# 6-Funciones MAC y Hash

# Motivación

- En el contexto de las condiciones, ya dijimos se necesitaban una serie de servicios de seguridad, entre los más comunes:
  - Confidencialidad: $A \Rightarrow M \Rightarrow B$, solo $A$ y $B$ pueden entender el mensaje $M$. Solo $A$ y $B$ conocen el mensaje $M$ (solo $A$ y $B$ conocen la clave).
  - Autenticación: $A \Rightarrow M \Rightarrow B$, $B$ tiene la certeza que el mensaje viene de $A$.

$$C$$

  - Firma: $A \Rightarrow M \Rightarrow B$, $C$ tiene la certeza que el mensaje que llega a $B$ viene de $A$.

# Motivación

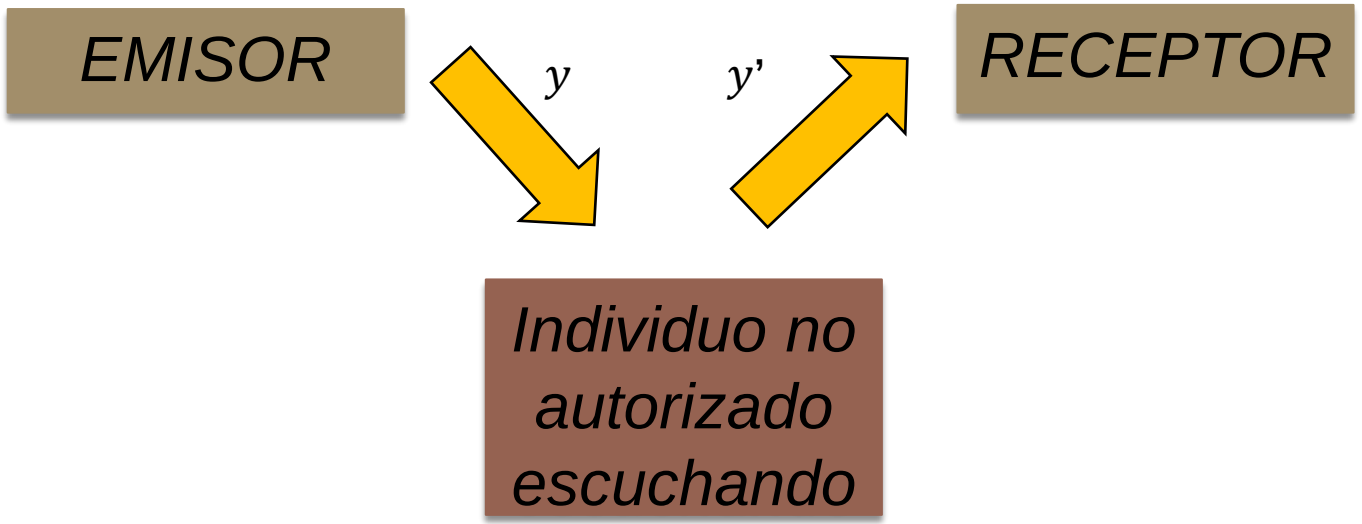
- Como se pueden proporcionar estos servicios:
  - Criptografía simétrica
  - Criptografía pública
  - Mac y Hash

# Motivación: Servicios en Criptografía simétrica

- Supongamos que utilizamos un protocolo para proporcionar confidencialidad con criptografía simétrica:
  - $A: M \Rightarrow E_k \Rightarrow Canal\ Inseguro \Rightarrow D_k \Rightarrow M: B$ (en principio este protocolo da Confidencialidad y Autentificación)
- Qué pasa si hay un ataque por modificación (se viola la integridad):

# Motivación: Servicios en Criptografía simétrica

- Sustituye $y$ por $y'$ en el canal inseguro:
  - $A: M \Rightarrow y = E_k(M) \Rightarrow y \Rightarrow Canal\ Inseguro \Rightarrow y' \Rightarrow D_k\ (y') \Rightarrow M': B$



# Motivación: Servicios en Criptografía simétrica

- Supongamos $x \in P \Rightarrow \frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \ll 1.$
- Por tanto $B$ se da cuenta que el mensaje recibido ha sido interceptado y cambiado, o se ha producido un error en la transmisión.
- Por lo tanto este protocolo trivial es capaz de producir confidencialidad y autentificación.
- Sin embargo si no se cumple que $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \ll 1$, es decir hay muchos mensajes con sentido, es posible que el $y'$ pueda llegar a tener sentido.
- De esta forma estos dos servicios proporcionados anteriormente se pierden.

# Motivación: Servicios en Criptografía simétrica

- Para solucionar este problema se puede utilizar una función FCS (*frame check sequence*): es un código de detección de errores agregado a una trama en un protocolo de comunicación.
- Así esta función genera un código que depende del mensaje, $FCS(M)$, de tamaño $n$ bits.
- Para utilizar esta función podemos seguir dos opciones.
- Opción 1:

$$A: M \Rightarrow y = E_k(M) \Rightarrow [y \,||\, FCS(y)] \Rightarrow Canal\ Inseguro$$

$$\Rightarrow [y \,||\, FCS(y)] \Rightarrow Compara \begin{cases} FCS(y) \\ FCS(y) \end{cases} \Rightarrow D_k(y) \Rightarrow M: B$$

- Pero esta opción no es buena ya que un atacante puede cambiar el paquete enviado $[y \,||\, FCS(y)]$ por $[y' \,||\, FCS(y')]$, y $B$ no se va a dar cuenta al calcular la $FCS(y')$, ya que precisamente en el bloque enviado está concatenado ese valor $FCS(y')$.
- Esto pasa debido a que el código de tamaño $n$ bits $FCS(M)$ no va protegido.

# Motivación: Servicios en Criptografía simétrica

- Opción 2:

  $A: M \Rightarrow y = E_k\,[M \,||\, FCS(M)] \Rightarrow Canal\ Inseguro \Rightarrow y \Rightarrow D_k(y) = [M \,||\, FCS(M)]$

  $\Rightarrow Compara \begin{cases} FCS(M) \\ FCS(M) \end{cases} \Rightarrow M: B$

- Esta opción es la correcta ya que el código de tamaño $n$ bits de $FCS(M)$ va protegido con la clave del cifrado.
- Ahora bien cuál es la probabilidad de un observador elija bien el código de tamaño $n$ bits de $FCS(M)$:
  - Si $FCS(M)$ es de un bit la probabilidad es de $P = \frac{1}{2}$
  - Si $FCS(M)$ es de dos bits la probabilidad es de $P = \frac{1}{4}$
  - …
  - Si $FCS(M)$ es de $n$ bits la probabilidad es de $P = \frac{1}{2^n}$
- Las funciones $FCS(M)$ se implementan mediante funciones Hash y Mac.

# Motivación: Servicios en Criptografía simétrica

- El servicio de firma no se puede proporcionar con criptografía simétrica.
- No se puede controlar por la simetría de la clave.
- Debido a la simetría de la clave no sabemos si manda $A$ o manda $B$.
- Es decir un observador externo no sabe quién firma de los dos, ya que la clave es la firma para los dos.

# Motivación: Servicios en Criptografía Pública

- Supongamos el siguiente protocolo con criptografía pública:
  - $A: M \Rightarrow E_{e_B} \Rightarrow Canal\ Inseguro \Rightarrow D_{d_B} \Rightarrow M: B$
- En este protocolo hay Confidencialidad (solo $A$ y $B$ pueden entender el mensaje $M$).
- No hay autentificación, ya que $e_B$ es pública y cualquiera la puede utilizar. Es decir $B$ no puede tener la certeza de que el mensaje venga de $A$.
- Tampoco $A$ está seguro que el mensaje le va a llegar a $B$, ya que nadie te asegura que $e_B$ sea realmente de $B$ (necesitamos certificados X.509).
- Firma tampoco tiene.

# Motivación: Servicios en Criptografía Pública

- Supongamos el siguiente protocolo con criptografía pública:
  - $A: M \Rightarrow E_{d_A} \Rightarrow Canal\ Inseguro \Rightarrow D_{e_A} \Rightarrow M: B$
- En este protocolo hay Autentificación ($B$ tiene la certeza que el mensaje viene de $A$).
- No hay confidencialidad, ya que $e_A$ es pública y cualquiera la puede utilizar para descifrar el mensaje. Es decir el mensaje que envía $A$ lo puede ver todo el mundo mediante $e_A$.
- Tampoco $B$ está seguro que el mensaje le va a llegar proviene realmente de $A$, ya que nadie te asegura que $e_A$ sea realmente de $A$ (necesitamos certificados X.509).

# Motivación: Servicios en Criptografía Pública

- El siguiente protocolo mejora la situación bastante:
  - $A: M \Rightarrow E_{e_B} \Rightarrow E_{d_A} \Rightarrow Canal\ Inseguro \Rightarrow D_{e_A} \Rightarrow D_{d_B} \Rightarrow M: B$
- Este sencillo protocolo tiene Autenticación, Confidencialidad y Firma.
- Este protocolo autentica ya $B$ tiene la certeza que el mensaje viene de $A$, ya que solo este conoce $d_A$.
- Además tiene firma ya $A$ solo conoce $d_A$.
- Pero aún no se soluciona el problema de que $A$ no está seguro que el mensaje que envía le llega a $B$ realmente, ya que nadie te asegura que $e_B$ sea realmente de $B$ (necesitamos X.509 en la siguiente transparencia).
- Parece que con criptografía pública solucionamos algunas cosas importantes, como la firma digital.
- Pero es muy costoso cifrar con criptografía pública.
- La criptografía pública se suele utilizar para cifrar claves (que luego se utilizan con criptografía simétrica), o para firma digitalmente.
- Para generar los servicios básicos que hemos visto con criptografía simétrica necesitamos las funciones Mac y Hash, que es lo que vamos a ver a continuación.

