Funciones Hash

- Función Hash: Función que asocia a una cadena de longitud arbitraria otra de longitud fija.
- Función Hash débilmente libre de colisiones (*second-preimage resistance*): Aquella que dado x es computacionalmente imposible hallar $x' \neq x$ tal que $h(x) = h(x')$.
- Función Hash fuertemente libre de colisiones (*collision resistance*): Aquella para la cual es computacionalmente imposible hallar x y x' tales que $h(x) = h(x')$.
- Función Hash unidireccional (*preimage resistance)*: Aquella que dado z es computacionalmente imposible hallar x tal que $h(x) = z$.

Toda función hash fuertemente libre de colisiones es unidireccional.

Funciones Hash

SHA 256 (224)

SHA 512 (384)

SHA1: NO usar en nuevas aplicaciones. Hay ataques más eficientes que los basados en la *paradoja del cumpleaños.*
Ya se han encontrado colisiones https://shattered.io/

MD5: NO usar en nuevas aplicaciones. Está completamente roto.
Ver por ejemplo: http://www.cits.rub.de/MD5Collisions
http://www.mathstat.dal.ca/~selinger/md5collision

Arquitectura SHA1, SHA2

Merkle-Damgård hash function

- Preproceso:
  - Añadir *padding*.
  - Trocear el resultado en bloques, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$, de longitud fijada.
  - Inicializar los valores H_i del hash.
- Proceso: Procesar los bloques uno a uno:
  `Para` $t = 1...N$
  $(H_0, ..., H_k) = f(H_0, ..., H_k, M^{(t)})$
- Hash: $H_0 \| ... \| H_k$

SHA512
Secure Hash Standard (SHS)

SHA-3 http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

- 2/11/2007 el NIST hace un llamamiento público para desarrollar una nueva función hash criptográficamente segura.
- 31/10/2008: NIST recibe 64 propuestas,
- 10/12/2008: 51 son admitidas en la primera ronda.
- 24/07/2009: 14 pasan a la segunda ronda. Se abre un periodo de un año de exposición pública para presentar comentarios.
- 09/12/2010: NIST selecciona 5 finalistas: BLAKE, Grøstl, JH, Keccak, Skein
- El 2 de octubre de 2012 se anunció el ganador: **Keccak**.

Sha-3 standardization. *FIPS 202, SHA-3 Standard.*

SHA-3 Selection Announcement

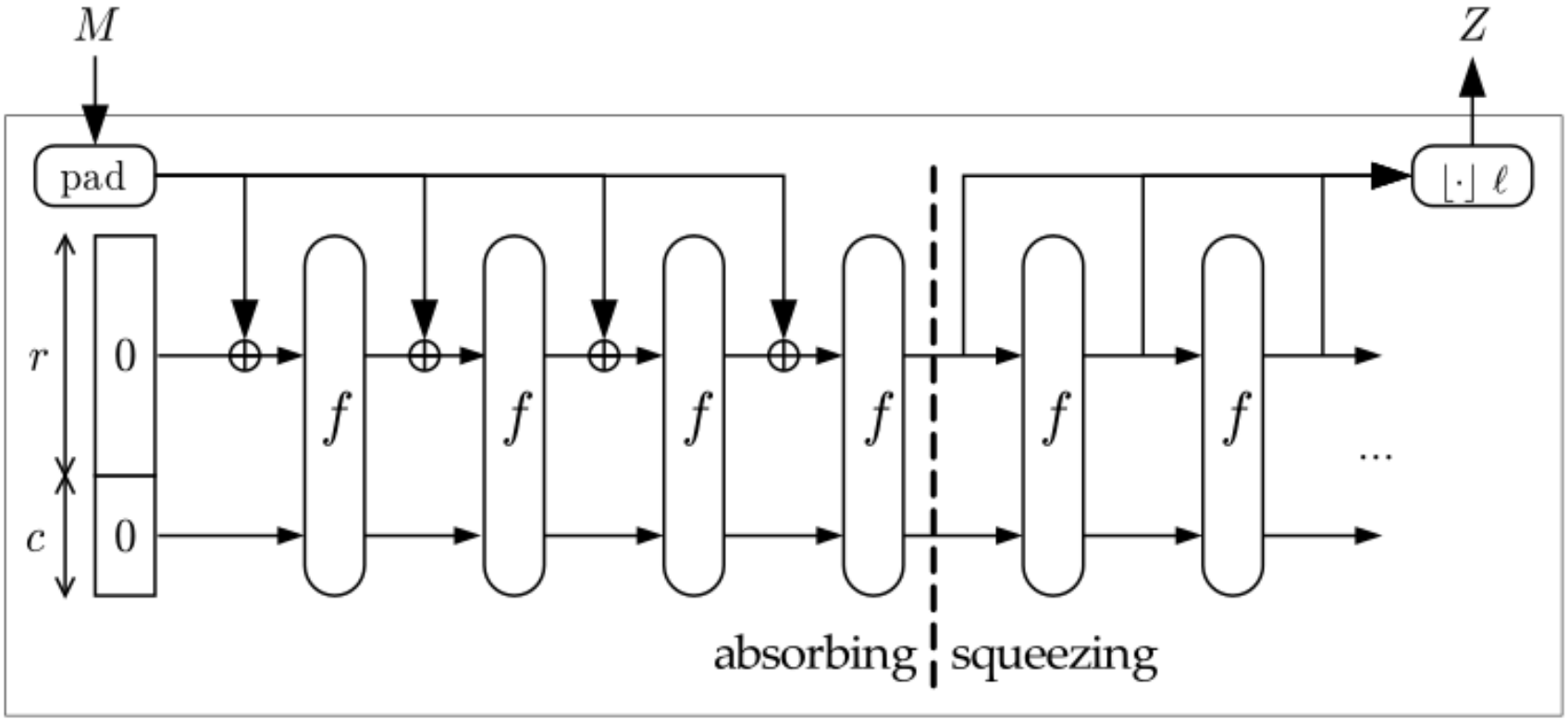
[...] NIST chose KECCAK [...] for its elegant design, large security margin, good general performance, excellent efficiency in hardware implementations, and for its flexibility. KECCAK uses a new ***sponge construction*** chaining mode, based on a fixed permutation, that can readily be adjusted to trade generic security strength for throughput, and can generate larger or smaller hash outputs as required. [...]

