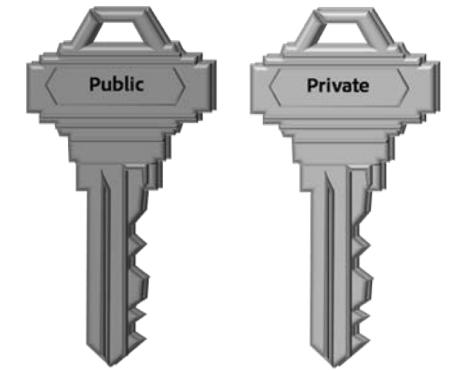
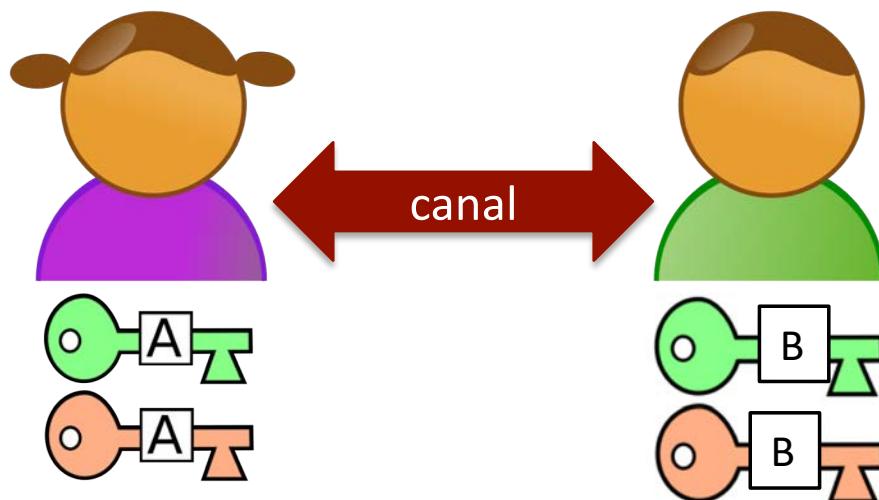


Algoritmos asimétricos (o de clave pública)



Introducción

- El concepto de criptografía de clave pública (o asimétrica) fue inventado en 1976 por *Diffie y Hellman*, e independientemente por *Merkle*, para dar solución a algunos de los problemas de los criptosistemas simétricos



- La asimetría reside en que, **las claves K y K^* son distintas**, al contrario de lo que ocurría en el caso simétrico

Introducción

- Las claves se utilizan por pares, de tal forma que cada usuario U posee dos claves:
 - una **clave pública**, conocida por todos los usuarios
 - una **clave privada**, conocida sólo por U
- Una clave se usa para cifrar, y la otra para descifrar
- Si se cifra un mensaje con una de las claves, esa misma no servirá para descifrar, sino que necesariamente habrá que usar la otra
- Existen tres funcionalidades básicas con cripto. asimétrica



