIÓN:** Los bits de entrada y los bits de salida no deben estar correlacionados. Vinculado con esto, es deseable un efecto de **AVALANCHA** en la cual cada bit de entrada afecta a cada bit de salida.



# Estructura Básica de un Hash

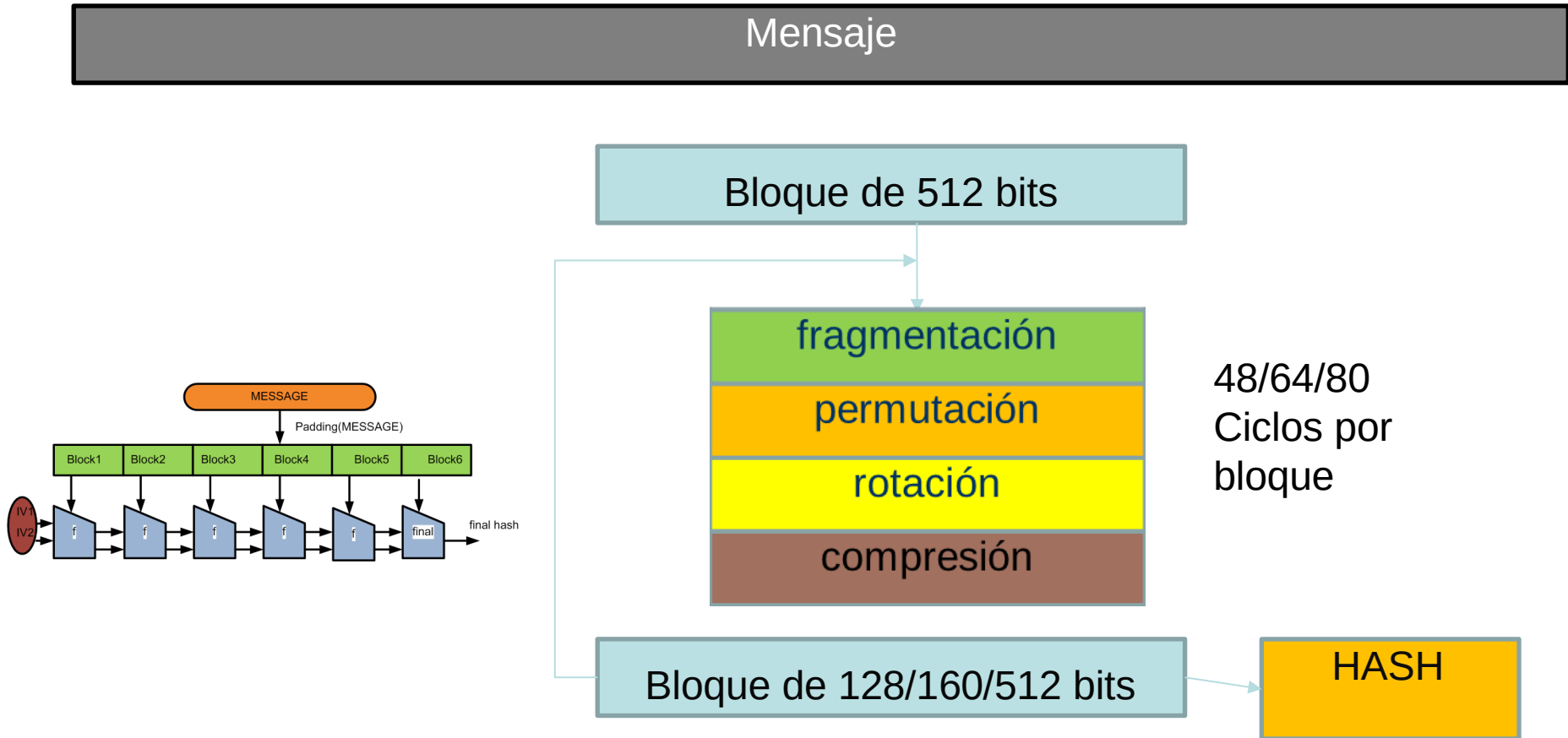


**Figure 11.10 General Structure of Secure Hash Code**



# MODELO BASICO DE FUNCIONES HASH

Familia MD4-MD5-SHA-SHA1-RIPEMD128-RIPEMD160-SHA512



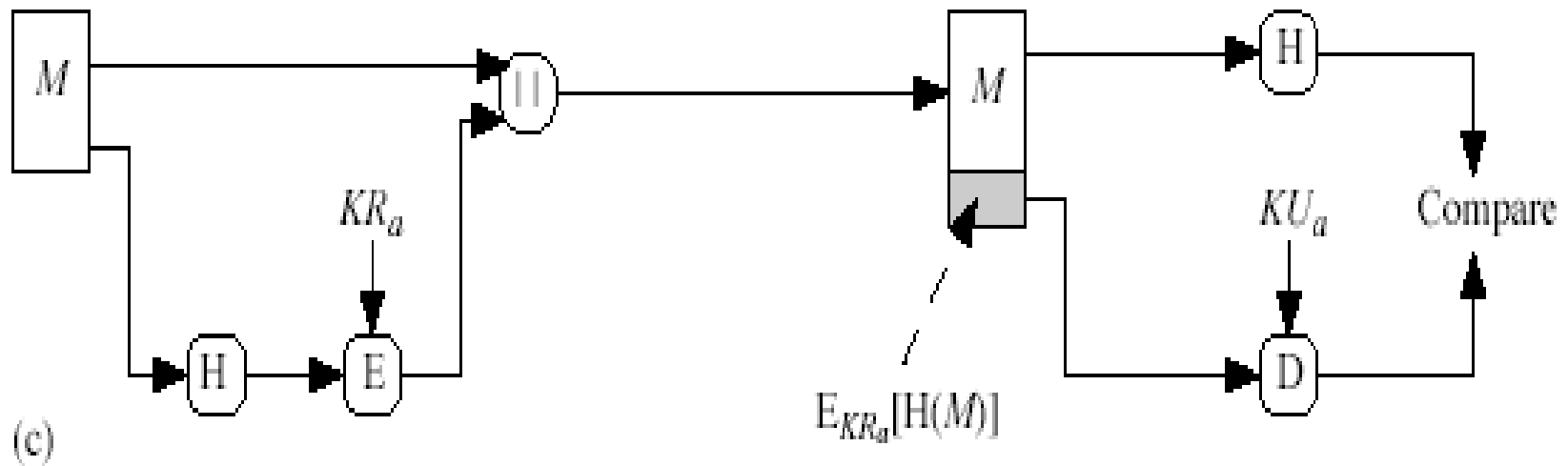


# Firma Digital y Funciones de Hash

---

- Una de las aplicaciones más interesantes de la actual criptografía es la posibilidad real de añadir a un mensaje su firma digital: autenticación.
  - Dado que los sistemas de clave pública son muy lentos, en vez de firmar digitalmente el mensaje completo.
  - Se incluirá como firma digital una operación de cifra con la clave privada del emisor sobre un resumen o HASH de dicho mensaje
-

# Firma Digital y Funciones de Hash



Mensaje =  $M$  ▲

Función Resumen =  $h(M)$

Firma (rúbrica):  $r = E_d\{h(M)\}$


Calcula:  $E_e(r) = h(M)$

Compara: ¿ $h(M') = h(M)$ ?

## Demostración práctica de colisión en SHA-1

25 febrero, 2017 Por Hispasec — Deja un comentario

Un grupo de investigadores ha **anunciado** que han logrado desarrollar una técnica que **hace práctico para elaborar dos archivos con la misma huella digital SHA-1**.

Por supuesto, como es habitual en los últimos años un descubrimiento de estas características necesitaba un nombre atractivo, una página web y un logo chulo: SHattered  <http://shattered.io/>

Hay que recordar que **una función hash no cifra, sino que crea un resumen** o «*firma*» de un conjunto de datos, que es pasado como parámetro a esta función. Así, nos es útil para verificar la integridad, por ejemplo, de un archivo. Tan solo tenemos que aplicar dicha función sobre el archivo recibido y verificar que el hash obtenido es el mismo que el anunciado por el emisor.

<http://shattered.io/static/infographic.pdf>

<https://unaaldia.hispasec.com/2017/02/demostracion-practica-de-colision-en-sha-1.html>