Additionally, KECCAK complements the existing SHA-2 family of hash algorithms well. NIST remains confident in the security of SHA-2 which is now widely implemented, and the SHA-2 hash algorithms will continue to be used for the foreseeable future, as indicated in the NIST hash policy statement. **One benefit that KECCAK offers as the SHA-3 winner is its difference in design and implementation properties from that of SHA-2.** It seems very unlikely that a single new cryptanalytic attack or approach could threaten both algorithms. Similarly, the very different implementation properties of the two algorithms will allow future application and protocol designers greater flexibility in finding one of the two hash algorithms that fits well with their requirements. [...]

The sponge construction

http://sponge.noekeon.org/

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 transformation f operating on a fixed number b of bits. 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.



sponge

SHA-3 Family

- SHA3-224, SHA3-256, SHA3-384, SHA3-512: The SHA-3 hash function that produces 224, 256, 384, 512 bit digests.
  In each case, the capacity is double the digest length, i.e., $c = 2d$,

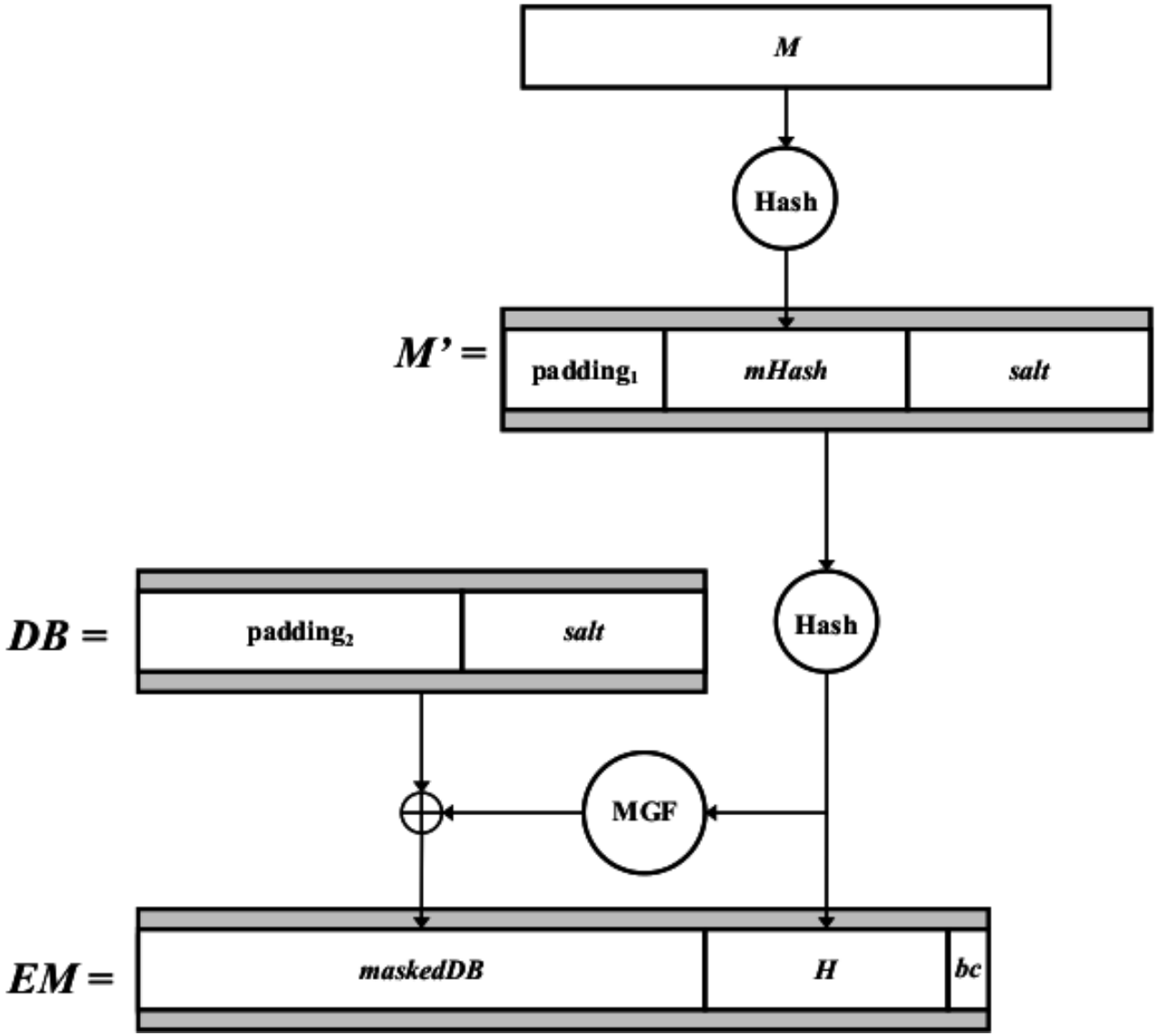
- SHAKE128, SHAKE256: The SHA-3 XOF *extendable-output function* that generally supports 128, 256 bits of security strength, if the output is sufficiently long ($d \geq 256/512$ minimun).
  The capacity is 256, 512.

Security strengths of SHA-1, SHA-2, SHA-3