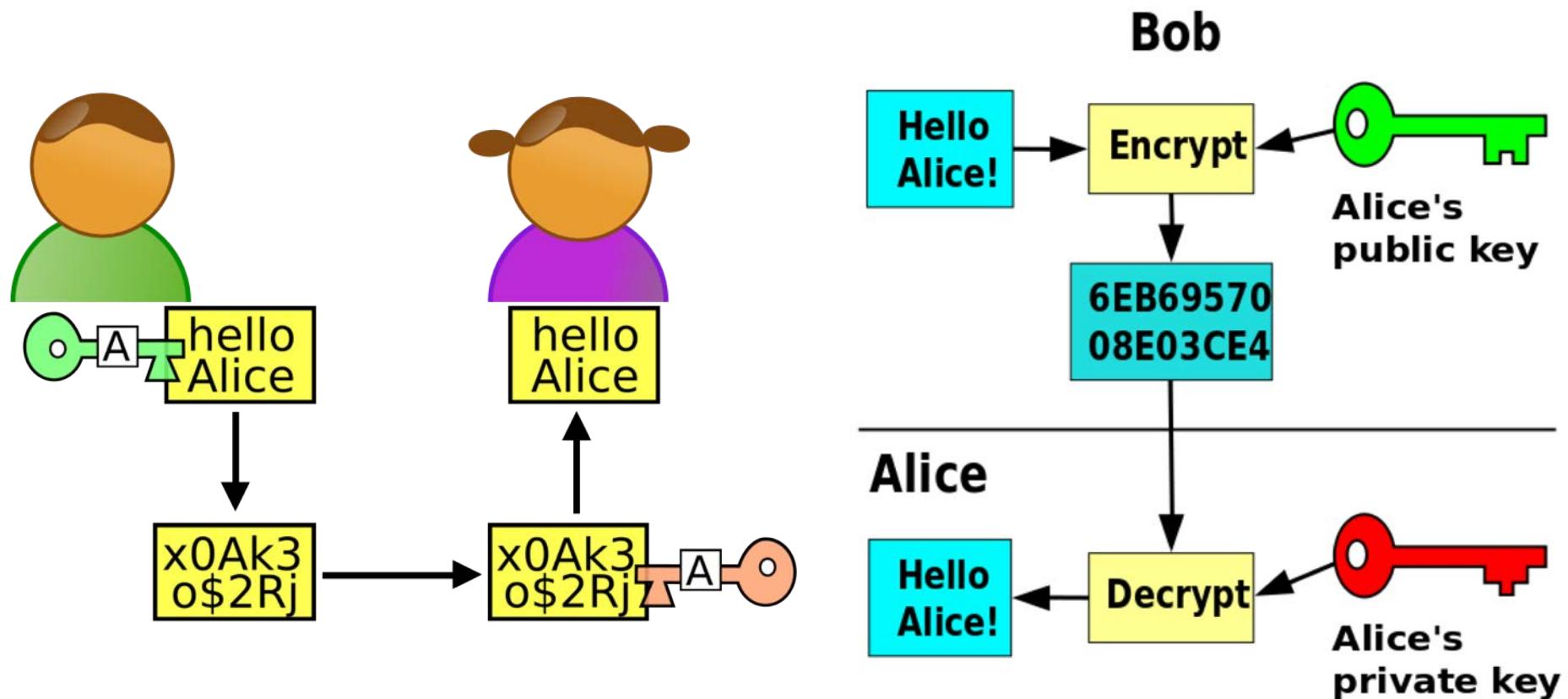
CIFRADO

FIRMA DIGITAL

INTERCAMBIO DE
CLAVES

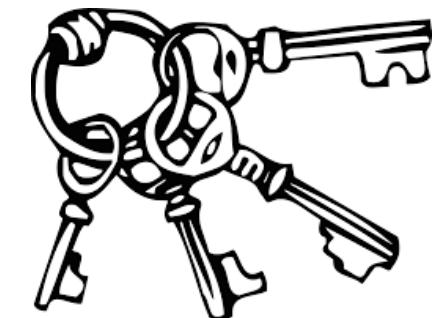
Cifrado/descifrado

- En el caso de que *Bob* necesite del servicio de confidencialidad para su comunicación con *Alice*, el **cifrado/descifrado** se realiza de la siguiente forma:



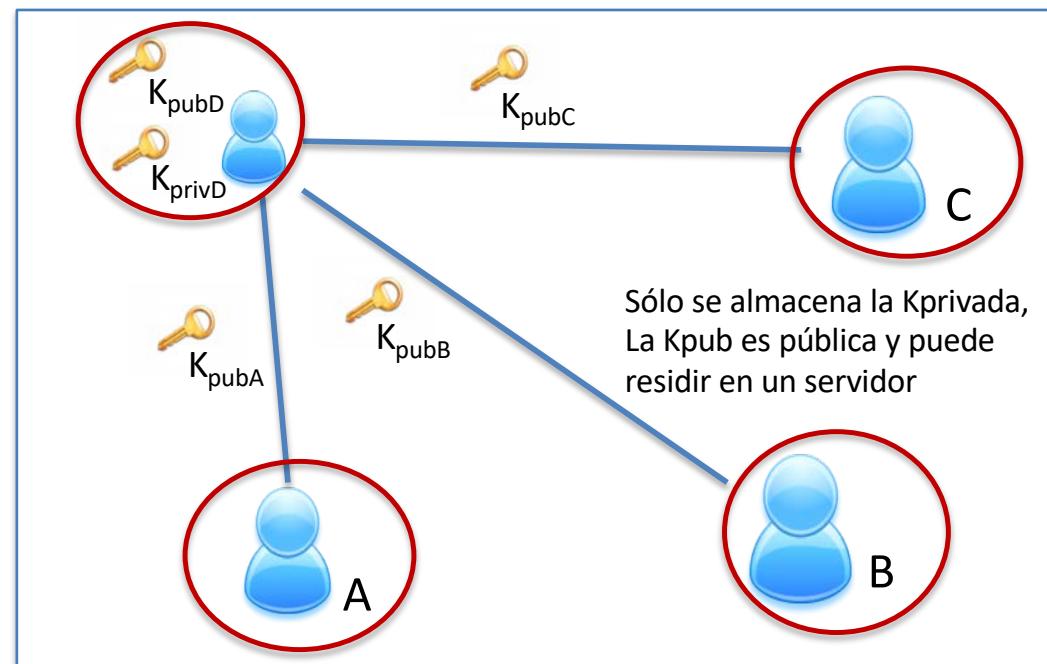
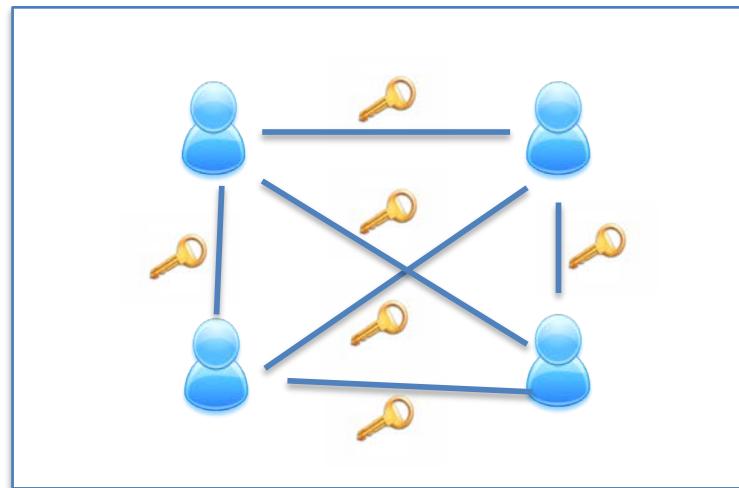
Cifrado/descifrado

