CRIPTOGRAFIA APLICADA

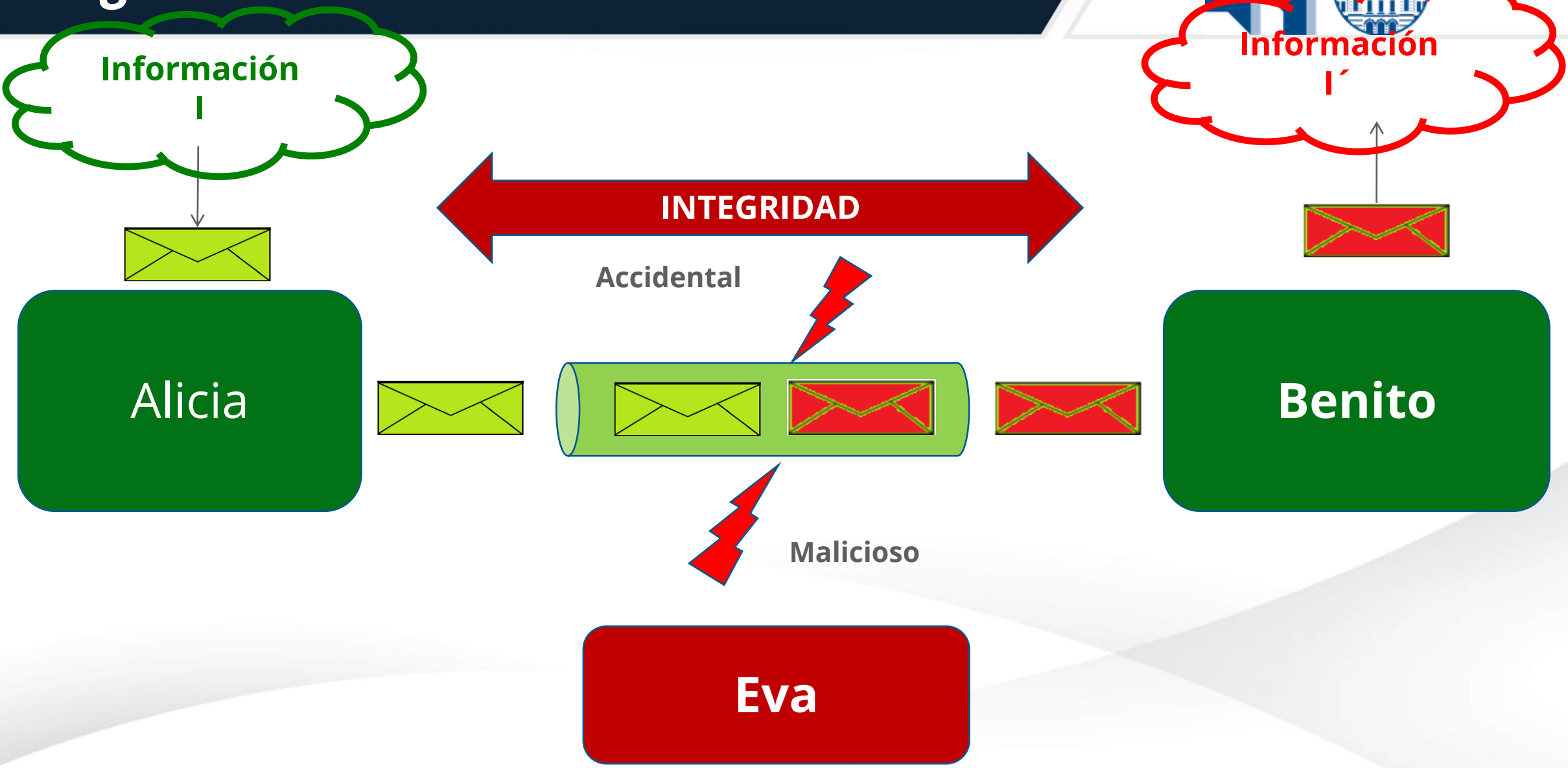
Ing Germán Bollmann

AGENDA

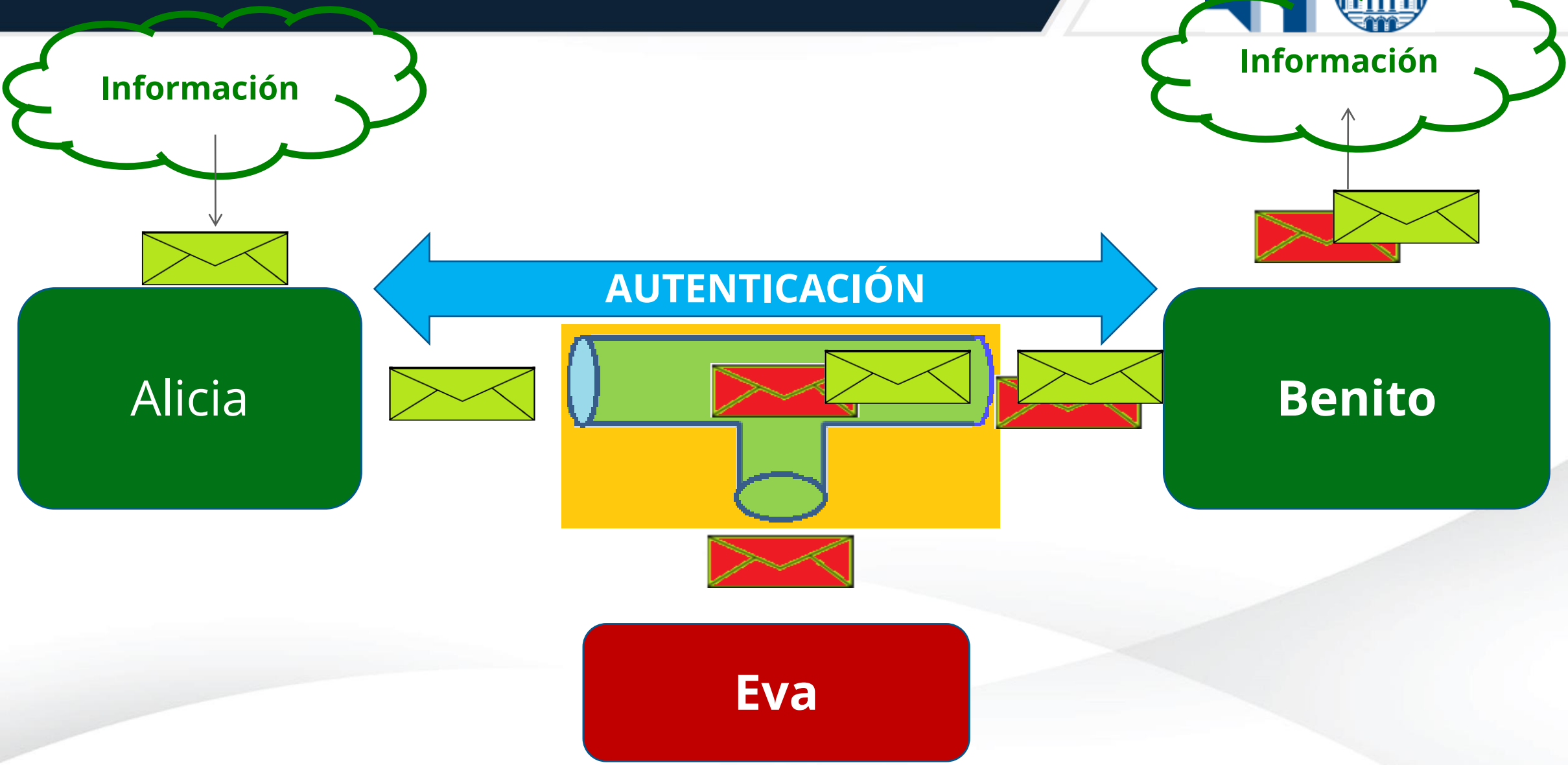
Funciones
Hash (Resumen)
MAC (Código Autenticador de Mensaje)
HMAC (Hash-Mac)
KDF (Funciones Derivadoras de Mensajes)