# Ejercicio HASH

---

Usando <http://onlinemd5.com/u> otra herramienta

- Hacer el md5, sha1 y sha256 de los recursos que están en el campus
  - Pdfmd5
  - Imagenesmd5
  - Pdfsha1
- Editar la imagen de James Brown y verificar que se le cambia el HASH.



## Ejercicio HASH 2

---

Leer las consideraciones del documento `incibe_toma_evidencias_analisis_forense.pdf`

- ¿Cómo se usa un HASH en una pericia?
- ¿Qué HASH usaría para una pericia informática?  
(md5, sha1, sha256, etc)



# Funciones de Hash

SHA3





# antecedentes

---

- Luego de la publicación de vulnerabilidades en SHA el NIST - National Institute of Standards and Technology organizo grupos de trabajo para evaluar el estado de las funciones de hash aprobadas.
- Se aconsejo una transición de SHA-1 a la familia SHA-2 de funciones de hash.
- También se decidió organizar un concurso similar a AES para contar con una función de hash.



# Concurso sha3

---

- publicado el 2 de noviembre de 2007, y se aceptaron presentaciones hasta el 31 de octubre del 2008.
- El 9 de diciembre de 2008 se anuncio a 64 candidatos para la primer ronda y se aceptaron 51
- En 2010 quedaron 14 candidatos aceptados en segunda ronda
- en 2011 5 candidatos en tercera y última ronda como finalistas.
- En 2012 se selecciona KECCAK como SHA3



# Requerimientos

---

- Debe tener tamaños compatibles con la familia SHA-2, es decir, debe tener salidas de tamaño 224, 256, 384 y 512 bits. Esto es para facilitar la migración de implementaciones.
- Debe proveer el nivel de seguridad esperado para las siguientes aplicaciones:
  - firmas digitales,
  - derivación de claves
  - HMACs
  - generadores de secuencias pseudo aleatorias

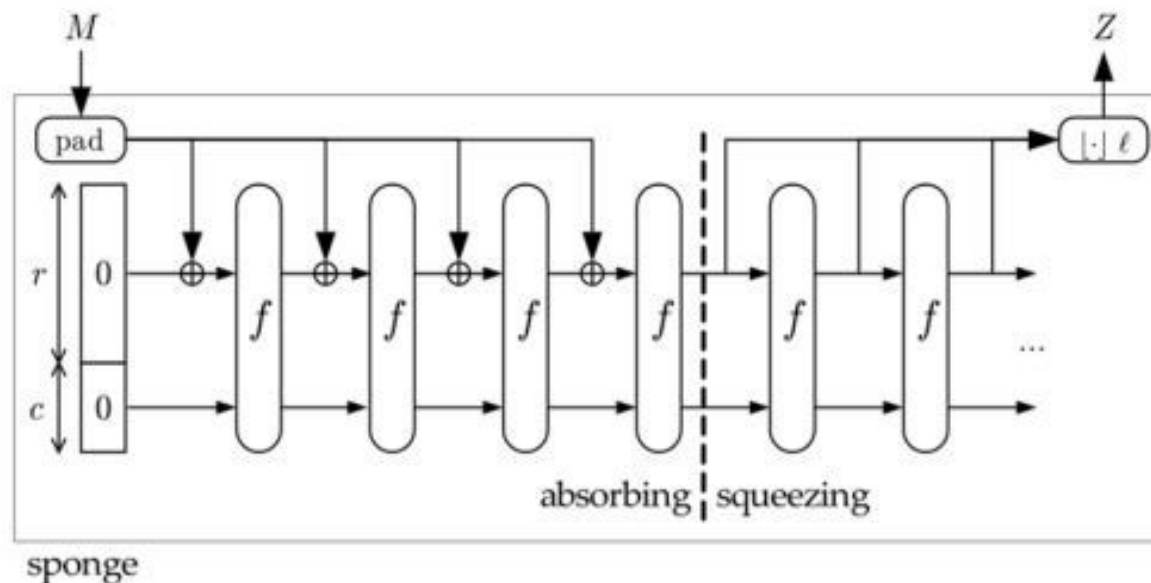


- Resistencia a colisiones con orden de  $n/2$  bits
- Resistencia a preimágenes
- Resistencia a segundas preimágenes
- Eficiencia superior a sha2 en plataforma definida como de referencia y en plataformas de 8 bits
- Capacidad de procesar los mensajes con una sola pasada