# Motivación: Servicios en Criptografía Pública, X.509

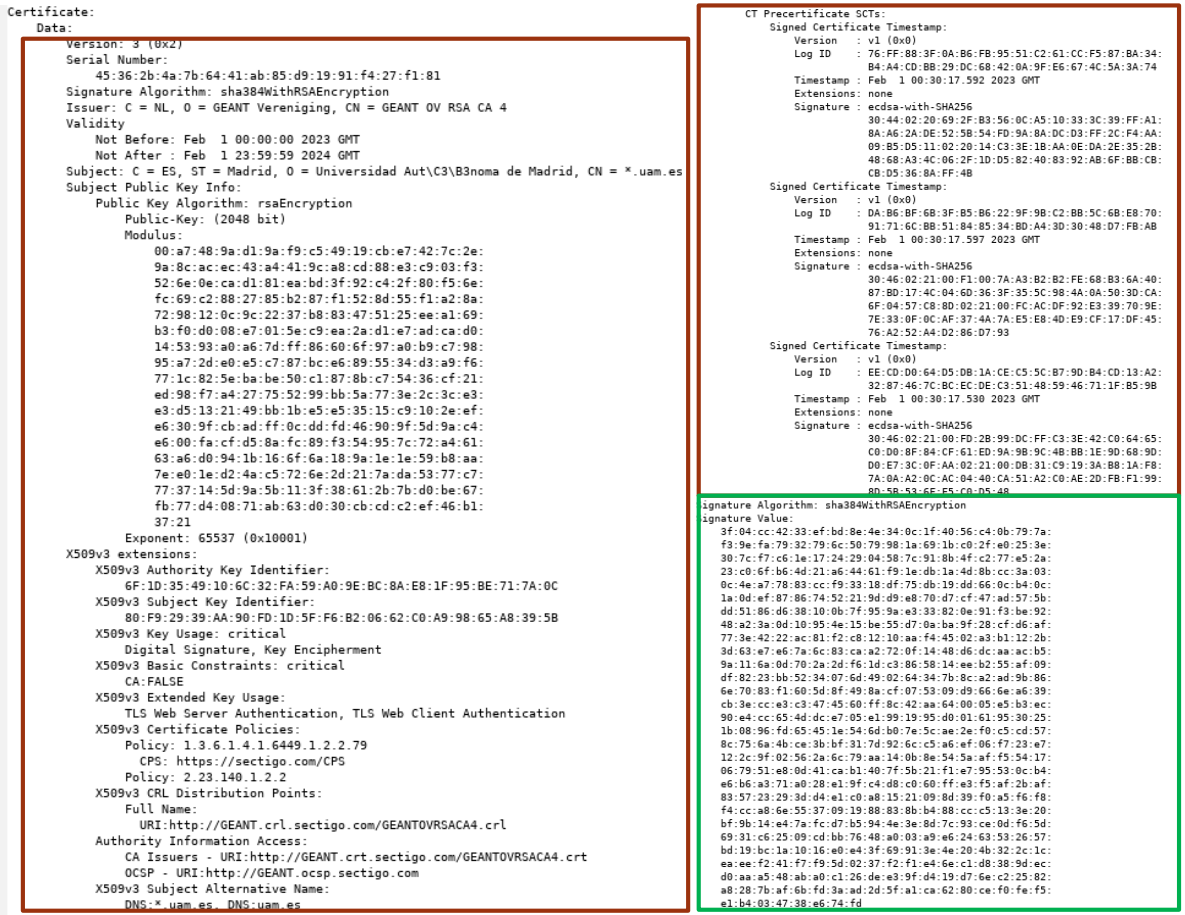
X.509 obsoleto, como ejemplo ilustrativo

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 7829 (0x1e95)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/Email=server-certs@thawte.com
        Validity
            Not Before: Jul  9 16:04:02 1998 GMT
            Not After : Jul  9 16:04:02 1999 GMT
        Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                 OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                    33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                    66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                    70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                    16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                    c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                    8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                    d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                    e8:35:1c:9e:27:52:7e:41:8f
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
        92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
        ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
        d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
        0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
        5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
        8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
        68:9f
```

Imagen extraida http://es.wikipedia.org/wiki/X.509

- Al final está la firma del certificado.
- Para poner la firma, la CA calcula un hash MD5 (en este caso) de la primera parte del certificado (la sección de Data: los datos del mismo más la clave pública).
- Se cifra ese hash con la clave privada, PR, de la CA.
- Si nos conectamos a www.freesoft.org y el sitio devuelve el certificado de la izq.
- Para validar este certificado, necesitamos el certificado de la CA (Thawte Server CA).
- Se toma la PU del certificado de la CA para decodificar la firma del primer certificado, obteniéndose un hash MD5.
- Este hash MD5 debe coincidir con el hash MD5 calculado sobre la primera parte del certificado.
- Si no se valida OK no se asegurara que el certificado de www.freesoft.org está vinculado con esa clave pública.

# Motivación: Servicios en Criptografía Pública, X.509



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            45:36:2b:4a:7b:64:41:ab:85:d9:19:91:f4:27:f1:81
        Signature Algorithm: sha384WithRSAEncryption
        Issuer: C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
        Validity
            Not Before: Feb  1 00:00:00 2023 GMT
            Not After : Feb  1 23:59:59 2024 GMT
        Subject: C = ES, ST = Madrid, O = Universidad Aut\C3\B3noma de Madrid, CN = *.uam.es
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a7:48:9a:d1:9a:f9:c5:49:19:cb:e7:42:7c:2e:
                    9a:8c:ac:ec:43:a4:41:9c:a8:cd:88:e3:c9:03:f3:
                    52:6e:0e:ca:d1:81:ea:bd:3f:92:c4:2f:80:f5:6e:
                    fc:69:c2:88:27:85:b2:87:f1:52:8d:55:f1:a2:8a:
                    72:98:12:0c:9c:22:37:b8:83:47:51:25:ee:a1:69:
                    b3:f0:d0:08:e7:01:5e:c9:ea:2a:d1:e7:ad:ca:d0:
                    14:53:93:a0:a6:7d:ff:86:60:6f:97:a0:b9:c7:98:
                    95:a7:2d:e0:e5:c7:87:bc:e6:89:55:34:d3:a9:f6:
                    77:1c:82:5e:ba:be:50:c1:87:8b:c7:54:36:cf:21:
                    ed:98:f7:a4:27:75:52:99:bb:5a:77:3e:2c:3c:e3:
                    e3:d5:13:21:49:bb:1b:e5:e5:35:15:c9:10:2e:ef:
                    e6:30:9f:cb:ad:ff:0c:dd:fd:46:90:9f:5d:9a:c4:
                    e6:00:fa:cf:d5:8a:fc:89:f3:54:95:7c:72:a4:61:
                    63:a6:d0:94:1b:16:6f:6a:18:9a:1e:1e:59:b8:aa:
                    7e:e0:1e:d2:4a:c5:72:6e:2d:21:7a:da:53:77:c7:
                    77:37:14:5d:9a:5b:11:3f:38:61:2b:7b:d0:be:67:
                    fb:77:d4:08:71:ab:63:d0:30:cb:cd:c2:ef:46:b1:
                    37:21
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                6F:1D:35:49:10:6C:32:FA:59:A0:9E:BC:8A:E8:1F:95:BE:71:7A:0C
            X509v3 Subject Key Identifier:
                80:F9:29:39:AA:90:FD:1D:5F:F6:B2:06:62:C0:A9:98:65:A8:39:5B
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.6449.1.2.2.79
                  CPS: https://sectigo.com/CPS
                Policy: 2.23.140.1.2.2
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://GEANT.crl.sectigo.com/GEANTOVRSACA4.crl
            Authority Information Access:
                CA Issuers - URI:http://GEANT.crt.sectigo.com/GEANTOVRSACA4.crt
                OCSP - URI:http://GEANT.ocsp.sectigo.com
            X509v3 Subject Alternative Name:
                DNS:*.uam.es, DNS:uam.es
        CT Precertificate SCTs:
            Signed Certificate Timestamp:
                Version   : v1 (0x0)
                Log ID    : 76:FF:88:3F:0A:B6:FB:95:51:C2:61:CC:F5:87:BA:34:
                            B4:A4:CD:BB:29:DC:68:42:0A:9F:E6:67:4C:5A:3A:74
                Timestamp : Feb  1 00:30:17.592 2023 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:44:02:20:69:2F:B3:56:0C:A5:10:33:3C:39:FF:A1:
                            8A:A6:2A:DE:52:5B:54:FD:9A:8A:DC:D3:FF:2C:F4:AA:
                            09:B5:D5:11:02:20:14:C3:3E:1B:AA:0E:DA:2E:35:2B:
                            48:68:A3:4C:06:2F:1D:D5:82:40:83:92:AB:6F:BB:CB:
                            CB:D5:36:8A:FF:4B
            Signed Certificate Timestamp:
                Version   : v1 (0x0)
                Log ID    : DA:B6:BF:6B:3F:B5:B6:22:9F:9B:C2:BB:5C:6B:E8:70:
                            91:71:6C:BB:51:84:85:34:BD:A4:3D:30:48:D7:FB:AB
                Timestamp : Feb  1 00:30:17.597 2023 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:46:02:21:00:F1:00:7A:A3:B2:B2:FE:68:B3:6A:40:
                            87:BD:17:4C:04:6D:36:3F:35:5C:98:4A:0A:50:3D:CA:
                            6F:04:57:C8:BD:02:21:00:FC:AC:DF:92:E3:39:70:9E:
                            7E:33:0F:0C:AF:37:4A:7A:E5:E8:4D:E9:CF:17:DF:45:
                            76:A2:52:A4:D2:86:D7:93
            Signed Certificate Timestamp:
                Version   : v1 (0x0)
                Log ID    : EE:CD:D0:64:D5:DB:1A:CE:C5:5C:B7:9D:B4:CD:13:A2:
                            32:87:46:7C:BC:EC:DE:C3:51:48:59:46:71:1F:B5:9B
                Timestamp : Feb  1 00:30:17.530 2023 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:46:02:21:00:FD:2B:99:DC:FF:C3:3E:42:C0:64:65:
                            C0:D0:8F:84:CF:61:ED:9A:9B:9C:4B:BB:1E:9D:68:9D:
                            D0:E7:3C:0F:AA:02:21:00:DB:31:C9:19:3A:B8:1A:F8:
                            7A:0A:A2:0C:AC:04:40:CA:51:A2:C0:AE:2D:FB:F1:99:
                            8D:5B:53:6F:E5:C0:D5:48
    Signature Algorithm: sha384WithRSAEncryption
    Signature Value:
        3f:04:cc:42:33:ef:bd:8e:4e:34:0c:1f:40:56:c4:0b:79:7a:
        f3:9e:fa:79:32:79:6c:50:79:98:1a:69:1b:c0:2f:e0:25:3e:
        30:7c:f7:c6:1e:17:24:29:04:58:7c:91:8b:4f:c2:77:e5:2a:
        23:c0:6f:b6:4d:21:a6:44:61:f9:1e:db:1a:4d:8b:cc:3a:03:
        0c:4e:a7:78:83:cc:f9:33:18:df:75:db:19:dd:66:0c:b4:0c:
        1a:0d:ef:87:86:74:52:21:9d:d9:e8:70:d7:cf:47:ad:57:5b:
        dd:51:86:d6:38:10:0b:7f:95:9a:e3:33:82:0e:91:f3:be:92:
        48:a2:3a:0d:10:95:4e:15:be:55:d7:0a:ba:9f:28:cf:d6:af:
        77:3e:42:22:ac:81:f2:c8:12:10:aa:f4:45:02:a3:b1:12:2b:
        3d:63:e7:e6:7a:6c:83:ca:a2:72:0f:14:48:d6:dc:aa:ac:b5:
        9a:11:6a:0d:70:2a:2d:f6:1d:c3:86:58:14:ee:b2:55:af:09:
        df:82:23:bb:52:34:07:6d:49:02:64:34:7b:8c:a2:ad:9b:86:
        6e:70:83:f1:60:5d:8f:49:8a:cf:07:53:09:d9:66:6e:a6:39:
        cb:3e:cc:e3:c3:47:45:60:ff:8c:42:aa:64:00:05:e5:b3:ec:
        90:e4:cc:65:4d:dc:e7:05:e1:99:19:95:d0:01:61:95:30:25:
        1b:08:96:fd:65:45:1e:54:6d:b0:7e:5c:ae:2e:f0:c5:cd:57:
        8c:75:6a:4b:ce:3b:bf:31:7d:92:6c:c5:a6:ef:06:f7:23:e7:
        12:2c:9f:02:56:2a:6c:79:aa:14:0b:8e:54:5a:af:f5:54:17:
        06:79:51:e8:0d:41:ca:b1:40:7f:5b:21:f1:e7:95:53:0c:b4:
        e6:b6:a3:71:a0:28:e1:9f:c4:d8:c0:60:ff:e3:f5:af:2b:af:
        83:57:23:29:3d:d4:e1:c0:a8:15:21:09:89:39:f0:a5:f6:f8:
        f4:cc:a8:6e:55:37:09:19:88:83:8b:b4:88:cc:c5:13:3e:20:
        bf:9b:14:e4:7a:fc:d7:b5:94:4e:3e:8d:7c:93:ce:0d:f6:5d:
        69:31:c6:25:09:cd:bb:76:48:a0:03:a9:e6:24:63:53:26:57:
        bd:19:bc:1a:10:16:e0:e4:3f:69:91:3e:4e:20:4b:32:2c:1c:
        ea:ee:f2:41:f7:f9:5d:02:37:f2:f1:e4:6e:c1:d8:38:9d:ec:
        d0:aa:a5:48:ab:a0:c1:26:de:e3:9f:d4:19:d7:6e:c2:25:82:
        a8:28:7b:af:6b:fd:3a:ad:2d:5f:a1:ca:62:80:ce:f0:fe:f5:
        e1:b4:03:47:38:e6:74:fd
```