Funciones Hash

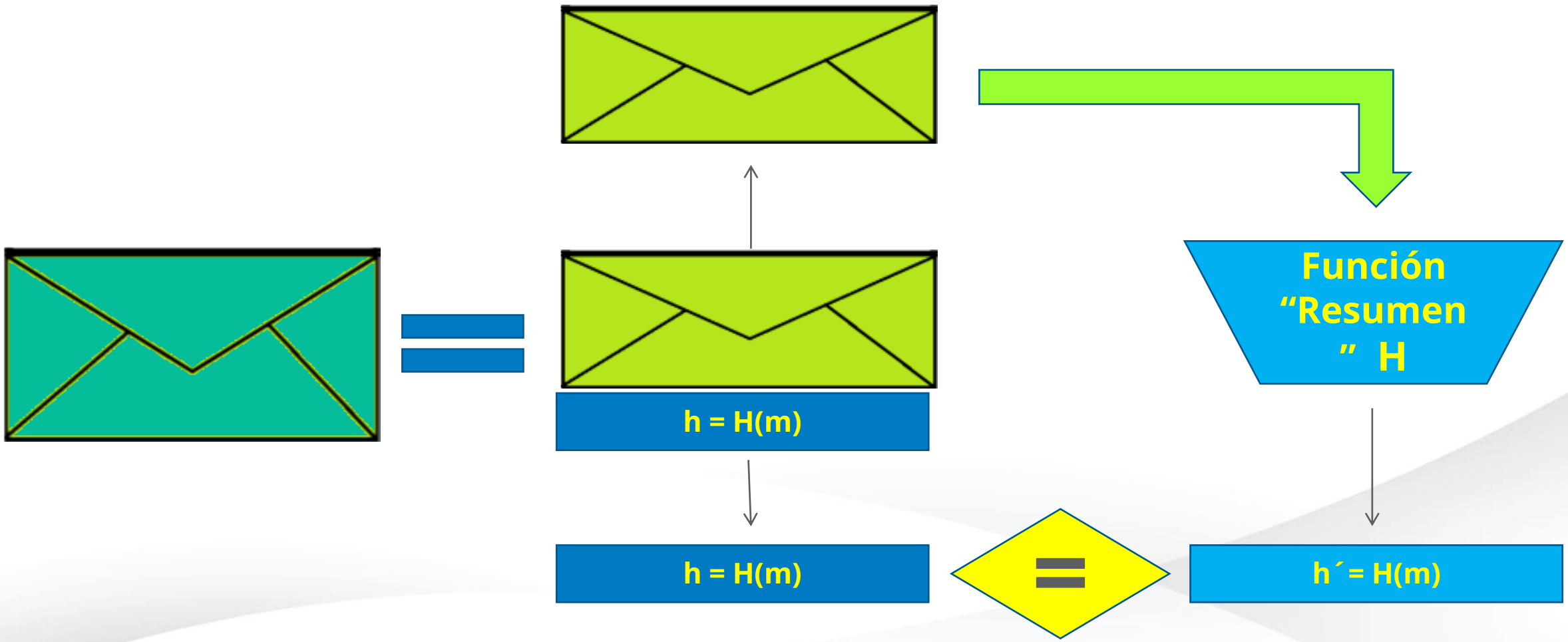
Integridad de la Información



Autenticación de la Información

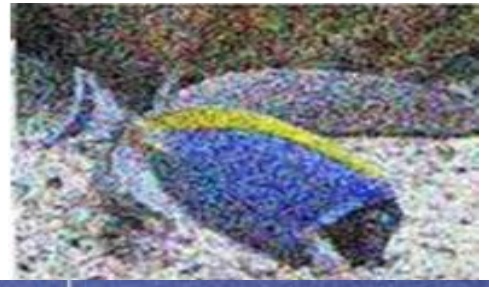


Proceso que se corre en el Receptor para verificar la Integridad



Integridad de la Información: Imágenes

El ruido no necesariamente está repartido por toda la imagen
Podría acumularse sólo en una porción de la misma



O afecte pequeños pixeles que pasan desapercibidos para el ojo.