## The sponge construction

The sponge construction is a simple iterated construction for building a function  $F$  with variable-length input and arbitrary output length based on a fixed-length permutation (or transformation)  $f$  operating on a fixed number  $b$  of bits. Here  $b$  is called the **width**.

The sponge construction operates on a state of  $b=r+c$  bits. The value  $r$  is called the **bitrate** and the value  $c$  the **capacity**.





# Documentación

---

- Detalles del concurso en
  - [http://ehash.iaik.tugraz.at/wiki/The SHA-3 Zoo](http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo)
- Proyecto SHA3
  - <https://csrc.nist.gov/projects/hash-functions/sha-3-project>
- Estandarización de NIST FIPS 180-4
  - <https://csrc.nist.gov/publications/detail/fips/180/4/final>



# Lista de algoritmos de Hash

hash	year	coll. res.	size (bits)	design	broken?
MD4	1990	64	128	32-bit ARX DM	1995
SHA-0 (SHA)	1993	80	160	32-bit ARX DM	1998
MD5	1993	64	128	32-bit ARX DM	2004
SHA-1	1995	80	160	32-bit ARX DM	2005
SHA-256 (SHA-2)	2002	128	256	32-bit ARX DM	
SHA-384 (SHA-2)	2002	192	384	64-bit ARX DM	
SHA-512 (SHA-2)	2002	256	512	64-bit ARX DM	
SHA-224 (SHA-2)	2008	112	224	32-bit ARX DM	
SHA-512/224	2012	112	224	64-bit ARX DM	
SHA-512/256	2012	128	256	64-bit ARX DM	
SHA3-224	2013	112	224	64-bit Keccak sponge	
SHA3-256	2013	128	256	64-bit Keccak sponge	
SHA3-384	2013	192	384	64-bit Keccak sponge	
SHA3-512	2013	256	512	64-bit Keccak sponge	
SHAKE128	2013	$\leq 128$	any	64-bit Keccak sponge	
SHAKE256	2013	$\leq 256$	any	64-bit Keccak sponge	



# 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  $H(x)$





# MAC

Autenticación de Mensajes



# Autenticación de Mensajes

---

Se necesita:

- Proteger la integridad del mensaje
- Validar la identidad del origen
- No-repudio del origen (resolución de disputas)

—

Para esto se usan tres alternativas o funciones:

- Encripción del mensaje
  - Criptografía publica
  - message authentication code (MAC)
-



# Función de encriptación: simétrica

---

La simple encriptación permite autenticación

Si se usa un sistema simétrico:

- El receptor puede suponer que el origen es válido
- Dado que solo el origen y el receptor conocen la clave
- El contenido no puede ser alterado
  - Siempre y cuando el mensaje tenga alguna estructura, redundancia o un checksum.



# Función de encriptación: asimétrica

---

Si se usa clave publica:

- Con la privada autentico el origen
- Y con la publica obtengo el secreto
- Permite detectar intentos de modificación o alteraciones
- Al costo de un doble cifrado (Autenticación + secreto)



# Message Authentication Code (MAC)

---

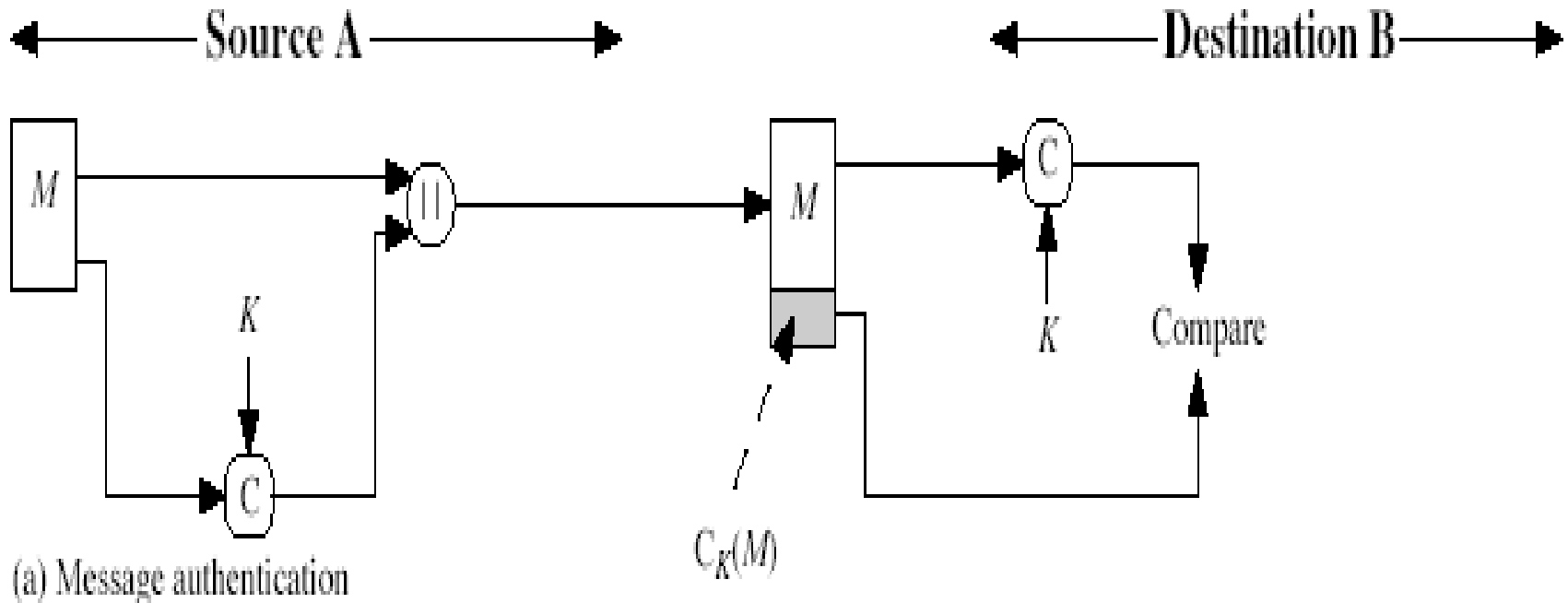
El algoritmo de MAC genera un bloque de bits de tamaño fijo:

- La salida depende del mensaje y de una clave
- Parecido a una encriptación, pero no es reversible

Se concatena al mensaje

El receptor realiza el mismo calculo y verifica el  
MAC Permite asegurar que el mensaje no fue  
alterado

# Message Authentication Code



M	Mensaje
C	Función MAC
K	clave
$\parallel$	Concatenación



# Message Authentication Codes

---

Se puede usar en conjunto con el cifrado

- Generalmente se usan claves distintas
- El MAC se puede incorporar antes o después
- Generalmente el MAC se incorpora antes de cifrar

Porque MAC?

- A veces se necesita solamente autenticar
- Otras se necesita que el MAC persista para por ejemplo ser archivado

MAC no es Firma Digital

---



# Propiedades de las MAC

---

Un MAC es un resumen criptográfico

$$\text{MAC} = C_k(M)$$

- Convierte un mensaje de tamaño arbitrario en un autenticador de tamaño fijo de pocos bytes.

Es una función muchos-uno

- Potencialmente hay muchos mensajes con la misma MAC
  - Encontrarlos debe ser muy difícil
-





# MAC: Requerimientos

---

El MAC debe satisfacer::

1. Conocido un mensaje y su MAC, debe ser computacionalmente imposible encontrar otro mensaje con el mismo MAC
2. Las MAC deben estar distribuidas uniformemente.
3. La MAC debe depender de todos los bits del mensaje.

# Uso de cifradores simétricos para construir un MAC



Se puede usar cualquier algoritmo con un modo de operación de los encadenados y quedarse con el ultimo bloque

**Data Authentication Algorithm (DAA)** es un MAC basado en AES-CBC

- IV=0 y zero-pad del bloque final

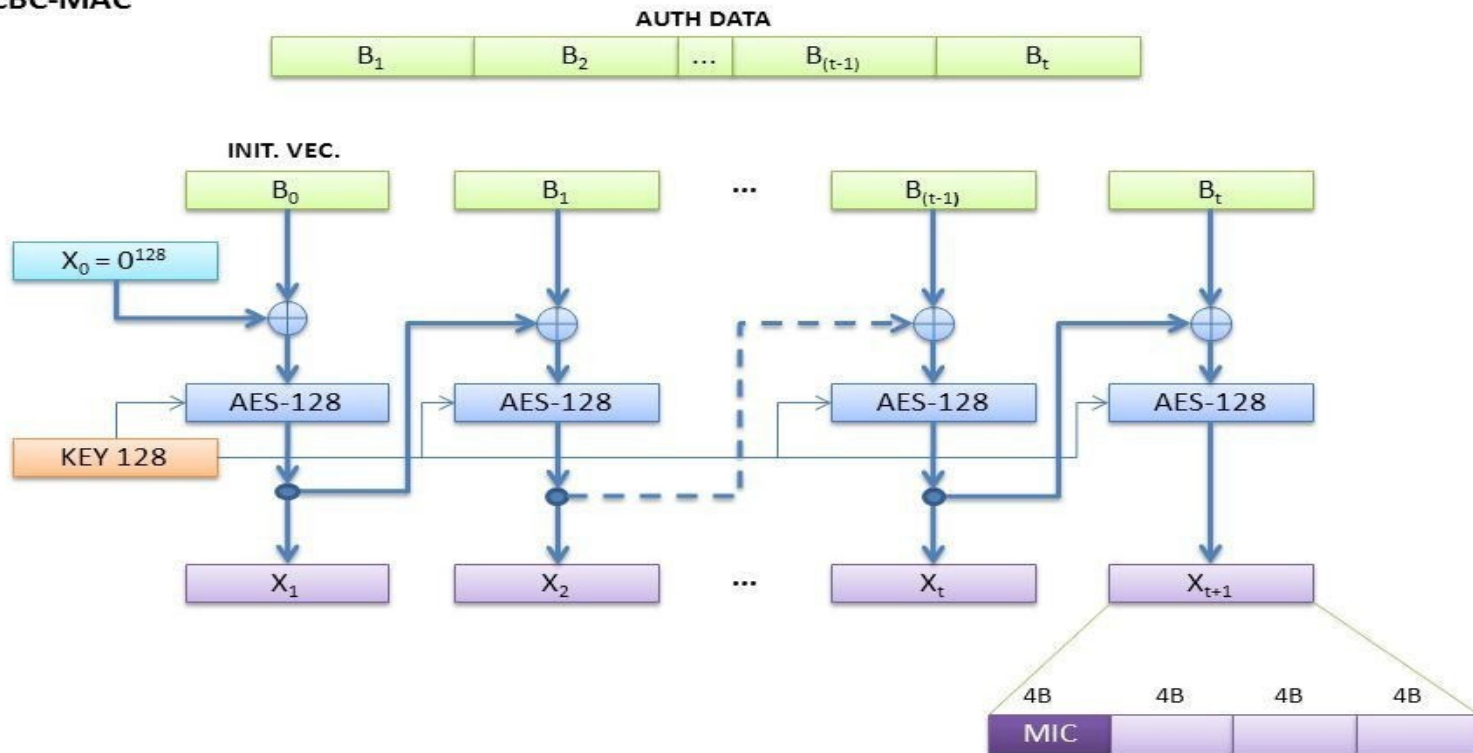
Actualmente el algoritmo MAC no es considerado seguro (56 bits)