	Size	**Collision**	**Preimage**	**2nd Preimage**
SHA-1	160	< 80	160	160-L(M)
SHA-224	224	112	224	min(224,256-L(M))
SHA-512/224	224	112	224	224
SHA-256	256	128	256	256-L(M)
SHA-512/256	256	128	256	256
SHA-384	384	192	384	384
SHA-512	512	256	512	512-L(M)
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE128	d	min(d/2, 128)	$\geq$min(d, 128)	min(d, 128)
SHAKE256	d	min(d/2, 256)	$\geq$min(d, 256)	min(d, 256)

L(M)=$\lceil \log_2(len(M)/B) \rceil$, B is the block length of the function in bits, B=512 for SHA-1, SHA-224, SHA-256, and B=1024 for SHA-512.

RSA https://tools.ietf.org/html/rfc3447



Firmas basadas en funciones hash (*hash-based signatures*)

No están basadas en los mismos principios que la criptografía de clave pública sino que hacen uso de funciones *hash* para realizar la firma en sí.

Su principal inconveniente es que claves y firmas son muy grandes comparadas con las de la criptografía de clave pública.

Su ventaja, de cara al futuro, es su *resistencia* a los ordenadores cuánticos.

Lamport One Time Signature (LOTS) (I)

Una clave sólo se puede usar para una firma.

- **Clave privada**

$$(x_1, y_1), \ldots, (x_k, y_k)$$

siendo k el número de bits de la salida de la función HASH usada.

- **Clave pública**

$$(Pub_{x_1}, Pub_{y_1}), \ldots, (Pub_{x_k}, Pub_{y_k})$$

siendo $Pub_{x_i} = HASH(x_i)$, $Pub_{y_i} = HASH(y_i)$.

Si se usa SHA256, $k = 256$ y el tamaño de la clave privada y de la pública es 16Kbytes (ya que x_i e y_i tienen 256 bits cada uno, cómo mínimo, y necesitamos 256 pares.