- Del esquema anterior se entiende que *Alice* y *Bob* no necesitan acordar a priori ninguna clave (a diferencia de los algoritmos simétricos)
- Pero *Bob* ha de conocer la clave pública de *Alice*
 - y también la clave pública de cada uno de los usuarios con los que desee contactar
- Para ello, la solución más simple es que *Bob* almacene las claves públicas de los otros usuarios en un **key-ring** personal
 - Aunque la solución más común es que las claves públicas estén almacenadas en un **directorio de claves** en Internet



Cifrado/descifrado

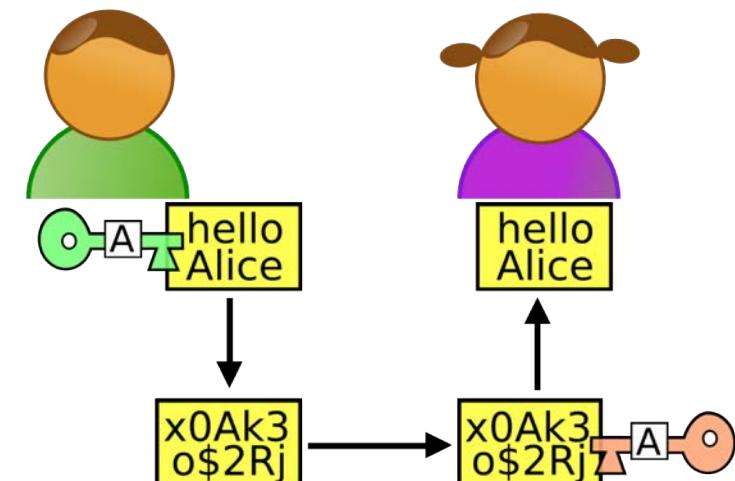
- Se deduce fácilmente que, usando un criptosistema de clave pública, y para una comunidad de n usuarios:
 - el número de claves en el sistema será **$2n$**
 - en lugar de $(n * (n-1))/2$ como era el caso simétrico



Cifrado/descifrado

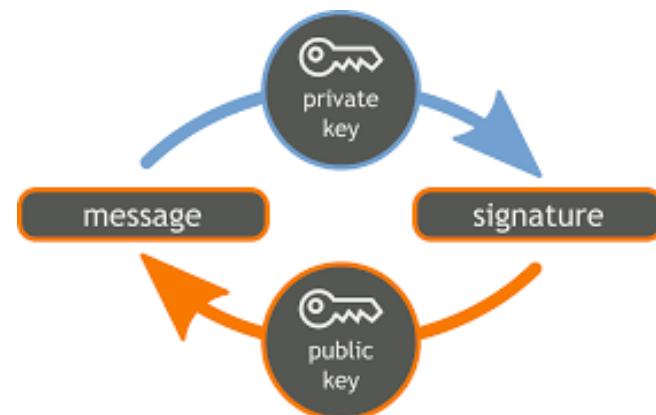


- Otras características relevantes:
 - es **computacionalmente imposible deducir la clave privada** del usuario U a partir de su clave pública
 - cualquier usuario con la **clave pública de U** puede **cifrar** un mensaje hacia U , pero no descifrarlo
 - cifrar el mensaje con la clave pública es como poner el correo en un buzón (todo el mundo puede hacerlo)
 - sólo U , con la correspondiente **clave privada**, **puede descifrar** el mensaje
 - descifrar el mensaje con la clave privada es como coger el correo del buzón (sólo el que tiene la llave del buzón puede hacerlo)