# CMAC: Cipher-based Message Authentication Code

Basado en AES <https://tools.ietf.org/html/rfc4493>

## AUTHENTICATION (TX/DATA)

AES-CBC-MAC



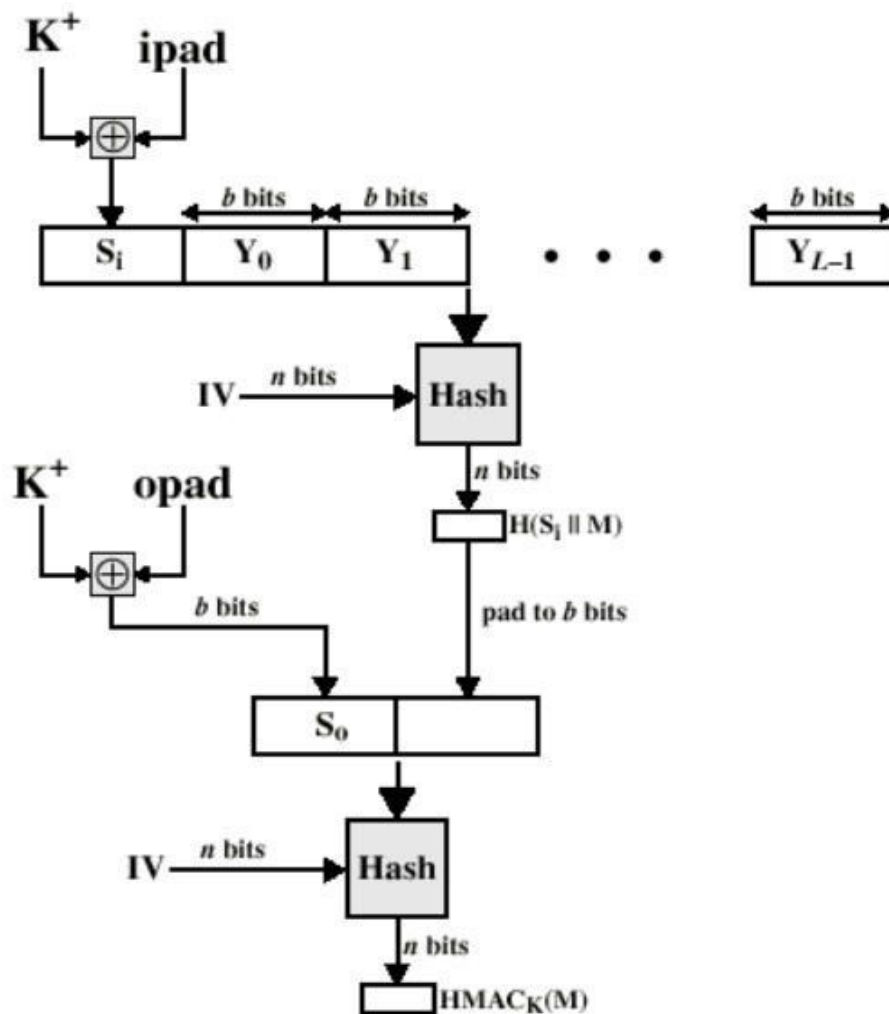


Construir el MAC usando como modulo un hash como el sha256

- Los hash suelen ser mas rapido que los algoritmos simetricos
- Bibliotecas disponibles

Ej :  
HMAC\_SHA256

<https://www.ietf.org/rfc/rfc2104.txt>



ipad = the byte 0x36 repeated B times  
 opad = the byte 0x5C repeated B times.  
 $HMAC = H((Key \oplus \text{ipad}) || H(Key \oplus \text{opad} || m))$