- Este es el certificado que tiene la UAM, mucho más moderno.
- Para visualizar cualquier certificado lo descargas en formato en formato .crt o .pem y los transformas a texto con *openssl*:
  - openssl x509 -in _.uam.es.crt -text -noout > certificadoUAM
- Esencialmente la forma de funcionar es la misma.
- Para poner la firma, la CA (en este caso GEANT Vereniging) calcula un hash sha384 (en este caso) de la primera parte del certificado (la sección de Data: los datos del mismo más la clave pública).
- Posteriormente la CA cifra con RSA de módulo 2048 bits ese hash con su clave privada, PR.
- Cualquiera puede validar es certificado.

# Funciones MAC (*Message Authentication Code*): Propiedades

1. Una función MAC es pública (no tiene por qué tener inversa).
2. Existe una clave privada k ($MAC_k$). La clave secreta k, solo la conocen el remitente y destinatario, pero no los atacantes.
3. El tamaño del mensaje puede ser tan grande como queramos, pero la salida es fija.
4. La salida de denomina "*digest*" y es fija de un determinado tamaño de bits n: $lb(MAC_k(M))=n$ (lb=longitud de bits).
5. Es computacionalmente muy difícil encontrar un mensaje M' tal que $MAC_k(M')=MAC_k(M)$.
6. La probabilidad de que dos mensajes $M_1$ y $M_2$ diferentes tengan $MAC_k(M_1)=MAC_k(M_2)$ es $P = \frac{1}{2^n}$.

# Funciones MAC: Ejemplos de Implementación

- Esta implementación es un tipo de MAC llamado CBC-MAC.
- Los estándares que definen este algoritmo son:
  - "FIPS PUB 113 *Computer Data Authentication*". Archived from the original on 2011-09-27. Retrieved 2010-10-10. (https://csrc.nist.gov/publications/detail/fips/113/archive/1985-05-30). Esta ya obsoleto.
  - El algoritmo CBC-MAC es equivalente al algoritmo 1 MAC del ISO/IEC 9797-1.
- Hay otros tipos de implementación para la función MAC, como:
  - HMAC (ejemplos HMAC-SHA-256 o HMAC-SHA3-512).
  - UMAC (http://fastcrypto.org/umac/).

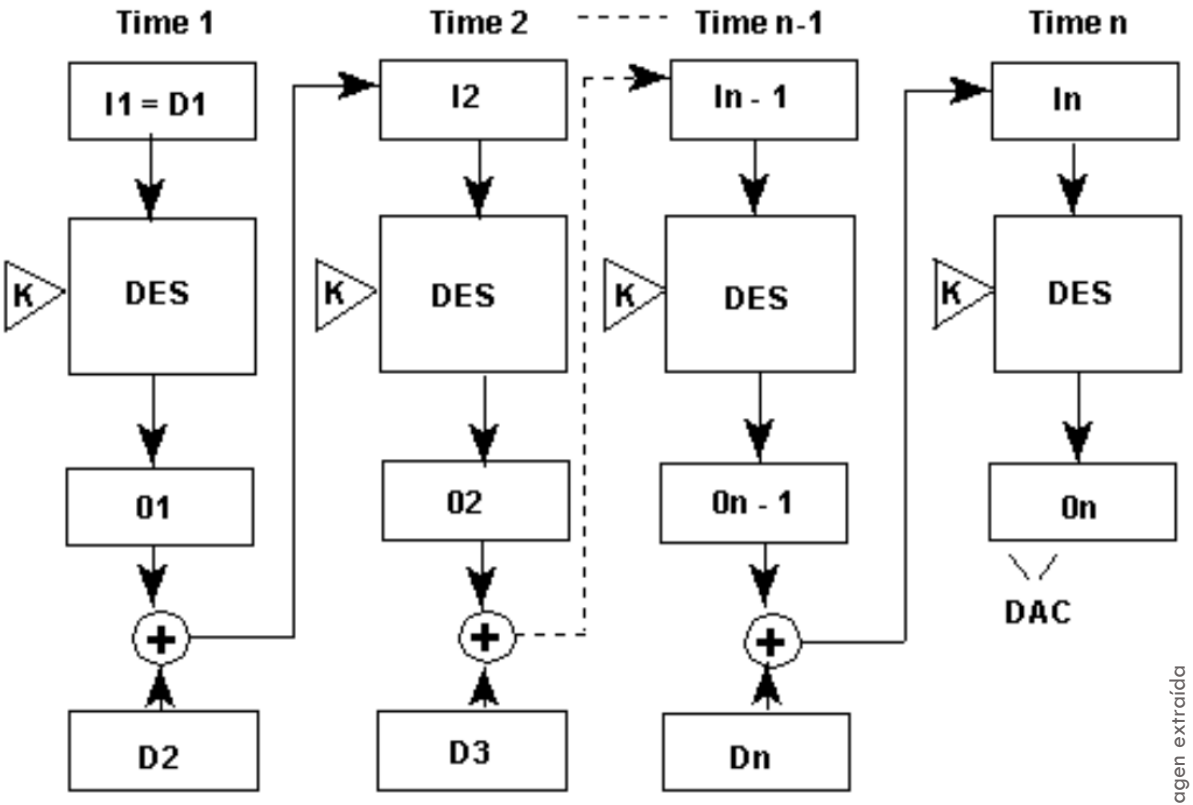
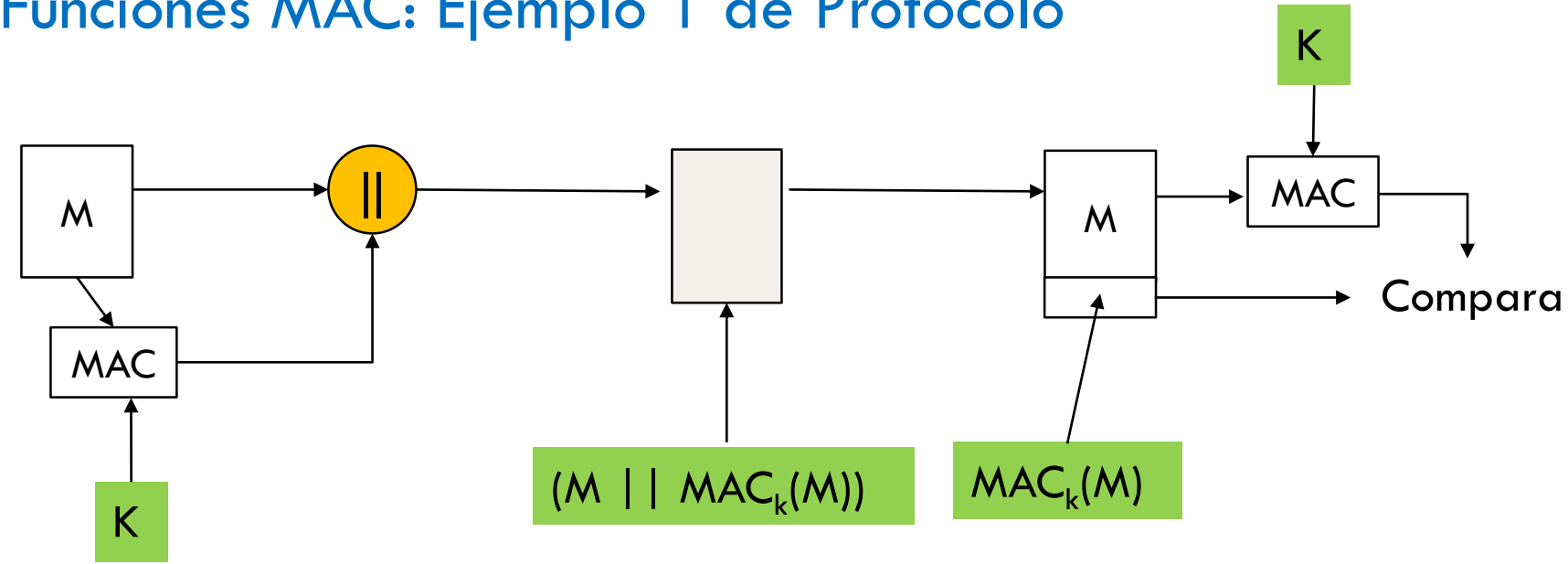