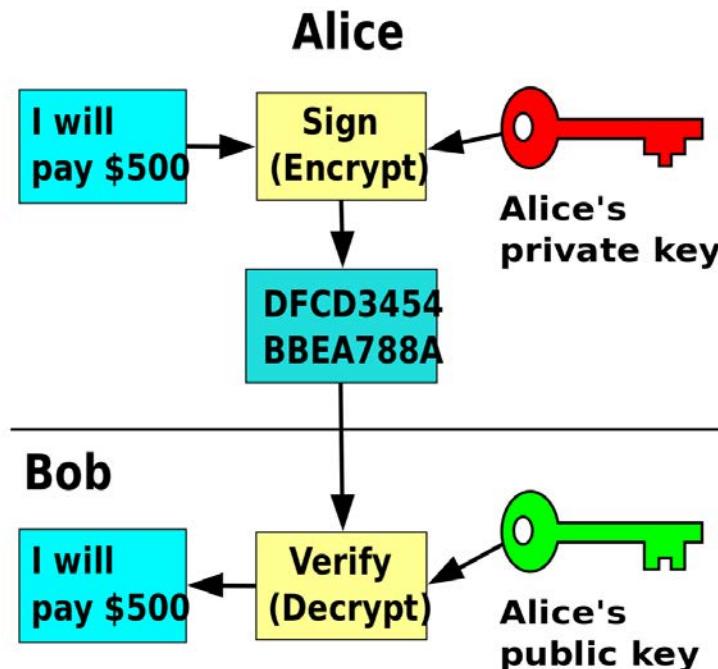
Cifrado/descifrado

- Por lo tanto, hemos visto tres **ventajas inmediatas de la criptografía de clave pública** con respecto a la simétrica
 1. Cuando dos usuarios se comunican confidencialmente **no necesitan acordar una clave a priori**
 2. Por lo anterior, **no resulta problemático que estén físicamente lejanos y no puedan reunirse presencialmente**
 3. El **número de claves** en el sistema **se reduce** sustancialmente
- Como se ve en la figura, existe aún una **ventaja adicional** tanto o más importante que las anteriores
 - Esa ventaja se deriva de la **dualidad de funcionamiento** de algunos (no todos) algoritmos de clave pública



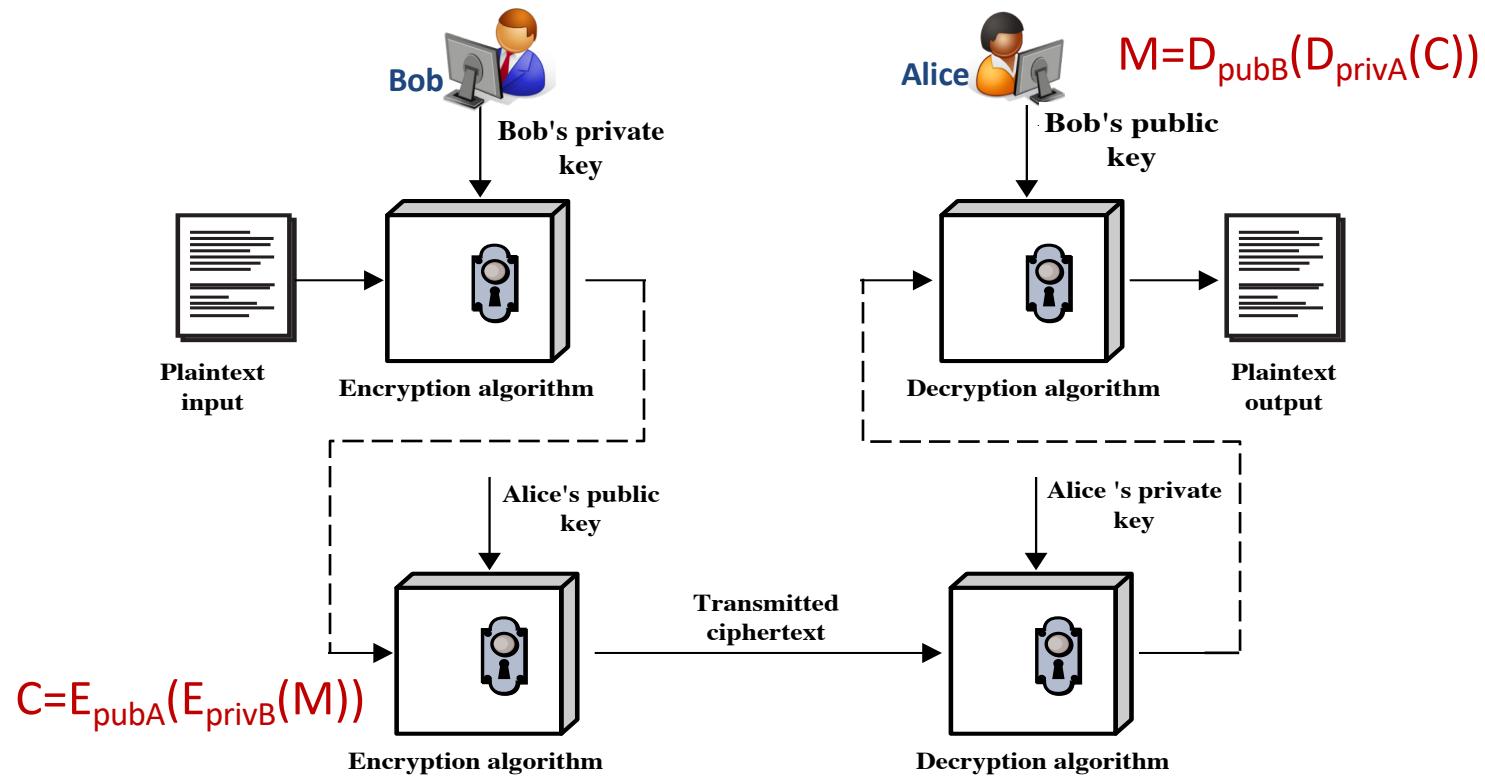
Firma digital

- *Bob* cifra el mensaje usando su propia clave privada, y cualquiera que tenga la clave pública de *Bob* podrá descifrar el criptograma
- Cuando *Alice* descifra el criptograma y obtiene el mensaje original, le queda garantizado que el mensaje viene de *Bob*
 - porque *Bob* es el único que pudo hacer la operación de cifrado (ya que sólo él posee la clave privada que generó el criptograma)