O pensar que esa es la resolución original y no la versión con ruido

Integridad de la Información: Texto I

Los hermanos sean unidos
porque ésa es la ley primera,
tengan unión verdadera,
en cualquier tiempo que sea,
porque si entre ellos pelean
los devoran los de afuera.

Un padre que da consejos,
más que padre es un amigo
y así como tal les digo
que vivan con precaución
que nadie sabe en que rincón
se esconde el que es su enemigo.

José Hernández (*Martín Fierro)*

Los herm&#”s sean unidos
porque ésa es la ley primera,
tengan unión verdadera,
&! Cual*[}r tiempo que sea,
porque si xxntre yyzns pelean
los devoran los de afuera.

¡? padre que da consej(/,
más que padre es un amigo
y así como mz2 3r% digo
858 vivan con precaución
que nadie sabe en que rincón
4? =22pde el que es su enemigo.

José Hernández (*Martín Fierro)*

Integridad de la Información: Texto II

Según un etsduio de una uivennrsdiad ignlsea no ipmotra el odren en el que las ltears etsen ersciats, la uicna csoa ipormnte es que la pmrirea y la utlima ltera esten ecsritas en la psiocion cocrrtea. El rsteo peuden estar taotlmntee mal y aun prodas lerelo sin pobrleams. Etso es pquore no lemeos cada ltera en si msima, pero si la paalbra cmoo un todo. ¿No te parcee aglo icrneible?

Es más fácil detectar ruido en un texto, porque las palabras poseen “redundan

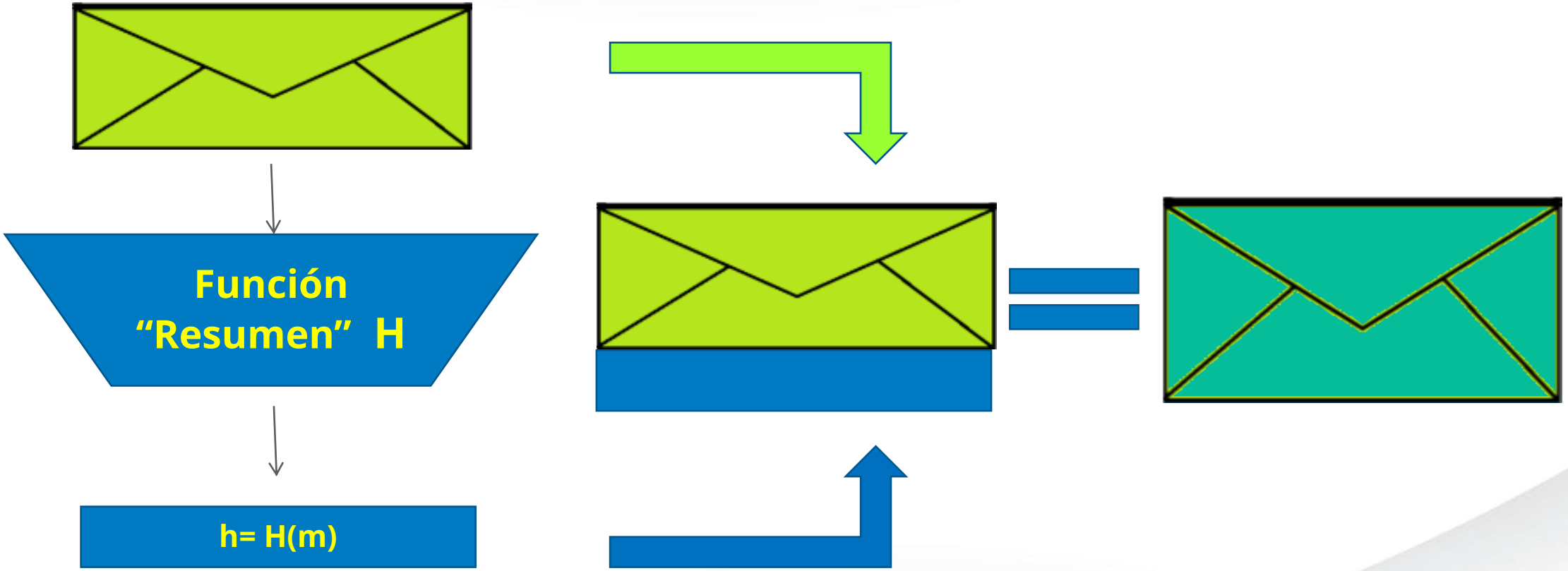
Integridad de la Información: Números

Los números no poseen redundancia y por ello detectar los errores es imposible.

4000 1234 5678 9010

5709 1234 5678 9012

Proceso que se corre en el Emisor para preservar la Integridad



De acuerdo a las propiedades que cumpla o no la función H, esta podrá ser clasificada como :

- Hash Criptográfico
- Hash No Criptográfico

Funciones de verificación de INTEGRIDAD

- **MECANISMOS**
  - **NO CRIPTOGRÁFICOS *(Checksum)***
    - **Dígitos Verificadores**
    - **Verificación de Redundancia Cíclica** (Cyclic Redundancy Check)
    - **Otros**
  - **CRIPTOGRÁFICOS**
    - **Hash**
      - **Tradicionales**
      - KDF (Key Derivation Function)
    - **MAC (Message Authentication Codes)**

Aplicaciones más difundidas

- Controlar la integridad de los mensajes
- Autenticar mensajes
- Obtención de claves aleatoria
- Almacenamiento seguro de contraseñas
- Huella digital de archivos (Base de datos)

Propiedades de las funciones hash

h(M) será segura si tiene las siguientes características:

1. Unidireccionalidad: conocido un resumen h(M), debe ser computacionalmente imposible encontrar M a partir de dicho resumen.
2. 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.
3. Facilidad de cálculo: debe ser fácil calcular h(M) a partir de un mensaje M.
4. 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.

Colisiones: resistencia débil y fuerte

5. Colisión simple: será computacionalmente imposible conocido M, encontrar otro M' tal que h(M) = h(M'). Esto se conoce como *resistencia débil a las colisiones*.
6. Colisión fuerte: será computacionalmente difícil encontrar un par (M, M') de forma que h(M) = h(M'). Esto se conoce como *resistencia fuerte a las colisiones*.