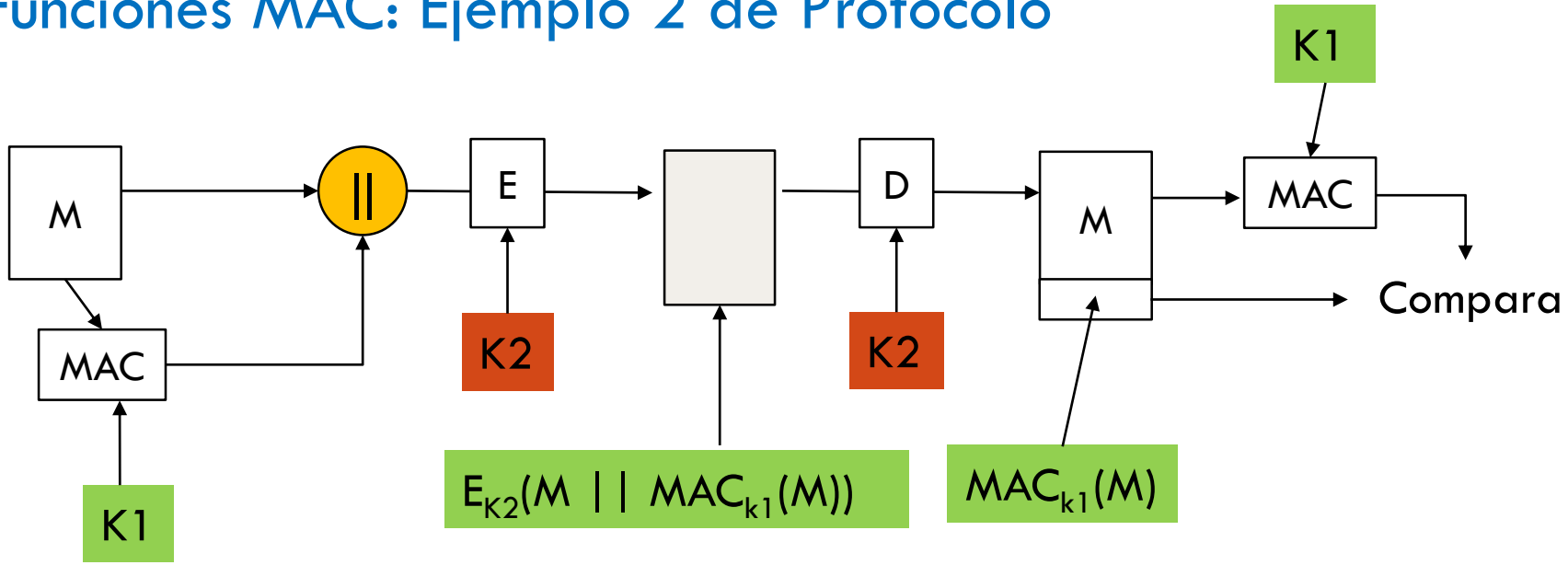

Imagen extraída https://web.archive.org/web/20110927022556/http://www.itl.nist.gov/fipspubs/fip113.htm

# Funciones MAC: Ejemplo 1 de Protocolo



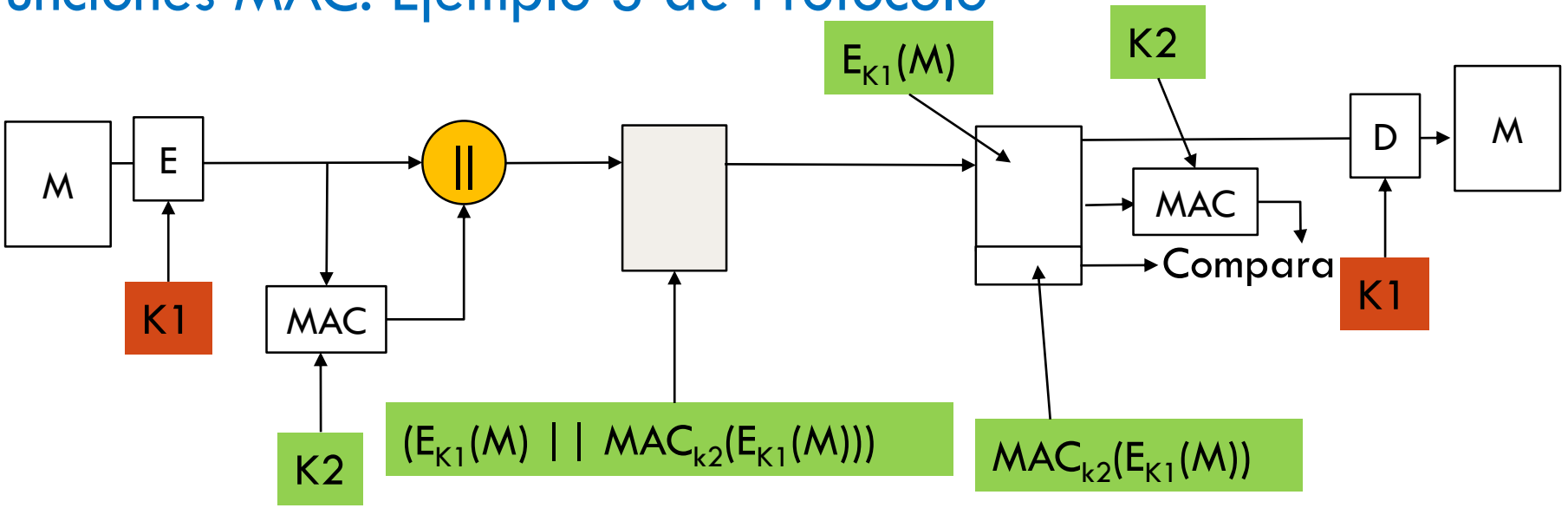

- Confidencialidad: NO (ya que el mensaje va en claro).
- Autenticación: SI ligada al texto plano (solo A y B tienen la clave K).
- Firma: NO.

# Funciones MAC: Ejemplo 2 de Protocolo




- Confidencialidad: SI (solo A y B tienen la clave K2).
- Autenticación: SI ligada al texto plano (solo A y B tienen la clave K1).
- Firma: NO.

# Funciones MAC: Ejemplo 3 de Protocolo




- Confidencialidad: SI (solo A y B tienen la clave K1).
- Autenticación: SI ligada al texto cifrado (solo A y B tienen la clave K2).
- Firma: NO.

# Funciones Hash: Propiedades

1. Una función Hash es pública.
2. Una función Hash es una función de una sola vía (*One Way Function*). H(x) es relativamente fácil de calcular para cualquier x dado, siendo posible hacer ambas implementaciones de hardware y software. Se dice que tiene **resistencia a la primera pre-imagen** (para un valor hash h=H(x), decimos que x es la imagen inversa de h, o simplemente la pre-imagen).
3. Una función hash se dice que es determinista ya que dada una cadena de entrada siempre devuelve el mismo valor hash.
4. El tamaño del mensaje puede ser tan grande como queramos, pero la salida es fija.
5. La salida de denomina "*digest*" y es fija de un determinado tamaño de bits n: lb(Hash(M))=n (lb=longitud de bits).
6. Cumple la propiedad de *Week Collition Free*: Si tenemos $m_1=H(M_1)$, no es computacionalmente posible encontrar un $M_2$ tal que $H(M_1)=H(M_2)$ (conocido como **resistente a la segunda pre-imagen**).
7. Cumple la propiedad de *Strong Collition Free*: No es computacionalmente posible que existan $M_1$ y $M_2$ tal que $H(M_1)=H(M_2)$ (conocido también como **resistencia a colisiones**).
8. Pseudoaleatoriedad: La salida de H cumple las pruebas estándares para pseudoaleatoriedad (muy relacionado con el efecto avalancha de la función Hash).