Firma digital

- Adicionalmente, es posible usar en secuencia las operaciones de firma digital y cifrado/descifrado, obteniendo autenticidad y confidencialidad en un mismo envío



Desventajas de criptografía de clave pública

- A pesar de estas ventajas, los algoritmos de clave pública tienen una **desventaja** de peso, el del gran tamaño de sus claves
 - Ejemplo de clave pública:

```
98 3f ad 19 36 93 3d 3e fe 76 42 14 fd 35 6f f1  
fa ad 22 7a 58 e3 46 d0 5d c6 5a f9 62 2d 8f 31  
5e fe b4 30 fe 50 74 ac d6 9d 1d e0 62 c6 49 dd  
14 12 7d 71 0b ac 06 c1 3f d7 06 87 e0 90 89 d6  
e5 e3 03 b2 f2 27 b1 9f 33 c8 aa 6b 36 4a a3 c4  
3f 79 41 9d 89 46 2f 2b 3e 63 d4 38 56 91 aa 1d  
b1 0d 42 75 4d f3 87 4e e3 0f 4d cc b4 6c bf 62  
13 87 ea d0 9b 8e b6 e2 ff 19 f4 94 09 d5 96 61
```



- Además, los algoritmos de clave pública se basan en **funciones matemáticas complejas** (en lugar de en las convencionales sustituciones y permutaciones de los simétricos)
- Ambos hechos hacen que **el rendimiento** de estos algoritmos sea **sustancialmente menor** que el de los simétricos
 - en general, se puede afirmar que son unos 1000 veces más lentos

Desventajas de criptografía de clave pública

```
$ openssl speed rc4
```

To get the most accurate results, try to run this program when this computer is idle.

```
Doing rc4 for 3s on 16 size blocks: 73270739 rc4's in 2.99s
Doing rc4 for 3s on 64 size blocks: 19548456 rc4's in 2.99s
Doing rc4 for 3s on 256 size blocks: 5017905 rc4's in 2.99s
Doing rc4 for 3s on 1024 size blocks: 1274653 rc4's in 2.98s
Doing rc4 for 3s on 8192 size blocks: 159407 rc4's in 2.97s
```

```
$ openssl speed aes
```

To get the most accurate results, try to run this program when this computer is idle.

```
Doing aes-128 cbc for 3s on 16 size blocks: 30108378 aes-128 cbc's in 2.97s
Doing aes-128 cbc for 3s on 64 size blocks: 7712443 aes-128 cbc's in 2.96s
Doing aes-128 cbc for 3s on 256 size blocks: 1953741 aes-128 cbc's in 2.98s
Doing aes-128 cbc for 3s on 1024 size blocks: 490976 aes-128 cbc's in 2.98s
Doing aes-128 cbc for 3s on 8192 size blocks: 61237 aes-128 cbc's in 2.98s
Doing aes-192 cbc for 3s on 16 size blocks: 26695873 aes-192 cbc's in 2.98s
Doing aes-192 cbc for 3s on 64 size blocks: 6930418 aes-192 cbc's in 2.98s
Doing aes-192 cbc for 3s on 256 size blocks: 1729199 aes-192 cbc's in 2.97s
Doing aes-192 cbc for 3s on 1024 size blocks: 444845 aes-192 cbc's in 2.98s
Doing aes-192 cbc for 3s on 8192 size blocks: 52989 aes-192 cbc's in 2.97s
Doing aes-256 cbc for 3s on 16 size blocks: 23329778 aes-256 cbc's in 2.97s
Doing aes-256 cbc for 3s on 64 size blocks: 5958585 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 256 size blocks: 1565944 aes-256 cbc's in 2.97s
Doing aes-256 cbc for 3s on 1024 size blocks: 377290 aes-256 cbc's in 2.97s
Doing aes-256 cbc for 3s on 8192 size blocks: 47844 aes-256 cbc's in 2.94s
```

```
$ openssl speed rsa
```

To get the most accurate results, try to run this program when this computer is idle.

```
Doing 512 bit private rsa's for 10s: 79651 512 bit private RSA's in 9.98s
Doing 512 bit public rsa's for 10s: 1079143 512 bit public RSA's in 9.95s
Doing 1024 bit private rsa's for 10s: 22746 1024 bit private RSA's in 9.96s
Doing 1024 bit public rsa's for 10s: 460663 1024 bit public RSA's in 9.96s
Doing 2048 bit private rsa's for 10s: 4362 2048 bit private RSA's in 9.96s
Doing 2048 bit public rsa's for 10s: 174994 2048 bit public RSA's in 9.97s
Doing 4096 bit private rsa's for 10s: 729 4096 bit private RSA's in 9.98s
Doing 4096 bit public rsa's for 10s: 50938 4096 bit public RSA's in 9.98s
```