Ataque por la paradoja del cumpleaños

Para tener *confianza* en encontrar dos mensajes con el mismo resumen h(M) -probabilidad ≥ 50%- no habrá que buscar en 2^n (p.e. 2^{128}), bastará una búsqueda en el espacio $2^{n/2}$ (2^{64}).

¡La complejidad algorítmica se reduce de forma drástica!

Función Resumen, Hash o Message Digest

mensaje

Algunas funciones H tienen limitada la longitud de la entrada. Aunque suele ser tan grande que a los fines prácticos se considera infinita.

Otras funciones, en cambio, no necesitan limitar la longitud de la entrada para su correcto funcionamiento.

Función "Resumen" H(m)

- **Dado m es ´"fácil" calcular h.**
- **Dado h es "imposible" calcular m**
- **Dado m1 es "imposible" hallar m2 tal que**
- **H8m1)=h(m2) (COLISIÓN , SINÓNIMOS)**

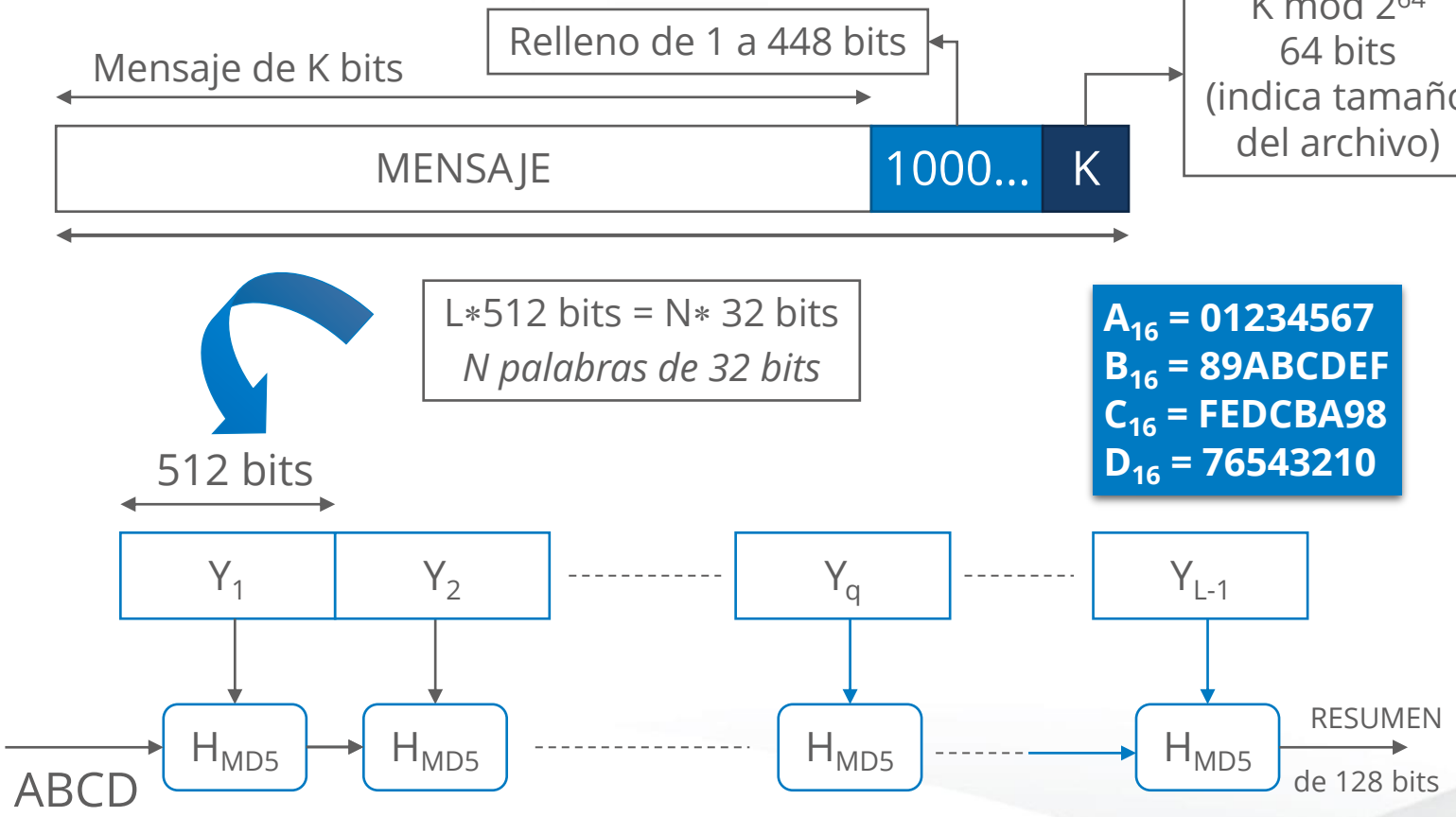
h = H(m)

- **La longitud de la salida puede variar para cada H**
- **Usualmente entre 128 y 1024 bits.**
- **Recibe los nombres de resumen, hash, digest o fingerprint.**
- **Hash: picar).**

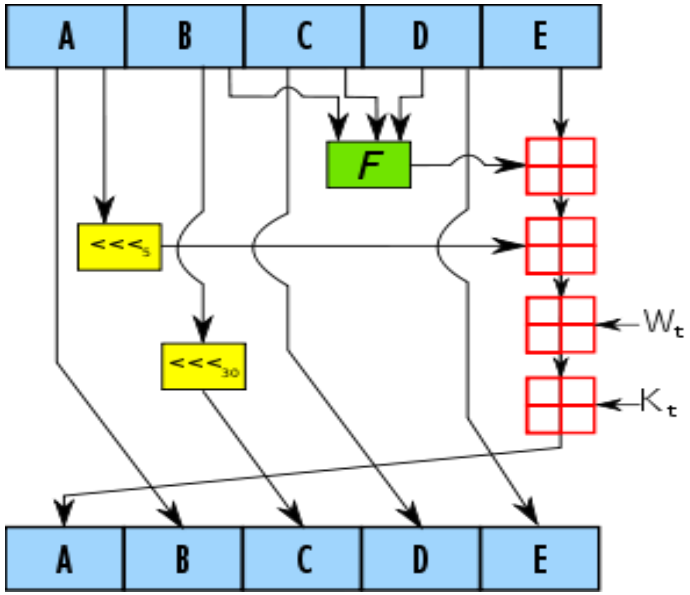
Algoritmos MD5 y SHA-1

- Ambos algoritmos procesan bloques de 512 bits
- emplean 4 funciones primitivas para generar el resumen del mensaje
- Número de pasos (vueltas)
  - MD5 64 que realiza MD5.
  - SHA-1 80
- Resumen obtenido:
  - MD5 entrega 128 bits de resumen.
  - SHA-1 da como resultado 160 bits.