# Funciones Hash: Propiedades (Avalancha)

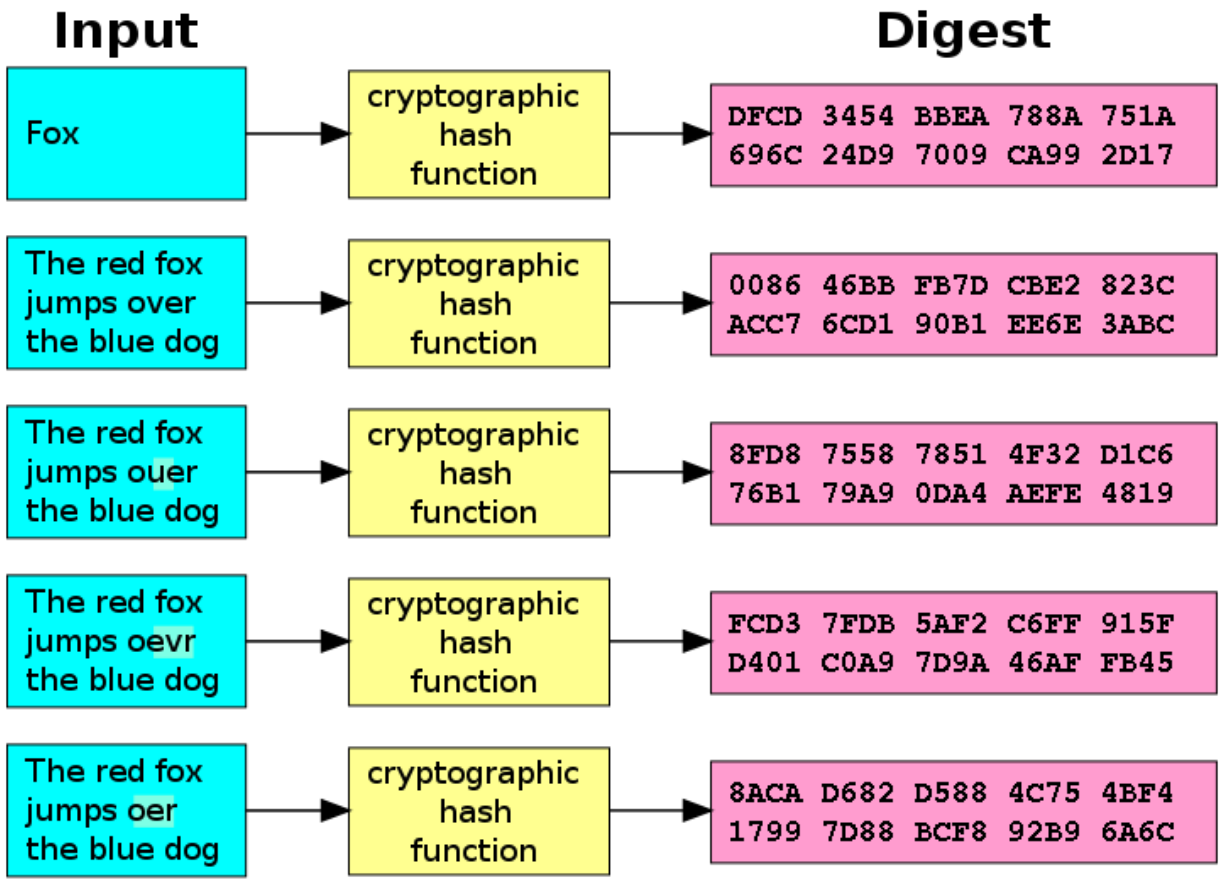



Imagen extraída de http://en.wikipedia.org/wiki/File:Cryptographic_Hash_Function.svg

- Hay que tener en cuenta que en una función un pequeño cambio en la entrada de la función Hash produce un "*digest*" completamente diferente.
- Así se desea que no haya correlación entre los bits de entrada y los bits de salida.
- Es decir, una modificación minúscula (por ejemplo un bit) en la entrada ocasiona cambios en el valor hash comparables a un cambio de cualquier otro tipo en la entrada.
- Por tanto cualquier cambio en el mensaje original idealmente hace cualquier bit del valor resumen resultante cambie con probabilidad de aproximadamente 0,5.
- Cuando esto sucede se dice que se produce un efecto avalancha deseado en la función Hash criptográfica.

# Funciones Hash: Ejemplos de Implementación

- Suelen seguir una estructura General: Merkle–Damgård.
- El bloque final también incluye el valor de la longitud total de la entrada a la función hash. La inclusión de la longitud hace que el trabajo del oponente más difícil.
- El algoritmo implica el uso repetido de una función de compresión, f, que tiene dos entradas:
- una entrada de la etapa anterior, variable de encadenamiento,
- y otra entrada el bloque del mensaje,
- y produce una salida para la siguiente función, que es la variable de encadenamiento para el siguiente.
- Al comienzo de hash, hay un valor inicial (IV) que se especifica como parte del algoritmo. Al final el valor de la variable de encadenamiento es el valor hash.
- Ejemplos: MD5, SHA-1 and SHA-2.

- También funciones hash basados en el uso de una técnica de encadenamiento de bloques de cifrado, pero sin la clave secreta (ya no se utilizan mucho).
- Rabin (Rabin, M. "Digitalized Signatures." Foundations of Secure Computation, DeMillo, R.; Dobkin, D.; Jones, A.; and Lipton, R., eds. New York: Academic Press, 1978) divide un mensaje M en bloques de tamaño fijo Mi, y usa un sistema de cifrado simétrico como DES para calcular el código hash.
- Usando H0 = 0 y cero de relleno de bloque final.
- Calcular: Hi = E (Mi Hi-1).
- Usa el resultado del bloque final como el valor hash.
- De manera similar a CBC, pero sin una clave.

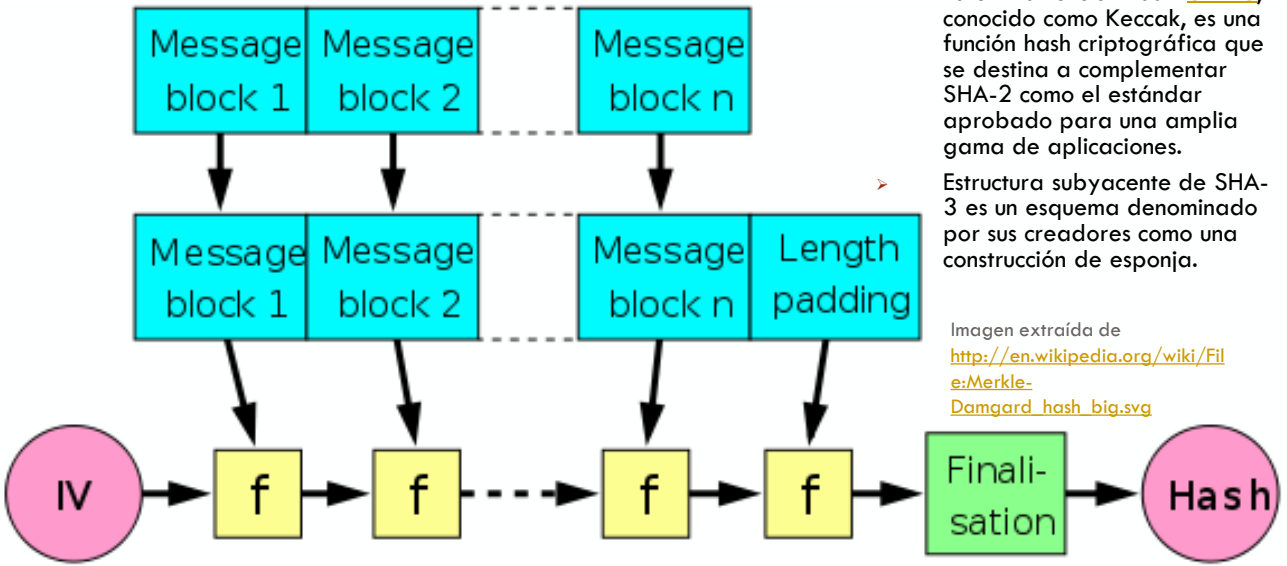
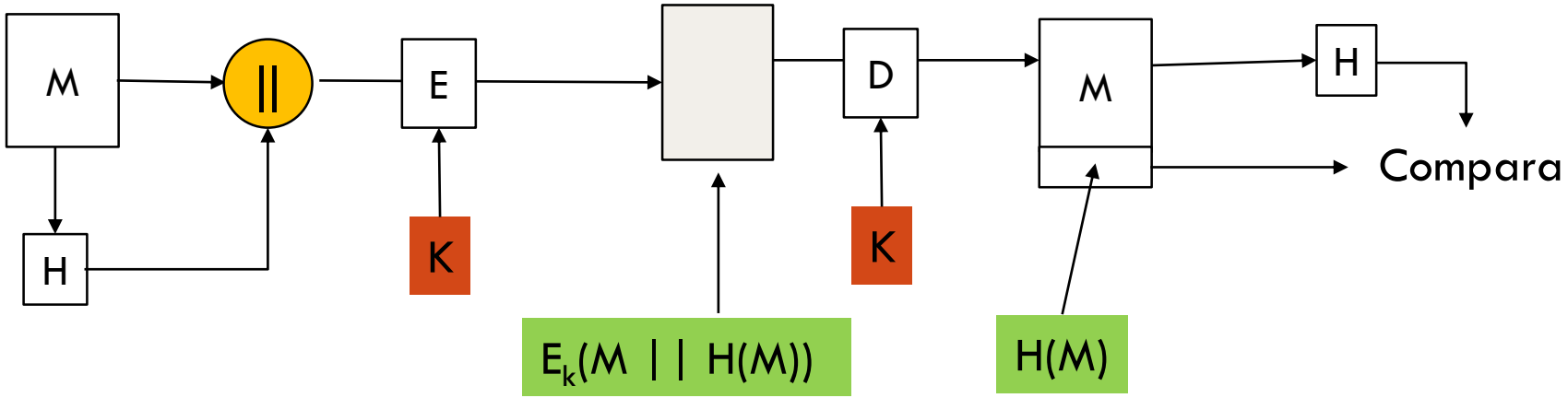