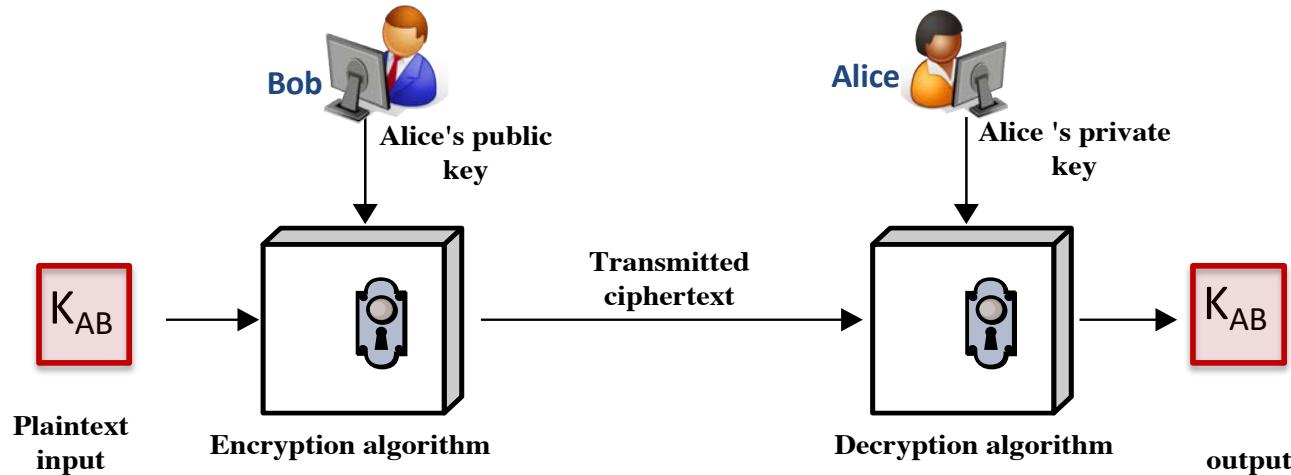
```
$ openssl speed dsa
```

To get the most accurate results, try to run this program when this computer is idle.

```
Doing 512 bit sign dsa's for 10s: 125836 512 bit DSA signs in 9.99s
Doing 512 bit verify dsa's for 10s: 114530 512 bit DSA verify in 9.99s
Doing 1024 bit sign dsa's for 10s: 54566 1024 bit DSA signs in 10.00s
Doing 1024 bit verify dsa's for 10s: 46194 1024 bit DSA verify in 10.00s
Doing 2048 bit sign dsa's for 10s: 18965 2048 bit DSA signs in 10.00s
Doing 2048 bit verify dsa's for 10s: 16315 2048 bit DSA verify in 10.00s
```