Bloques de 512 bits (caso MD5)



SHA-1 (1995) Big endian, 80 vueltas



Donde inicialmente:

A = 67452301 B = EFCDAB89
C = 98BADCFE D = 10325476
F = C3D2E1F0

Dependiendo de la ronda, la función **F** puede ser F, G, H, I:

F = (**B** AND **C**) OR (NOT **B** AND **D**) 1ª ronda
G = (**B** XOR **C** XOR **D**) 2ª ronda
H = (**B** AND **C**) OR (**B** AND **D**) OR (**C** AND **D**) 3ª ronda
I = (**B** XOR **C** XOR **D**) 4ª ronda

Donde: ⊞ es una suma mod 2^{32}

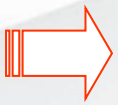
Más diferencias entre MD5 y SHA-1

La longitud máxima del mensaje para SHA-1 debe ser menor de 2^{64} bits, mientras que MD5 no tiene limitaciones de longitud.

MD5 emplea 64 constantes (una por cada paso), mientras que SHA-1 sólo emplea 4 (una para cada 20 pasos).

MD5 se basa en la arquitectura little-endian, mientras que SHA-1 se basa en la arquitectura big-endian. Por ello el vector ABCD inicial en MD5 y SHA-1 son iguales:

- `A = 01234567 (MD5) ⇒ 67452301 (SHA-1)`
- `B = 89ABCDEF (MD5) ⇒ EFCDAB89 (SHA-1)`
- `C = FEDCBA98 (MD5) ⇒ 98BADCFE (SHA-1)`
- `D = 76543210 (MD5) ⇒ 10325476 (SHA-1)`



Arquitecturas little-endian y big-endian

Arquitectura little-endian:

Esta es la arquitectura empleada en procesadores Intel de la familia 80xxx y Pentium.

Para almacenar una palabra en memoria, el byte menos significativo de los que forman dicha palabra se guarda en la posición más baja de la memoria.

Arquitectura big-endian:

Empleada por otras arquitecturas como SUN.

Para almacenar una palabra en memoria, el byte más significativo de los que forman dicha palabra se guarda en la posición más baja de memoria.

http://www.algoritmia.net/articles.php?id=57

Características de la familia SHA
Secure hash algorithm

- Publicados por el NIST
- SHA-0 (1993).
- SHA-1 (1995). En 1998 se encontró una vulnerabilidad en SHA-0 y por las dudas la NSA “elevó la seguridad del SHA-1”
- SHA-2 (2001) en sus diferentes versiones:

  SHA-224, SHA-256, SHA-384, y SHA-512
- SHA-3(2012)

Funciones Resumen más conocidas

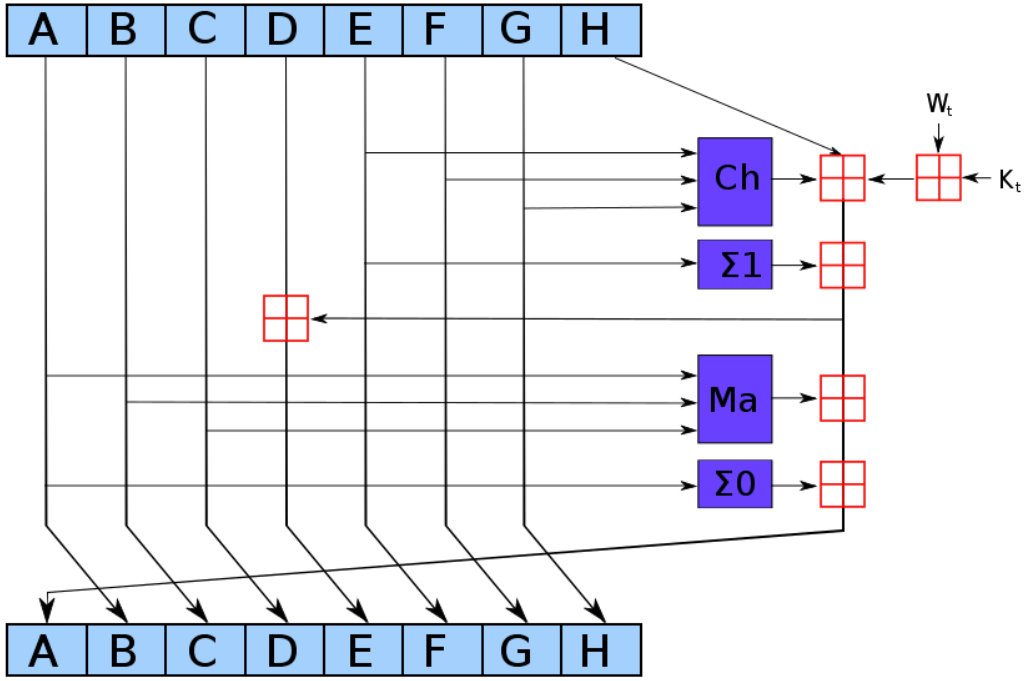
Algoritmo y variante		Tamaño de salida (bits).	Tamaño del estado interno (bits).	Tamaño del bloque (bits).	Tamaño máximo del mensaje (bits).	Longitud de la palabra (bits).	Iteraciones	Operaciones	Colisiones encontradas	Ejemplo de rendimiento (MiB/s)[13]
MD5 (como referencia)		128	128	512	$2^{64} - 1$	32	64	+, and, or, xor, rot	✗ **Si**	335
SHA-0		160	160	512	$2^{64} - 1$	32	80	+, and, or, xor, rot	✗ **Si**	-
SHA-1		160	160	512	$2^{64} - 1$	32	80	+, and, or, xor, rot	✗ **Si**	192
SHA-2	*SHA-224* *SHA-256*	224 256	256	512	$2^{64} - 1$	32	64	+, and, or, xor, shr, rot	Ninguna	139
	SHA-384 *SHA-512* *SHA-512/224* *SHA-512/256*	384 512 224 256	512	1024	$2^{128} - 1$	64	80	+, and, or, xor, shr, rot	Ninguna	154
SHA-3		224/256/384/512	1600 (5×5 array de palabras de 64-bit)	1152/1088/832/576	Ilimitado	64	24	and, xor, not, rot	Ninguna	