Imagen extraída de http://en.wikipedia.org/wiki/File:Merkle-Damgard_hash_big.svg

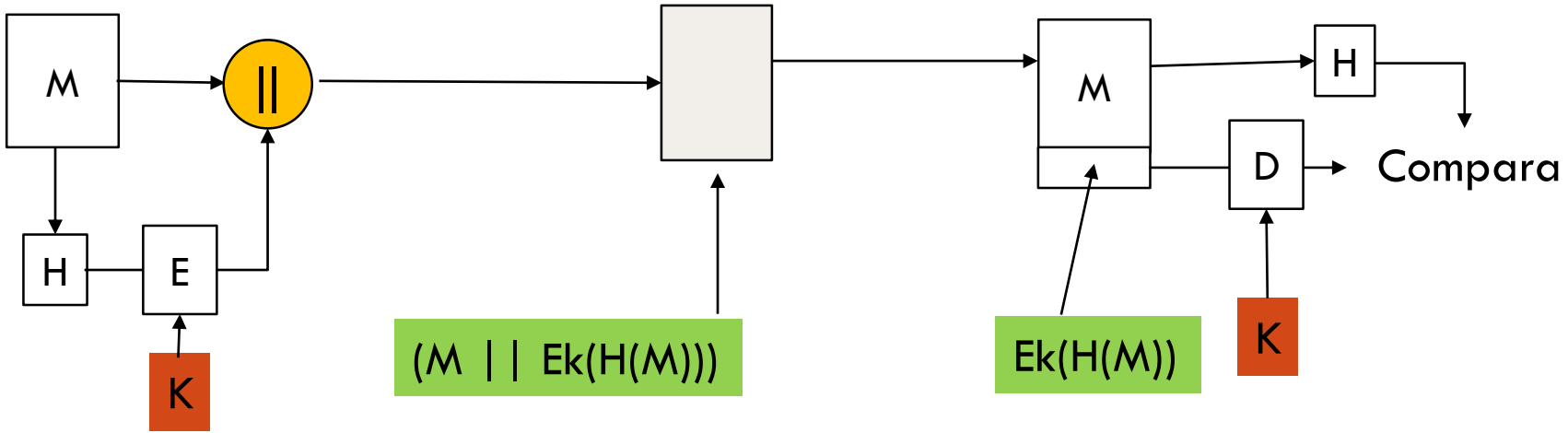
- NIST anunció en 2007 un concurso para la función hash **SHA-3** del NIST para la próxima generación.
- El diseño ganador fue anunciado por el NIST en octubre de 2012.
- En agosto de 2015 se anuncia como el nuevo estándar de funciones hash.
- La última función hash: SHA-3, conocido como Keccak, es una función hash criptográfica que se destina a complementar SHA-2 como el estándar aprobado para una amplia gama de aplicaciones.
- Estructura subyacente de SHA-3 es un esquema denominado por sus creadores como una construcción de esponja.

# Funciones Hash: Ejemplo 1 de Protocolo



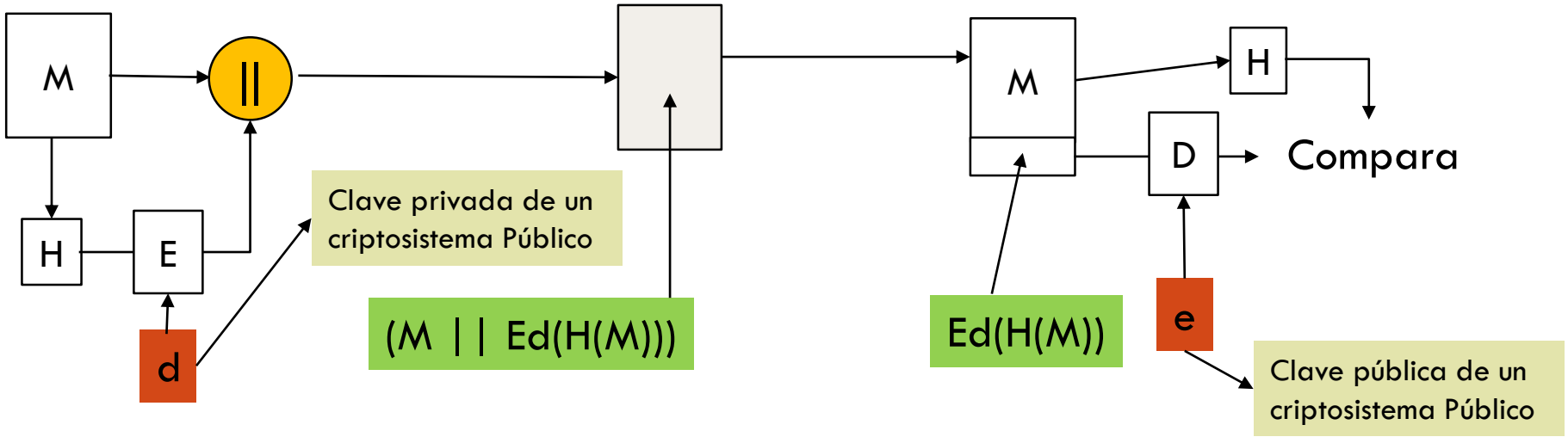

- Confidencialidad: SI (solo A y B tienen la clave K).
- Autenticación: SI (solo A y B tienen la clave K). Darse cuenta que está autentificado aunque se dé el caso $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$, por las propiedades de las funciones Hash criptográficas:
  - Si no estuviese la función Hash, y todos los mensajes tuviesen sentido ($\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$), entonces cualquier atacante en canal seguro podría cambiar el mensaje cifrado, y el receptor no se daría cuenta ya que todos los mensajes tienen sentido. Pero la función Hash elimina esta posibilidad.
- Firma: NO (la clave es simétrica y por tanto pertenece al emisor y al receptor).

# Funciones Hash: Ejemplo 2 de Protocolo



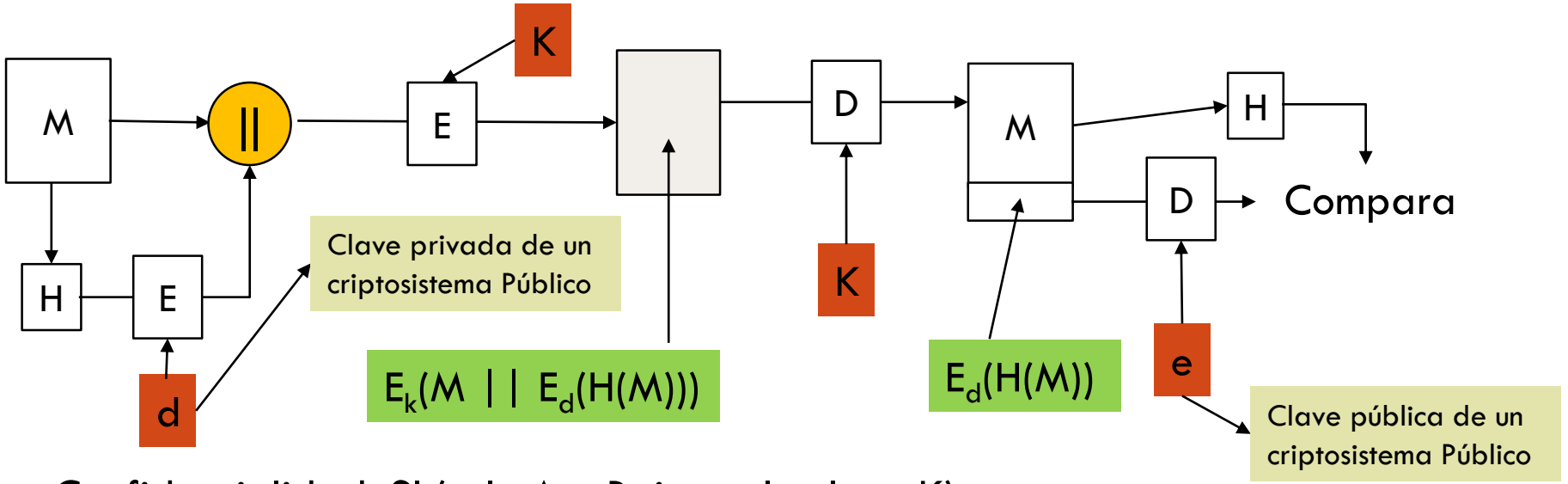

- Confidencialidad: NO (El mensaje va libre).
- Autenticación: SI (solo A y B tienen la clave K). Darse cuenta que está autentificado aunque se dé el caso $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$, por las propiedades de las funciones Hash criptográficas (en este caso resistencia a la segunda pre-imagen).
- Firma: NO (la clave es simétrica y por tanto pertenece al emisor y al receptor).

# Funciones Hash: Ejemplo 3 de Protocolo



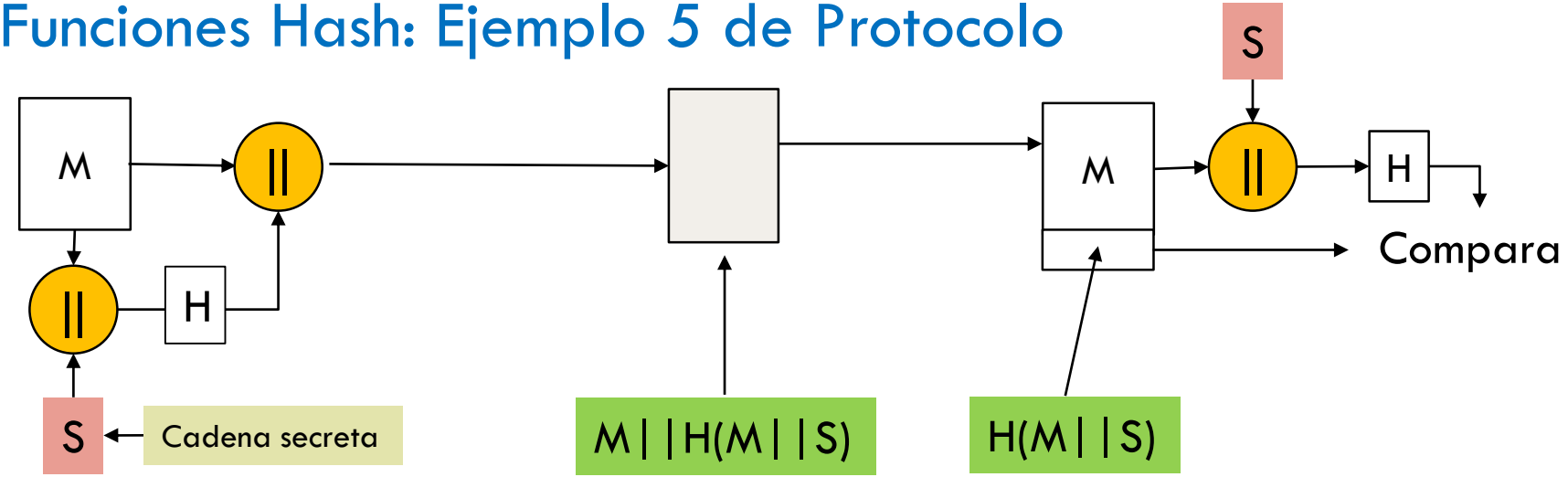

- Confidencialidad: NO (El mensaje va libre).
- Autenticación: SI (solo A conoce d). Darse cuenta que está autentificado aunque se dé el caso $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$, por las propiedades de las funciones Hash criptográficas (en este caso resistencia a la segunda pre-imagen).
- Firma: SI (por favor notar que lo único que se firma es el Hash, ¿Por qué?).