Intercambio de claves

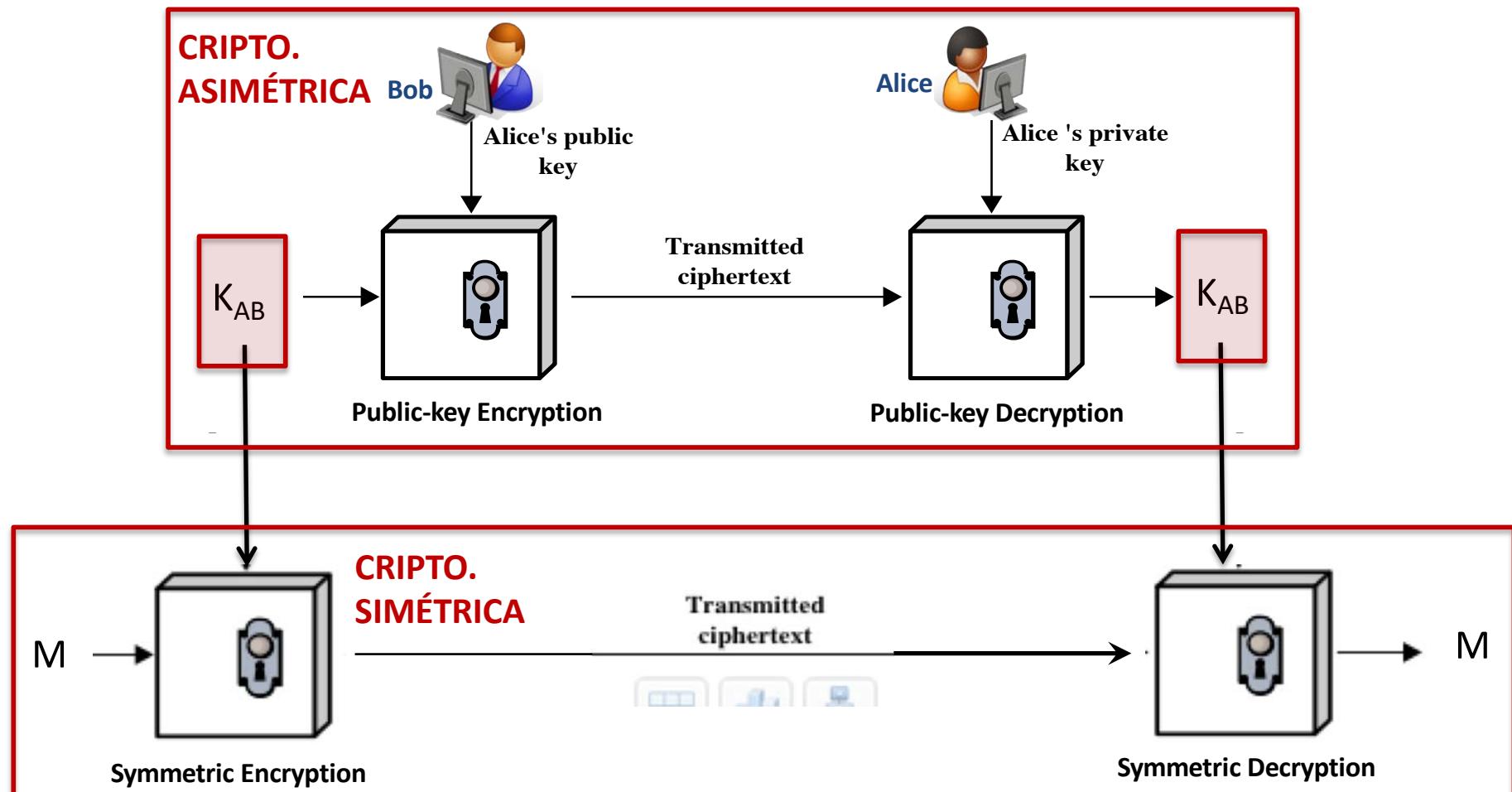
- Debido a su bajo rendimiento, se ha ideado una tercera funcionalidad para estos criptosistemas (además de cifrado/descifrado y firma digital): el **intercambio de claves**
 - *Paso 1:* Bob y Alice usan el algoritmo asimétrico para la transmisión (cifrado/descifrado) de la **clave secreta K_{AB}**
 - *Paso 2:* Ambos usarán K_{AB} para, posteriormente, cifrar sus comunicaciones con un algoritmo simétrico



- El resultado final es un **criptosistema híbrido** (uso de criptosistema de clave pública + uso de criptosistema simétrico)

Intercambio de claves

- Una evolución del planteamiento anterior es el siguiente:
 - se envía simultáneamente la clave de sesión y el mensaje cifrado con esa misma clave



Funcionalidades de la criptografía de clave pública

- Hay que hacer notar que, como muestra la siguiente tabla, no todos los algoritmos de clave pública son capaces de realizar las tres funcionalidades mencionadas:

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

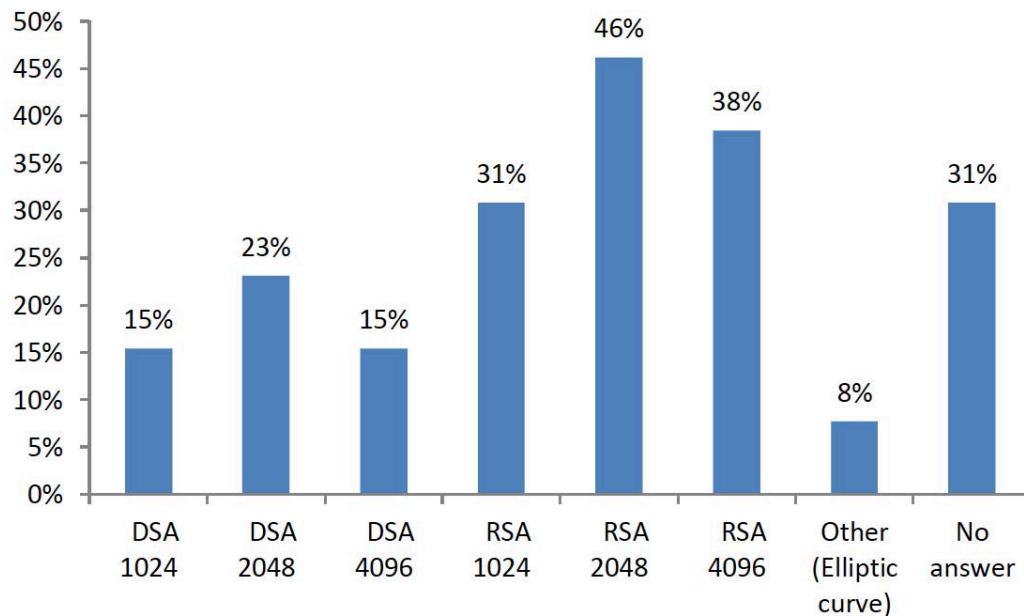
Cifrado/Descifrado

Firma digital

Intercambio de
claves

DSS: Digital Signature Standard, e incorpora
DSA (Digital Signature Algorithm)