SHA: Secure Hash Algorithm
MD: Message Digest

SHA-2 (2001) y SHA-3 (2015)

- Problemas con MD5 en verano de 2004
- El NIST reacciona tarde y publica una nota de prensa el 23 de enero 2007 con un borrador para el nuevo estándar de hash, que se llamará SHA-3
- En diciembre de 2010 el NIST selecciona en la tercera ronda a cinco finalistas: BLAKE, Ghøstl, JH, Keccak y Skein
- En octubre de 2012, NIST anuncia que el ganador es **Keccak**, desarrollado por los investigadores belgas Joan Daemen, Michael Peeters, Gilles Van Assche y el italiano Guido Bertoni
- SHA-3 es presentado como estándar en el NIST el 6 de febrero de 2013
- Se publica el estándar en agosto de 2015
- De momento usamos SHA-256

SHA-256

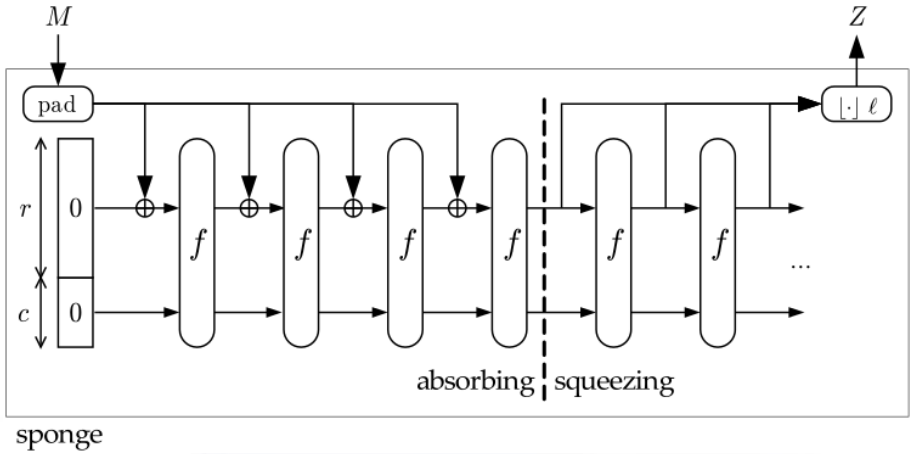


Ch (E ,F ,G) = (E AND F) ⊕ (NOT E AND G)
Ma (A ,B ,C) = (A AND B) ⊕ (A AND C) ⊕ (B AND C)
Σ0 (A) = (A ⋙ 2) ⊕ (A ⋙ 13) ⊕ (A ⋙ 22)
Σ1 (E) = (E ⋙ 6) ⊕ (E ⋙ 11) ⊕ (E ⋙ 25)

Donde: ⊞ es una suma mod 2^{32}

SHA-3 Construcción esponja

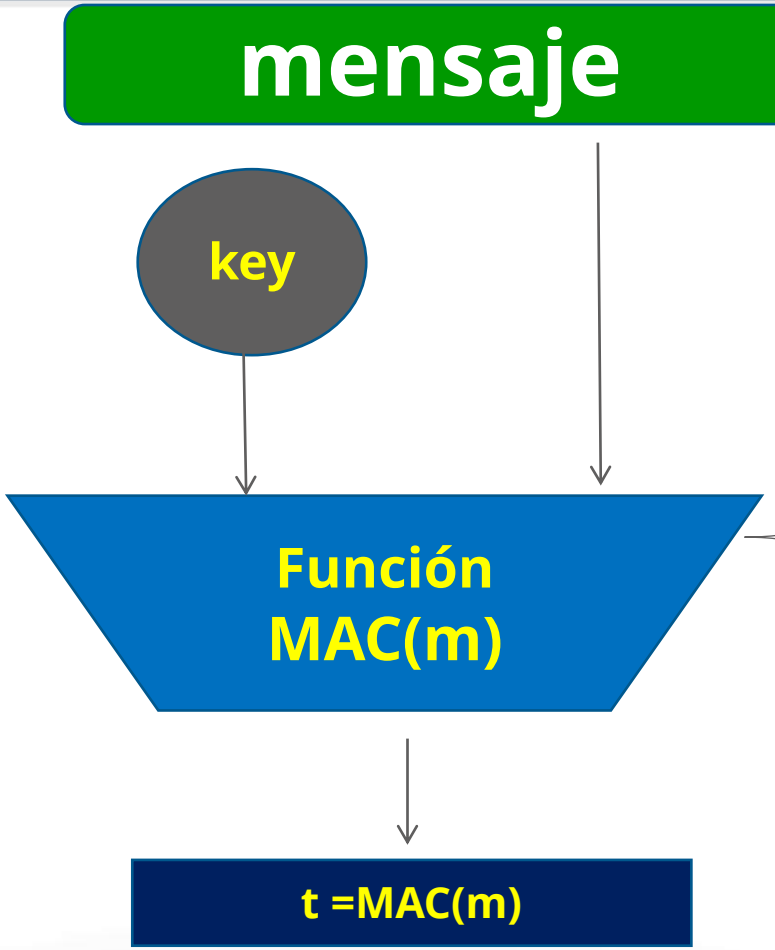
- Construcción esponja: función *F(M) = H = h(M)*
  - Entrada de longitud variable y salida de longitud variable
  - Permutación (o transformación) *f* de longitud fija
    - Ancho: número de bits = *b*
  - *O*pera sobre un estado de *b* bits, donde *b = r + c*
    - *r = bits of rate* y c = *bits of capacity*
- *M* + relleno (pad)
  - Bloques de *r* bits
  - S = estado (*b* bits) inicializado a cero
- Dos fases
  - Absorción (entrada)
  - Exprimido (salida)