# Funciones Hash: Ejemplo 4 de Protocolo



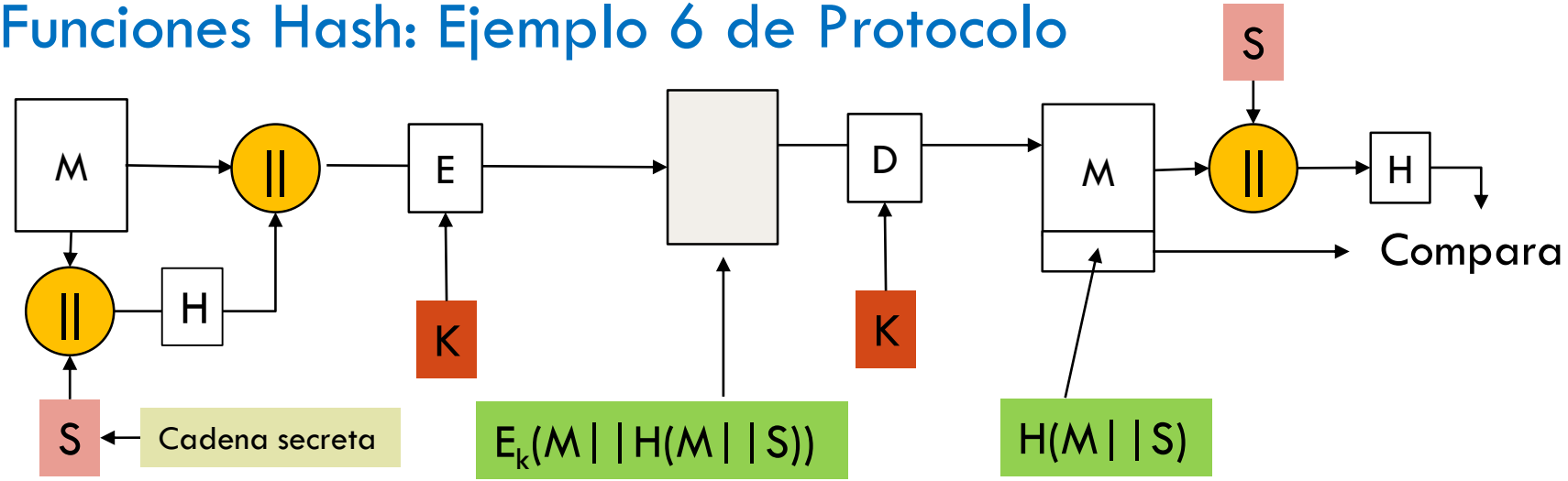

- Confidencialidad: SI (solo A y B tienen la clave K).
- Autenticación: SI (solo A conoce d). Darse cuenta que está autentificado aunque se dé el caso $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$, por las propiedades de las funciones Hash criptográficas (cualquier modificación en el canal inseguro la comparación final no funciona).
- Firma: SI (por favor notar que lo único que se firma es el Hash, ¿Por qué?).

# Funciones Hash: Ejemplo 5 de Protocolo




- Confidencialidad: NO (El mensaje va libre).
- Autenticación: SI (solo A y B tienen la cadena secreta S). Darse cuenta que está autentificado aunque se dé el caso $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$, por las propiedades de las funciones Hash criptográficas (en este caso resistencia a la segunda pre-imagen no se puede encontrar otro M' que tenga el mismo Hash).
- Firma: NO (la clave es simétrica y por tanto pertenece al emisor y al receptor).

# Funciones Hash: Ejemplo 6 de Protocolo




- Confidencialidad: SI (solo A y B tienen la clave K).
- Autenticación: SI (solo A y B tienen la clave S, y nadie la puede conseguir por la resistencia a la primera pre-imagen de la función Hash). Darse cuenta que está autentificado aunque se dé el caso $\frac{\#\ de\ x\ con\ sentido}{\#\ de\ x\ totales} \approx 1$, por las propiedades de las funciones Hash criptográficas (cualquier modificación en el canal inseguro la comparación final no funciona).
- Firma: NO (la clave es simétrica y por tanto pertenece al emisor y al receptor).

# Aspectos teóricos de las funciones Hash: *Week Collition Free*

- Si tenemos $m_1=H(M_1)$, no es computacionalmente posible encontrar un $M_2$ con $m_2=H(M_2)$, tal que $H(M_1)=H(M_2)$ (conocido como **resistente a la segunda pre-imagen**).
- ¿Pero cómo de difícil es?
- Supongamos que la función Hash tiene $n$ bits de salida, por lo tanto el número posible de salidas es $N = 2^n$.
- Por tanto $P(m_2 = m_1) = \frac{1}{2^n} = \frac{1}{N}$.
- Cuál es la probabilidad de que escogiendo un mensaje al azar no encuentre una colisión:
  - $P(m_2 \neq m_1) = 1 - \frac{1}{N}$.
- Cuál es la probabilidad de que escogiendo $t$ mensajes al azar no encuentre una colisión con uno dado:
  - $P(m_2 \neq m_1, t\ veces\ m_2\ distintos) = \left(1 - \frac{1}{N}\right)^t$.

# Aspectos teóricos de las funciones Hash: *Week Collition Free*

- *Inicio de paréntesis*: siempre cumple que $(1-x) \leq e^{-x}\ \forall\ x \geq 0$, y cuando $x$ es muy pequeño tenemos que $(1-x) \approx e^{-x}$: *Fin de paréntesis.*
- Entonces $P(m_2 \neq m_1, t\ veces\ m_2\ distintos) = \left(1-\frac{1}{N}\right)^t \approx e^{-\frac{t}{N}}$. Por lo tanto la probabilidad de encontrar un colisión ($m_2$) a un Hash determinado $m_1$ viene determinado por:
  - $P(m_2 = m_1) = 1 - e^{-\frac{t}{N}}$.
- Por tanto cuantos $m_2$ deberíamos generar para encontrar una colisión con una probabilidad determinada (despejamos t):
  - $t = NLn(1 - P(m_2 = m_1))$.
- Si queremos $P(m_2 = m_1) = 0{,}5$ sustituyendo arriba tenemos $t = 0.69N$, lo cual nos dice que $t \approx \frac{N}{2} = 2^{n-1}$.
- Es decir la fortaleza de la segunda pre-imagen es del orden de $2^{n-1}$: para encontrar una colisión a un Hash determinado debo calcular un total de al menos $2^{n-1}$ hashes para encontrar dicha colisión con una probabilidad del 50%.

# Aspectos teóricos de las funciones Hash: *Strong Collition Free*

- No es computacionalmente posible que existan dos mensajes escogidos al azar $M_1$ y $M_2$ tal que $H(M_1)=H(M_2)$ (conocido también como **resistencia a colisiones**).
- ¿Pero cómo de difícil es?
- Para calcularlo nos vamos a basar en el conocido problema del cumpleaños.
- ¿Cuál es el mínimo número de personas que tiene que haber en un recinto para que tengamos una probabilidad del 50% de encontrar dos personas que cumplen años el mismo día (una colisión de cumpleaños)?
- Definimos la probabilidad $P(N;t)$, como la probabilidad de encontrar un duplicado en $t$ elementos, estando cada uno de estos elementos en $N$posibles estados con igual probabilidad.
- Los $t$ elementos son el número de personas que cada una puede tener su cumpleaños en una de las $365$ posibilidades, es decir la probabilidad que buscamos es: $P(365;t)$.

# Aspectos teóricos de las funciones Hash: *Strong Collition Free*

- Primero calculemos la probabilidad de no encontrar duplicados $Q(365; t)$.
- Tenemos que tener en cuenta que para $t > 365$ es imposible calcular $Q$ (siempre hay duplicados). Por tanto suponemos $t \leq 365$.
- Ahora consideramos el número de diferentes maneras que podamos tener $t$ elementos sin duplicados:
  - Si $t = 2$, podemos elegir $365$ maneras para el primer elemento y $364$ para el segundo elemento, por tanto un total de $365 * 364$ maneras de escoger los elementos para que no exista un duplicado.
  - Si $t = 3$, un total de $365 * 364 * 363$ maneras de escoger los elementos para que no exista un duplicado.
  - Si $t = 4$, un total de $365 * 364 * 363 * 362$ maneras de escoger los elementos para que no exista un duplicado.
  - …
  - Si $t = t$, un total de $365 * 364 * \cdots * (365 - t + 1) = \frac{365!}{(365-t)!}$ maneras de escoger los elementos para que no exista un duplicado.
  - El número de casos totales ya no hay restricciones: $\underbrace{365 * 365 * \cdots * 365}_{t \text{ veces}} = 365^t$.

# Aspectos teóricos de las funciones Hash: *Strong Collition Free*

- Así la probabilidad de no encontrar duplicados $Q(365;t)$

$$= \frac{\#\ casos\ favorables}{\#\ casos\ totales} = \frac{\frac{365!}{(365-t)!}}{365^t} = \frac{365!}{(365-t)!*365^t}.$$

- Pero recordemos que estamos interesados en encontrar duplicados:
  - $P(365;t) = 1 - Q(365;t) = 1 - \frac{365!}{(365-t)!*365^t}.$
- Esto se puede generalizar para cualquier $N$, para encontrar los duplicados en el caso general:
  - $P(N;t) = 1 - \frac{N!}{(N-t)!*N^t}.$

Ejemplos: $P(365;23) = 0{,}5$; $P(365;40) = 0{,}9$; $P(365;70) = 0{,}9999997$.

- Recordar que $P(N;t)$ es la probabilidad de encontrar un duplicado en $t$ elementos, estando cada uno de estos elementos en $N$ posibles estados con igual probabilidad.