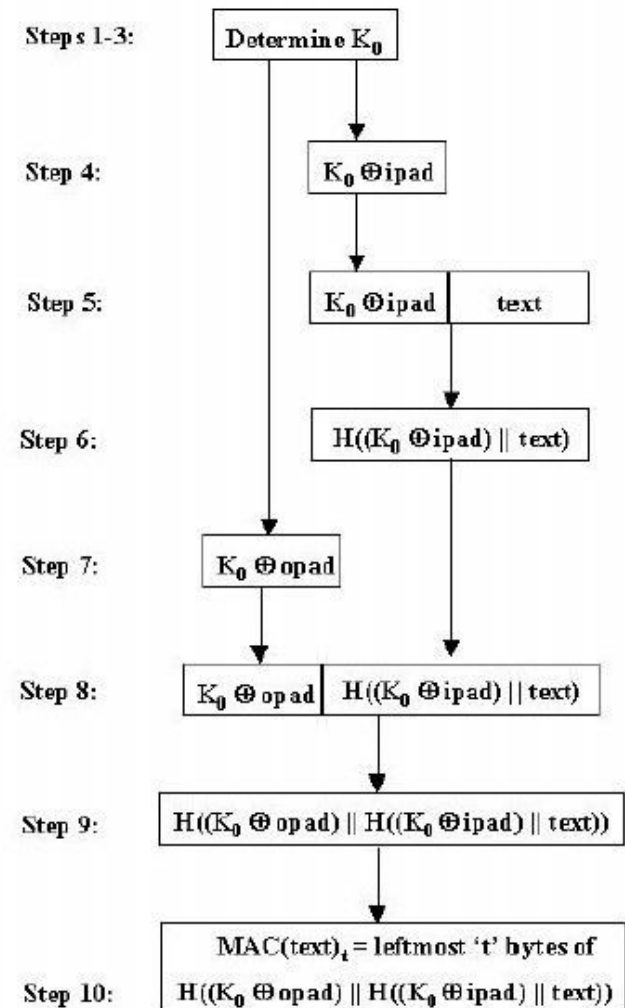
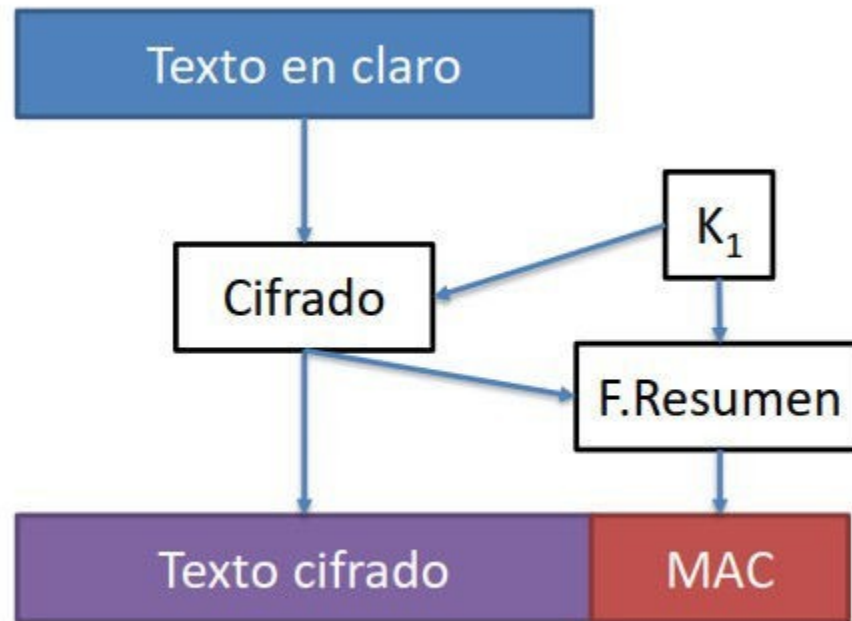