Funciones hash para la autenticación

- Las funciones hash vistas (MD5, SHA-1, etc.) pueden usarse además para autenticar a dos usuarios.
- Como carecen de una clave privada no pueden usarse de forma directa para estos propósitos. No obstante, existen algoritmos que permiten añadirles esta función.
- Entre ellos está HMAC, una función que usando los hash vistos y una clave secreta, autentica a dos usuarios mediante sistemas de clave secreta. Las funciones MAC, Message Authentication Code, y HMAC se tratarán en el próximo capítulo dedicado a la autenticación y firma digital.
- HMAC se usa en plataformas IP seguras como por ejemplo en Secure Socket Layer, SSL.

Código Autenticador de Mensaje MAC



No es necesario algoritmos “especiales” para desarrollar un MAC. Se pueden “reciclar” Algorimos de Bloque (Block Cipher) como así también algoritmos de Hash. Hay que determinar el manejo de la clave k y aseverar que cumple con las propiedades respectivas.

- **Dado m es ´”fácil” calcular t.**
- **Dado t es “imposible” calcular m**
- **Dado m_1 es “imposible” hallar m_2 tal que**
- **$MAC(m_1)=MAC(m_2)$ (COLISIÓN , SINÓNIMOS)**

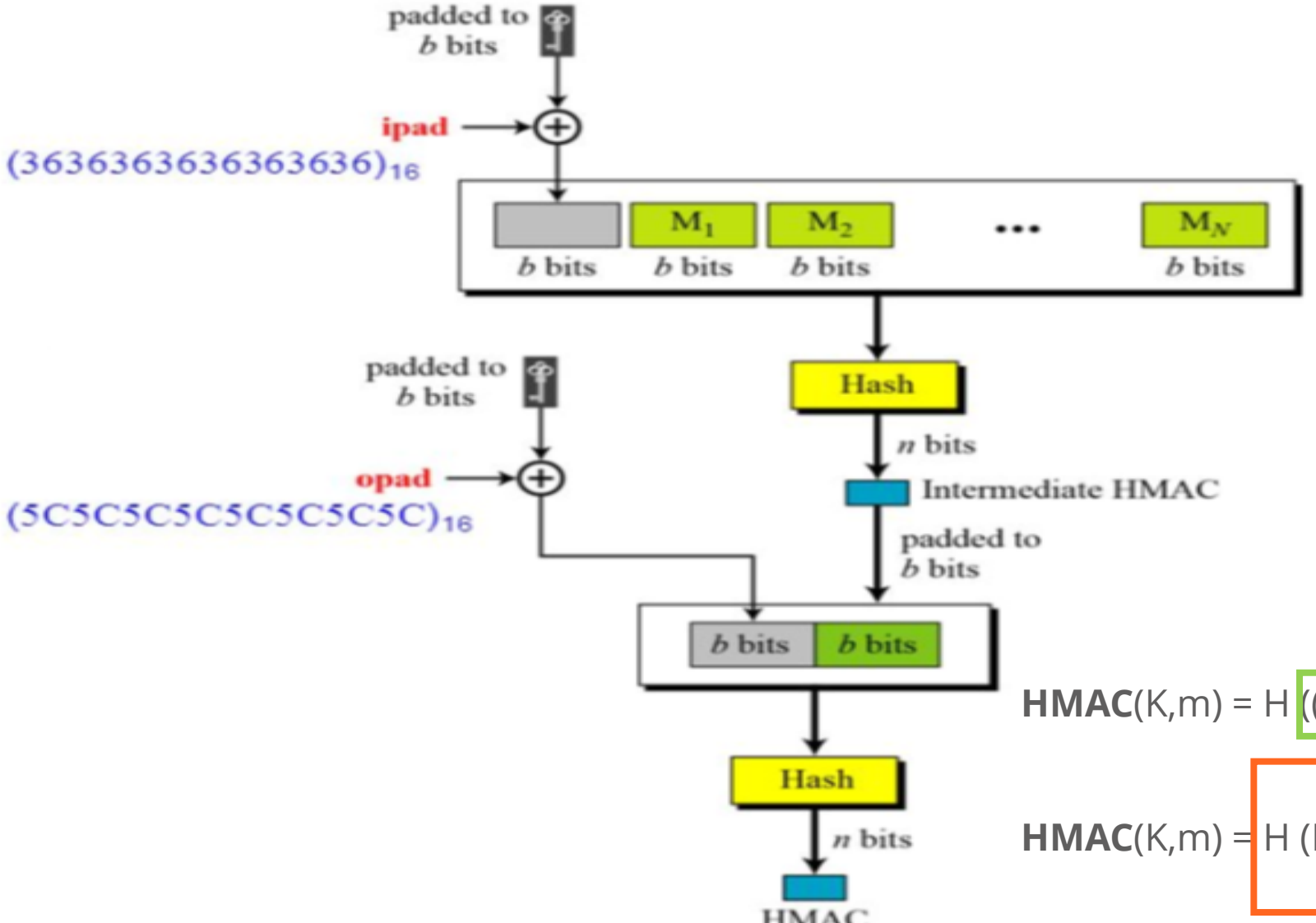
- **Dados $MAC(m_1);MAC(m_2),...MAC(m_n)$ no se puede obtener el valor de la clave k utilizada.**

Equivalentes al HASH

- **Usualmente 128,192 Y 256 Bits.**
- **Recibe el nombre de ”autenticador” o TAG**