# Aspectos teóricos de las funciones Hash: *Strong Collition Free*

- ¿Cómo utilizamos esto para las funciones Hash?
- Realmente la expresión $P(N;t) = 1 - \frac{N!}{(N-t)!*N^t}$ calcula la probabilidad de que colisionen el Hash de dos mensajes escogidos al azar para $t = 2$, cuando el "*digest*" es de $n$ bits (i.e. el número de posibles salidas del Hash es $N = 2^n$).
- La pregunta es cuál es el valor de la $t$ para que tengamos para que tengamos una cierta probabilidad aceptable de encontrar una colisión, por ejemplo $P(N;t) = 0{,}5$.
- Lo único que tenemos que hacer es despejar la $t$, pero al haber operaciones de factorial vamos a transformar esa esa expresión poquito y poder despejar $t$.

# Aspectos teóricos de las funciones Hash: *Strong Collition Free*

- $P(N;t) = 1 - \frac{N!}{(N-t)!*N^t} = 1 - \frac{N*(N-1)*\cdots*(N-t+1)*(N-t)*\cdots*1}{(N-t)!*N^t}$ ($t$ factores)

  $= 1 - \frac{N*(N-1)*\cdots*(N-t+1)*(N-t)!}{(N-t)!*N^t} = 1 - \frac{N*(N-1)*\cdots*(N-t+1)}{N^t} = 1$

  $- \left(\frac{N}{N} * \frac{N-1}{N} * \cdots * \frac{N-(t-1)}{N}\right).$

- *Inicio de paréntesis*: siempre cumple que $(1-x) \leq e^{-x}\ \forall\ x \geq 0$, y cuando $x$ es muy pequeño tenemos que $(1-x) \approx e^{-x}$: *Fin de paréntesis.*

- $P(N;t) = 1 - \left(\frac{N}{N} * \frac{N-1}{N} * \cdots * \frac{N-(t-1)}{N}\right) = 1 - \left(e^{-\frac{1}{N}} * e^{-\frac{2}{N}} * \cdots\right.$

  ($\frac{N}{N}$: $1$; $\frac{N-1}{N}$: $1-\frac{1}{N}$; $\frac{N-(t-1)}{N}$: $1-\frac{(t-1)}{N}$)

# Aspectos teóricos de las funciones Hash: *Strong Collition Free*

- *Inicio de paréntesis*: suma de una serie geométrica $\sum_{i=1}^{t-1} i = \frac{t(t-1)}{2}$: *Fin de paréntesis.*
- $P(N;t) = 1 - e^{-\frac{t(t-1)}{2N}}$.
- Ahora es muy fácil despejar $t$, sacando logaritmos naturales:
- $\frac{t(t-1)}{2} = -NLn\left(1 - P(N;t)\right) \Rightarrow$

$$t(t-1) = -2NLn(1 - P(N;t)) \Rightarrow$$

$t >> 1$

$$t^2 = -2NLn(1 - P(N;t)) \Rightarrow$$

$$t = \sqrt{-2NLn(1 - P(N;t))} \Rightarrow$$

Si $P(N;t) = 0{,}5 \Rightarrow t = 1{,}18 * \sqrt{N} \Rightarrow$ $\boxed{t \approx \sqrt{N} = 2^{\frac{n}{2}}.}$

Es decir la fortaleza de las colisiones es del orden de $\sqrt{N} = 2^{\frac{n}{2}}$: para encontrar una colisión en el Hash ($n$ bits de salida) de dos mensajes escogidos al azar debo escoger un total de al menos $\sqrt{N}$ parejas de dos mensajes para encontrar dicha colisión .

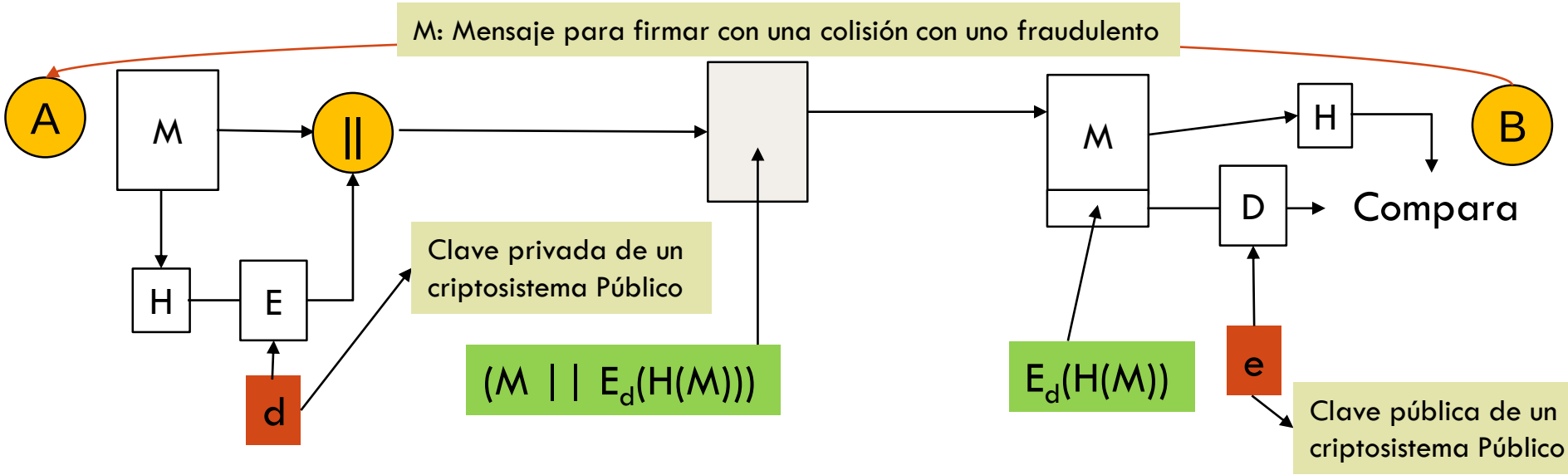
# Funciones Criptográficas HASH: Ataque del cumpleaños

- Al igual que con los algoritmos de cifrado, hay dos categorías de ataques: ataques de fuerza bruta y criptoanálisis.
- Un ataque de fuerza bruta no depende en el algoritmo específico, pero sólo depende de la longitud de bits de la salida del hash.
- Un ataque por criptoanálisis, por el contrario, es un ataque basado en las debilidades del algoritmo criptográfico.
- Por ejemplo el **Ataque de cumpleaños** explota la paradoja del cumpleaños: la posibilidad de que en un grupo de personas dos compartirán el mismo cumpleaños no es tan pequeña (sólo se necesitan 23 personas para una Pr> 0.5).
- Se puede generalizar el problema para buscar coincidente de dos conjuntos, y se necesitan solo probar $2^{n/2}$ para conseguir una colisión en un hash de n bits.
- Yuval (Gideon Yuval - How to Swindle Rabin Cryptologia 3:187-189, 1979) propuso la estrategia mostrada a explotar la paradoja del cumpleaños en un ataque resistente colisión. Tener en cuenta que la creación de muchos mensaje de variantes es relativamente fácil, ya sea por reformulación o simplemente variando la cantidad de espacios en blanco en el mensaje. Todo lo cual indica que se necesitan mayores salidas de MACs / Hashes.

# Funciones Criptográficas HASH: Ataque del cumpleaños

- Una función de hash que satisface todas las propiedades excepto "*Strong Collition Free*" se denomina función de Hash débil. Si la cumple se conoce como una función de hash fuerte.
- Una función hash fuerte protege de un ataque en el que una de las partes genera un mensaje para que otra persona lo firme (ataque del cumpleaños).
  - Supóngase que Bernardo escribe un mensaje de reconocimiento de Alicia de una deuda (es decir que Alicia le debe a Bernardo una cierta cantidad), y lo envía a Alicia para que lo firme (notar que solo firma el hash).
  - Pero Bernardo puede encontrar dos mensajes con el mismo hash, uno de los cuales requiere que Alicia pague una pequeña cantidad (la deuda verdadera) y el otro es un mensaje que reconoce una gran deuda.
  - Alicia firma el primer mensaje (pequeña deuda) y Alicia ya está tranquila, pero Bernardo es capaz de afirmar que el segundo mensaje de la deuda grande es auténtico, ya que había encontrado una colisión.

# Funciones Criptográficas HASH: Ataque del cumpleaños




- Se puede utilizar este protocolo que tiene firma, pero si el Hash es débil puede darse el ataque del cumpleaños.
- Bernardo genera dos conjuntos de mensajes: uno sobre la deuda real y otro fraudulento sobre una deuda mayor.
- Hace variaciones de los conjuntos de mensajes y busca una colisión que es factible por que $t \approx \sqrt{N} = 2^{\frac{n}{2}}$.
- Una vez que encuentra la colisión le manda el mensaje para que lo firme Alicia.
- Pero lo que no sabe Alicia es que Bernardo tiene calculada una colisión con un mensaje que tiene un deuda mayor.

# Funciones Criptográficas HASH: Ataque del cumpleaños

- Ejemplo de cómo crear variaciones de un mensaje sin alterar el contenido.
- En este simple ejemplo y seleccionando entre una de las dos opciones que se presentan, se puede crear hasta $2^{24}$ mensajes distintos.
- De esta manera se generan dos documentos de deudas: una con la cantidad real, y otra con una deuda mucho mayor.
- Se generan variaciones de los dos documentos y se busca la colisión.
- Se da a firmar el documento de la deuda real, y más tarde se demanda la deuda mayor.

```
{Estimado|Querido} cliente,

{Nos ponemos en contacto con usted|Le escribimos}
para {anunciarle|informarle} sobre la {charla|conferencia}
que se {realizará|llevará a cabo} el día 1 de enero de 2007,
a las 12 horas en {nuestro auditórium|nuestras premisas},
{que brindará|a cargo de} un {reconocido|prestigioso}
{autor|escritor} de {varios|numerosos} {libros|documentos}
sobre ciencia y tecnología.

{La charla|El encuentro} {consistirá en|comprenderá} los
siguientes {tópicos|contenidos}: {Internet, |Web 2.0, }
{E-learning y VoIP|VoIP y E-learning}.

{Esta|La presente} invitación es {sólo|únicamente} para
nuestros {más exclusivos clientes|clientes más exclusivos} y
es por {ello|esta razón}, que {esperamos|deseamos} contar con
su presencia.

Sin {más|otro particular}, {le saludamos|nos despedimos de
usted} {atentamente|cordialmente}.
```

Imagen extraída de https://es.wikipedia.org/wiki/Ataque_de_cumplea%C3%B1os