Figure 1: Illustration of the HMAC Construction

# Cifrado Autenticado. Cuando usar el MAC?



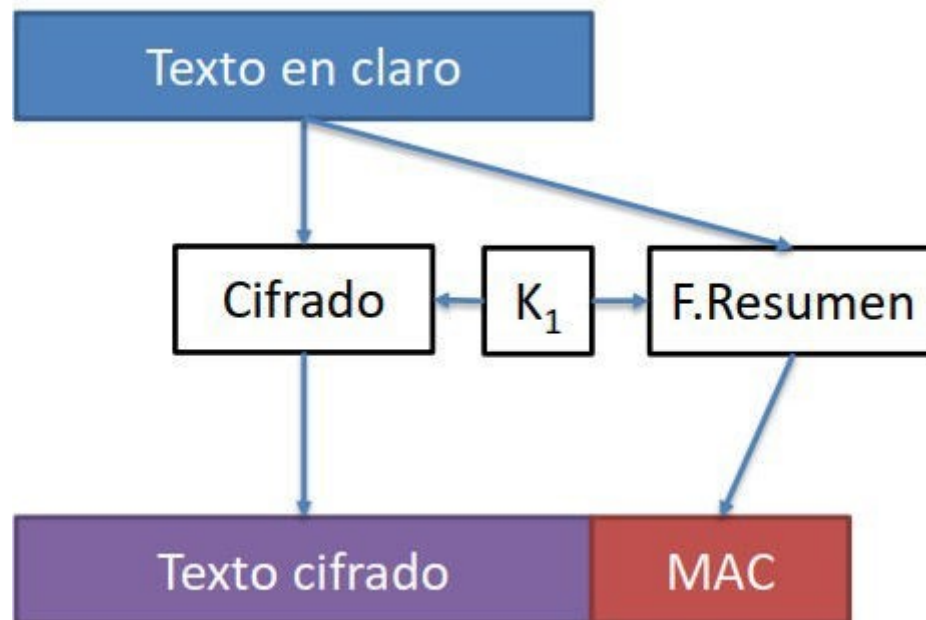
## *Encrypt-then-MAC*



# Cifrado Autenticado. Cuando usar el MAC?



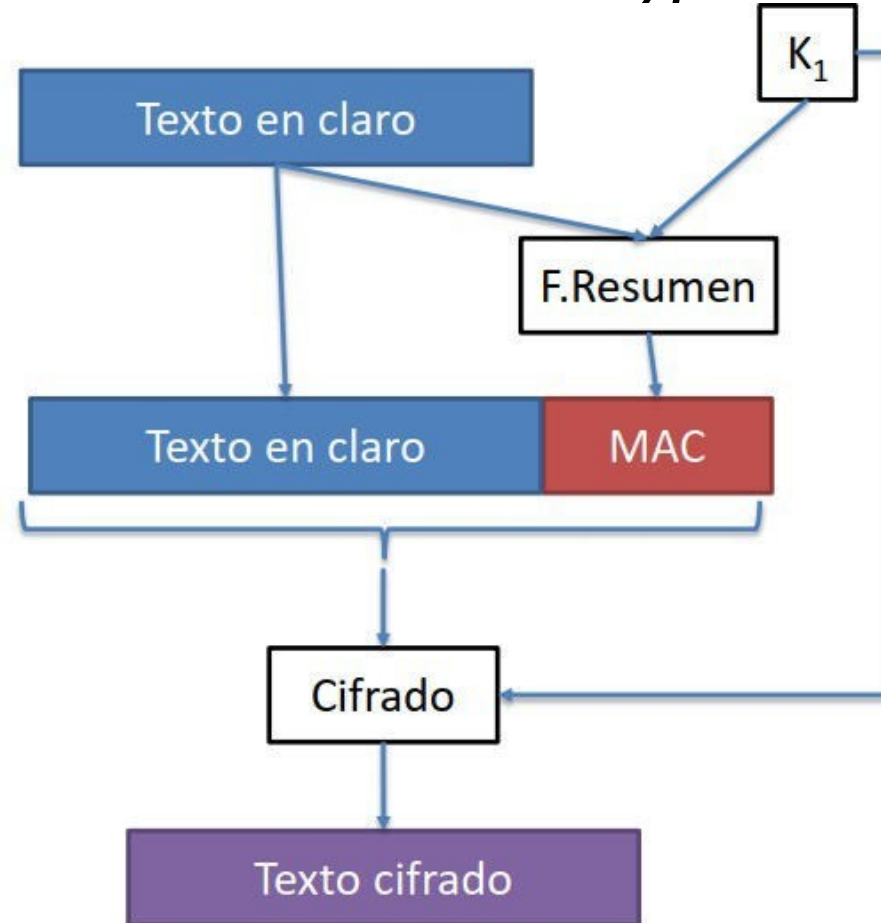
## *Encrypt-and-MAC*



# Cifrado Autenticado. Cuando usar el MAC?

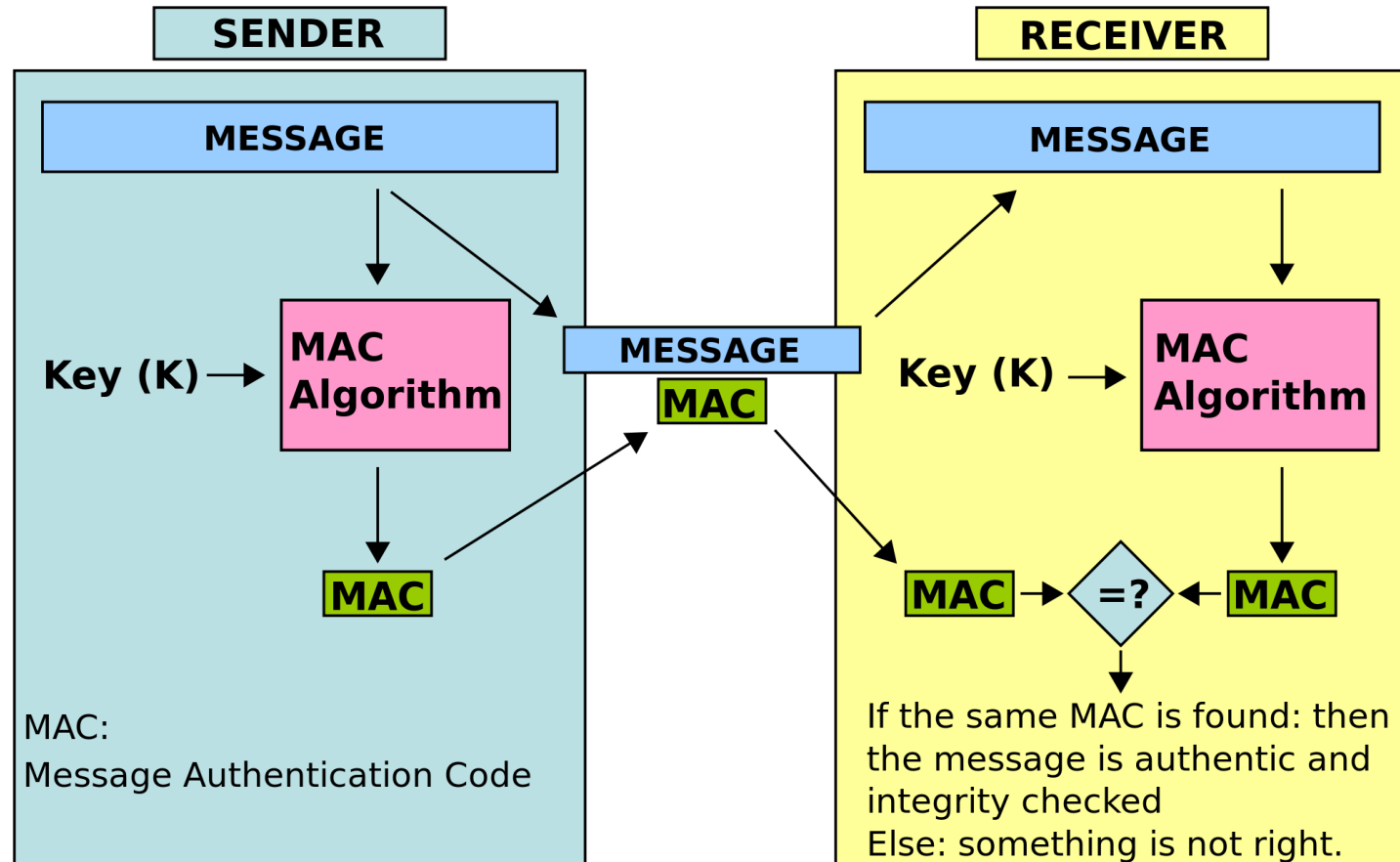


## *MAC-then-Encrypt*





# Uso típico del MAC





# Uso de los MACS

- En el protocolo AH y en la autenticación del protocolo ESP de ipsec
- En TLS para autenticar los mensajes
- Usar HMAC para firmar un JSON Web
- Tokens En Python

```
import hashlib
```

```
import hmac
```

```
message_mac = hmac.new("s3cr3t0", msg="Hola Mundo",  
digestmod=hashlib.sha256)
```

```
print message_mac.hexdigest()
```

```
36d3a0d55191b37d820435f406cd05266931eb8e970c3517b13b971a9a390d8  
e
```



# Ejercicio

# Validar el siguiente JWT que esta firmado con HMAC256

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWUiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkZhY3VsdGFklGRlIEluZ2VuaWVyaWEiLCJlZGFkljoMTUxliwiaWF0IjoxNTE2MjM5MDIyfQ.Zwl6favtwz9vohZQSRJquevW7hxA2i4X80chwDUao5E

- Utilizar <https://jwt.io>, la clave compartida del HMAC es fiuba
- Cambiarle la firma a fiuba2 ¿Qué paso?