Nota: MD5, SHA1 y SH2 vulnerables al Ataque de extensión de longitud (Merkle-Damgard)

HASH-MAC (HMAC)



Se puede usar cualquier algoritmo de hash y concatenar la clave *k* y el mensaje a autenticar. De manera que luego ese sea el texto a ingresar en el hash. *Su descripción se puede hallar en RFC 2104*

HMAC(K,m) = H ((**K** xor **opad**) | |H(**K** xor **ipad**)| | m)))

HMAC(K,m) = H (K2 | | H(K1 | |m))

Problemas de Seguridad aún con Hash

A) Algoritmos “determinísticos”

```
| bill@gander.org      | 5f4dcc3b5aa765d61d8327deb882cf99 (password)   |
| heinrich@supplier.de | 5f4dcc3b5aa765d61d8327deb882cf99 (password)   |
```

B) Tablas Arco Iris (Rainbow Tables) y Lookup Tables

crackstation.net

```
| tommy@customer.net   | 7f43c1e438dc11a93d19616549d4b701                 |
```

CrackStation

CrackStation » Password Hashing Security » Defuse Security »

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
7f43c1e438dc11a93d19616549d4b701
```

No soy un robot — reCAPTCHA Privacidad - Condiciones

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
7f43c1e438dc11a93d19616549d4b701	md5	luv2buy

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries here, and the lookup table implementation (PHP and C) is available here.

C) Claves son “recordables” (baja entropía): Ataque por Diccionario

```
(test)
(secret)
(iforgot)
(money)
(supplier)
(cheap)
(password)
(profit)
(whole)
(helpme)

(carbondale)

(secret)
(California)
(Disneyland)
(password)
```

5ebe2294ecd0e0f08eab7690d2a6ee69

7f43c1e438dc11a93d19616549d4b701

NIST Special Publication 800-63B
Digital Identity Guidelines
Authentication and Lifecycle Management

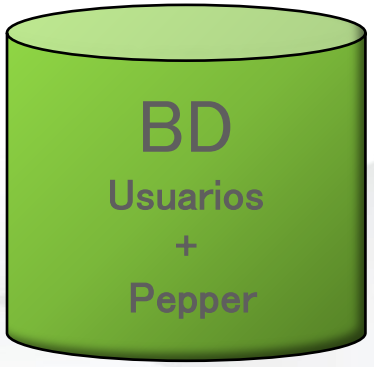
"La sal DEBE tener al menos 32 bits de longitud y se elegirá arbitrariamente para minimizar las colisiones de valores de sal entre los valores hash almacenados. Tanto el valor de sal como el hash resultante DEBERÁN almacenarse para cada suscriptor utilizando una autenticación secreta memorizada."

*Para PBKDF2 (*Password-based Key Derivation Function 2*), el factor de costo es un recuento de iteraciones: cuantas más veces se itera la función PBKDF2, más tiempo se tarda en calcular el hash de la contraseña. Por lo tanto, el recuento de iteraciones DEBERÍA ser tan grande como lo permita el rendimiento del servidor de verificación, por lo general al menos 10,000 iteraciones*

Almacenamiento de las Factores de AutenticaciónMecanismo 4

Usuario	HMAC	Salt
Aaaa	Llk@C%&430/&5v	10001
Bbbb	89$5%%wrm//#	101111
Cccc	==9C%&43*&")¿	1111111
dddd	"#$#{p&6Xz>¿*}:	000100

Usuario	Pepper
Aaaa	11111
Bbbb	10110
Cccc	11000
dddd	000100



https://pages.nist.gov/800-63-3/sp800-63b.html

Tiempos de Ataque

Cluster : 25 GPU's solamente con el propósito de crackear *hashes*.

Alfabeto : Mym, dígitos, caracteres especiales: *~`!@#$%^&*()_+-={}|[]¥:";'<>,.?/*

Algoritmo	Claves/seg	Tiempo
md5($password)	180 Billones/seg	9,4 horas
sha1($password)	61 Billones/seg	27 horas
md5crypt($password)	77 Millones /seg	2,5 años
bcrypt	71mil/seg	2700 años

http://www.michael-pratt.com/blog/8/Seguridad-en-el-almacenamiento-de-PasswordsContrasenas/

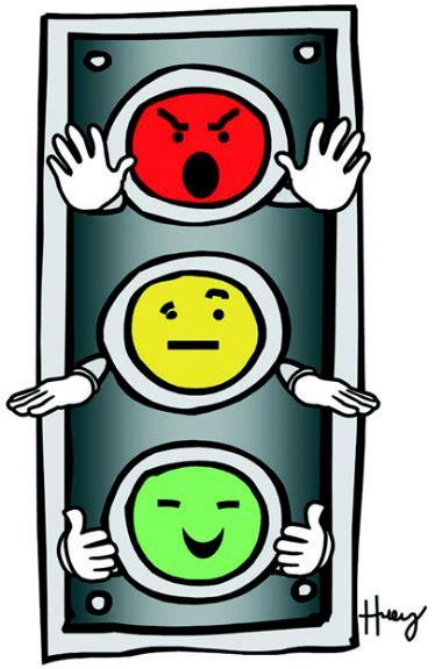
Algoritmos de HASH recomendados para contraseñas

HASH

- MD5 (1991)
- SHA1 (1995)
- Sha256crypt (2001)
- Sha512crypt (2001)

KDF Key Derivation Function

- PBKDF2 (2000)
- BCrypt (1999)
- Argon2 (2015)
- SCrypt (2009)



https://pthree.org/2016/06/28/lets-talk-password-hashing/
https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e
https://cryptobook.nakov.com/mac-and-key-derivation/kdf-deriving-key-from-password
https://pthree.org/2018/05/23/do-not-use-sha256crypt-sha512crypt-theyre-dangerous/

Aplicaciones reales

Ejemplos aplicados:

- echo “MENSAJE" | openssl dgst -sha256
- echo " MENSAJE" | openssl dgst -sha1
- echo " MENSAJE " | sha1sum
- echo "INFORMACION A ENVIAR " | openssl dgst -sha1 -hmac "CLAVE"

¡Gracias!