Lamport One Time Signature (LOTS) (II)

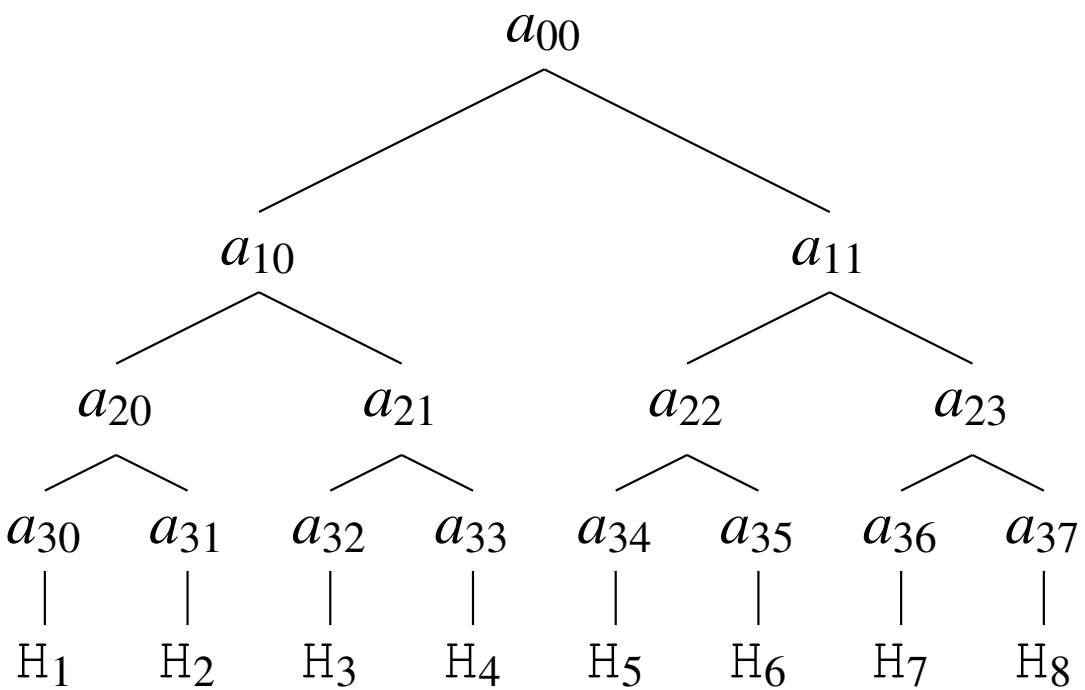
- **Firma** del documento M.
  Sea $h = HASH(M)$, $h = h_1 h_2 \dots h_k$, siendo h_i los bits de h.
  Definimos:
  $$S_i = \begin{cases} x_i & \text{si } h_i = 0, \\ y_i & \text{si } h_i = 1, \end{cases}$$
  La firma del documento M es $S = S_1, S_2, \dots, S_k$

- **Verificación** de la firma $S = S_1, S_2, \dots, S_k$ del documento M con la clave $(Pub_{x_1}, Pub_{y_1}), \dots, (Pub_{x_k}, Pub_{y_k})$.
  Sea $h = HASH(M) = h_1 h_2 \dots h_k$
  $$\begin{cases} \text{si } h_i = 0 \text{ comprueba que } HASH(S_i) = Pub_{x_i} \\ \text{si } h_i = 1 \text{ comprueba que } HASH(S_i) = Pub_{y_i}, \end{cases}$$

Merkle Signature Scheme (I)

Generación de claves (Una clave permite firmar $N = 2^k$ mensajes.)

- Se generan N pares de claves LOTS pública-privada $(Pub_i, Priv_i)$.
- Se calcula $\mathtt{H}_i = HASH(Pub_i)$
- Con $\mathtt{H}_i$ como hojas se genera un *hash-tree*: cada padre es el hash de sus hijos.



La **clave pública** es la raíz a_{00} del *hash-tree* (256 bits si se usa SHA256).

La **clave privada** son los pares $(Pub_i, Priv_i), i = 1, \ldots, N$

Merkle Signature Scheme (II)

Firma: Para firmar M elegimos el primer par $(Pub_i, Priv_i)$ no usado y lo firmamos con Lamport One Time Signature, siendo el resultado $\widetilde{Sig}$.

Para demostrar que la clave usada forma parte de la clave pública se revela Pub_i y los nodos necesarios del *hash-tree* que permiten recalcular la raíz. Por ejemplo, si $(Pub_5, Priv_5)$ es la clave usada para firmar publicamos $a_{10}, Pub_5, \mathtt{H}_6, a_{23}$; con Pub_5 calculamos $\mathtt{H}_5$, con $\mathtt{H}_5, \mathtt{H}_6$ calculamos a_{22}; con a_{22}, a_{23} calculamos a_{11}; y con a_{10}, a_{11} calculamos la raíz a_{00}. (Hay que publicar k nodos –profundidad del árbol– y la Pub_i.)

La firma es $Sig = (\widetilde{Sig}, Pub_i, k \text{ nodos})$.

Verificación de la firma:

- Con $\widetilde{Sig}$ y Pub_i se verifica la LOTS.
- Con los k nodos y Pub_i se verifica que se ha usado la clave pública a_{00}.