# Ejercicio

---

## Generar el HMAC256 del siguiente string

*Nos, los representantes del pueblo de la Nación Argentina, reunidos en Congreso General Constituyente por voluntad y elección de las provincias que la componen, en cumplimiento de pactos preexistentes, con el objeto de constituir la unión nacional, afianzar la justicia, consolidar la paz interior, proveer a la defensa común, promover el bienestar general, y asegurar los beneficios de la libertad para nosotros, para nuestra posteridad y para todos los hombres del mundo que quieran habitar en el suelo argentino; invocando la protección de Dios, fuente de toda razón y justicia: ordenamos, decretamos y establecemos esta Constitución para la Nación Argentina.*

- Hacer el HMAC con clave compartida “argentina”
- Utilizando <https://codebeautify.org/hmac-generator>



# MACS en las comunicaciones

---

- IpSec, ssh y TLS generan MACS para cada paquete transmitido,
- No todos los protocolos lo hacen. Ej 4G encripta pero no autentica el mensaje.
- Replay Attacks: Los MACS no son inmunes a ataques de repetición de paquetes. Se puede capturar un paquete y enviarlo nuevamente. Los protocolos deberán incorporar de alguna forma un numero secuencial para evitar esto



PRNG

# Pseudorandom Number Generation (PRNG)

---



## ■ Numeros Pseudoaleatorios

- Basados en Hash
- Basados en MAC
- Basados en RSA

## TOUR OF ACCOUNTING

OVER HERE  
WE HAVE OUR  
RANDOM NUMBER  
GENERATOR.



www.dilbert.com scottadams@aol.com

NINE NINE  
NINE NINE  
NINE NINE



10/25/01 © 2001 United Feature Syndicate, Inc.

ARE  
YOU  
SURE  
THAT'S  
RANDOM?



THAT'S THE  
PROBLEM  
WITH RAN-  
DOMNESS:  
YOU CAN  
NEVER BE  
SURE.



# PRNG using a Hash Function

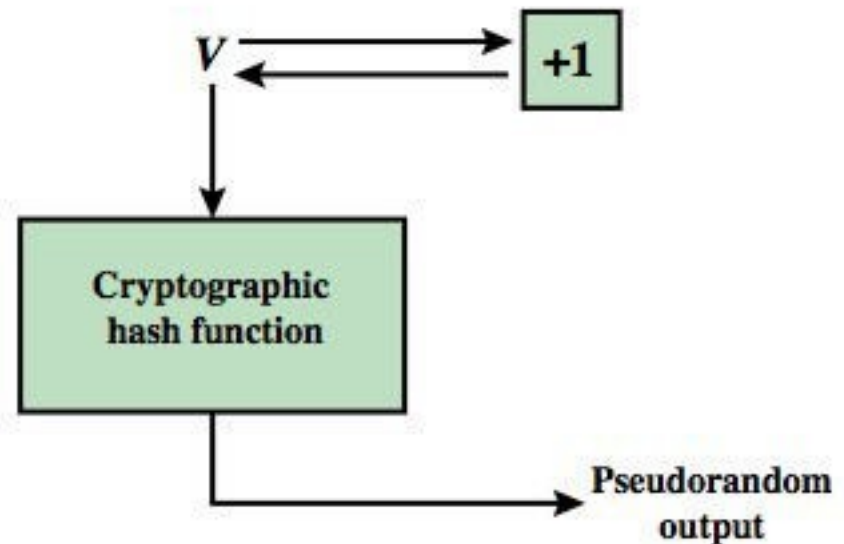
▲ hash PRNG from SP800-90 and ISO18031

☛ take seed  $V$

☛ repeatedly add 1

☛ hash  $V$

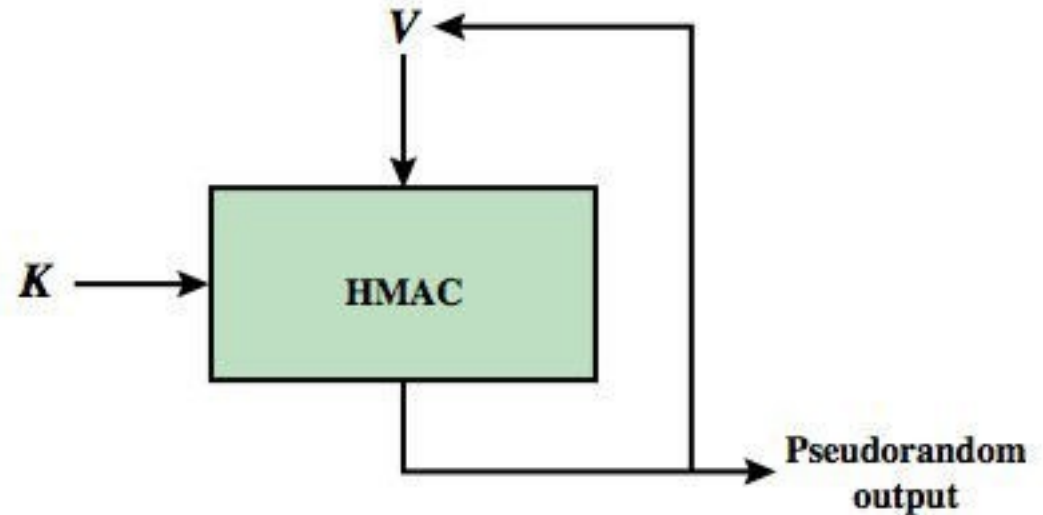
▲ secure if good hash used  
use  $n$ -bits of hash as random value



(a) PRNG using cryptographic hash function

# PRNG using a MAC

▲MAC  
SP800-90,  
802.11i,  
▲IEEE  
TLS



(b) PRNG using HMAC

# Micali-Schnorr PRNG usando RSA

## ANSI X9.82 - ISO 18031