- De acuerdo a un estudio reciente entre agencias e instituciones gubernamentales de la Unión Europea, el uso de algoritmos de clave pública es el mostrado en las figuras (para firma digital e intercambio de claves, respectivamente)



Algoritmo Diffie-Hellman

- Diseñado en 1976, se denomina normalmente “**algoritmo Diffie-Hellman de intercambio de clave**”
- Permite a dos usuarios cualesquiera **intercambiar**, de forma confidencial, una **clave secreta K (o de sesión)** para, posteriormente, cifrar de forma simétrica los mensajes entre ellos dos
- La efectividad del algoritmo depende de la **dificultad de computar logaritmos discretos**
 - Si α es una *raíz primitiva* del número primo q , entonces los números $\alpha \bmod q, \alpha^2 \bmod q, \dots, \alpha^{q-1} \bmod q$ son distintos entre sí, y sus valores son los enteros 1 a $q-1$, en cualquier orden
 - Para cualquier entero b y una raíz primitiva α del número primo q , se puede encontrar un exponente único i tal que:
$$b \equiv \alpha^i \pmod{q} \text{ donde } 0 \leq i \leq (q-1)$$
 i es el **logaritmo discreto de b** , y se representa $dlog_{\alpha,q}(b)$

Global Public Elements

q prime number

α $\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A $X_A < q$

Calculate public Y_A $Y_A = \alpha^{X_A} \text{ mod } q$

User B Key Generation

Select private X_B $X_B < q$

Calculate public Y_B $Y_B = \alpha^{X_B} \text{ mod } q$

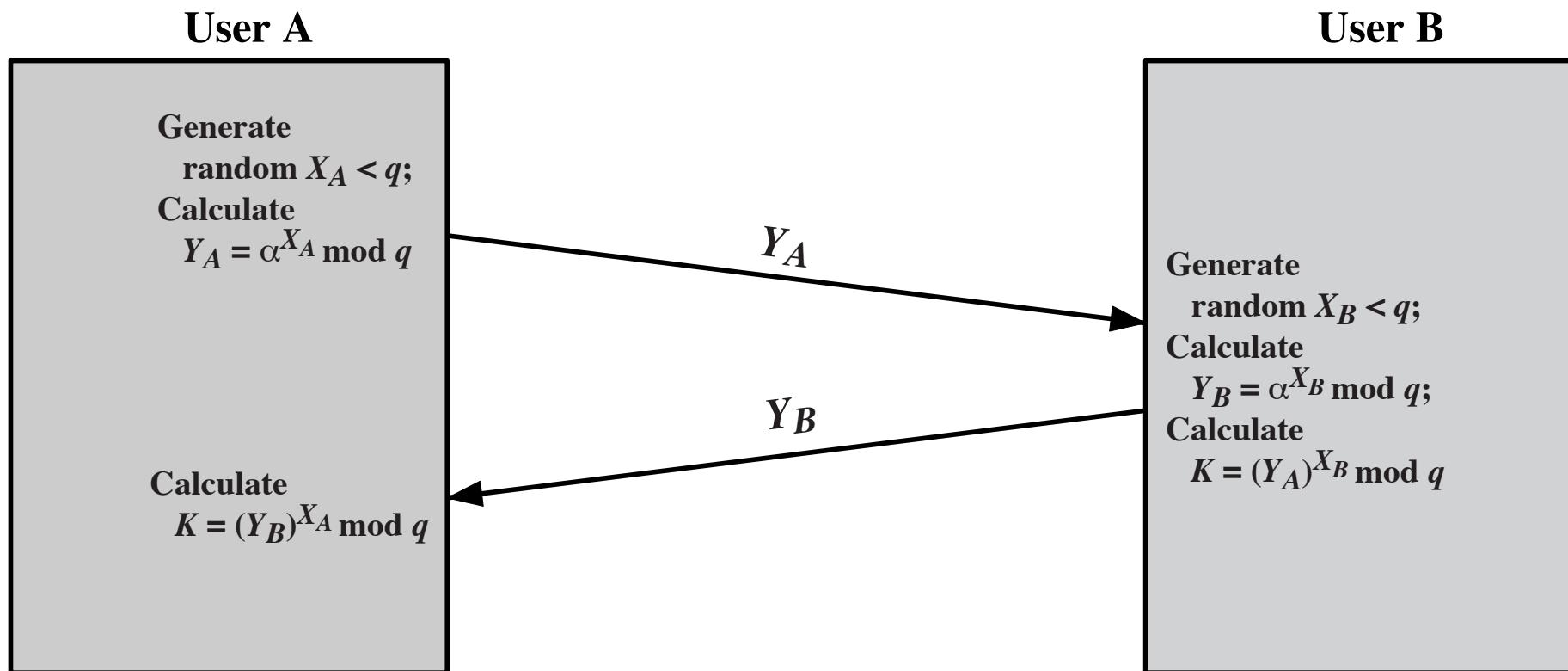
Calculation of Secret Key by User A

$$K = (Y_B)^{X_A} \text{ mod } q$$

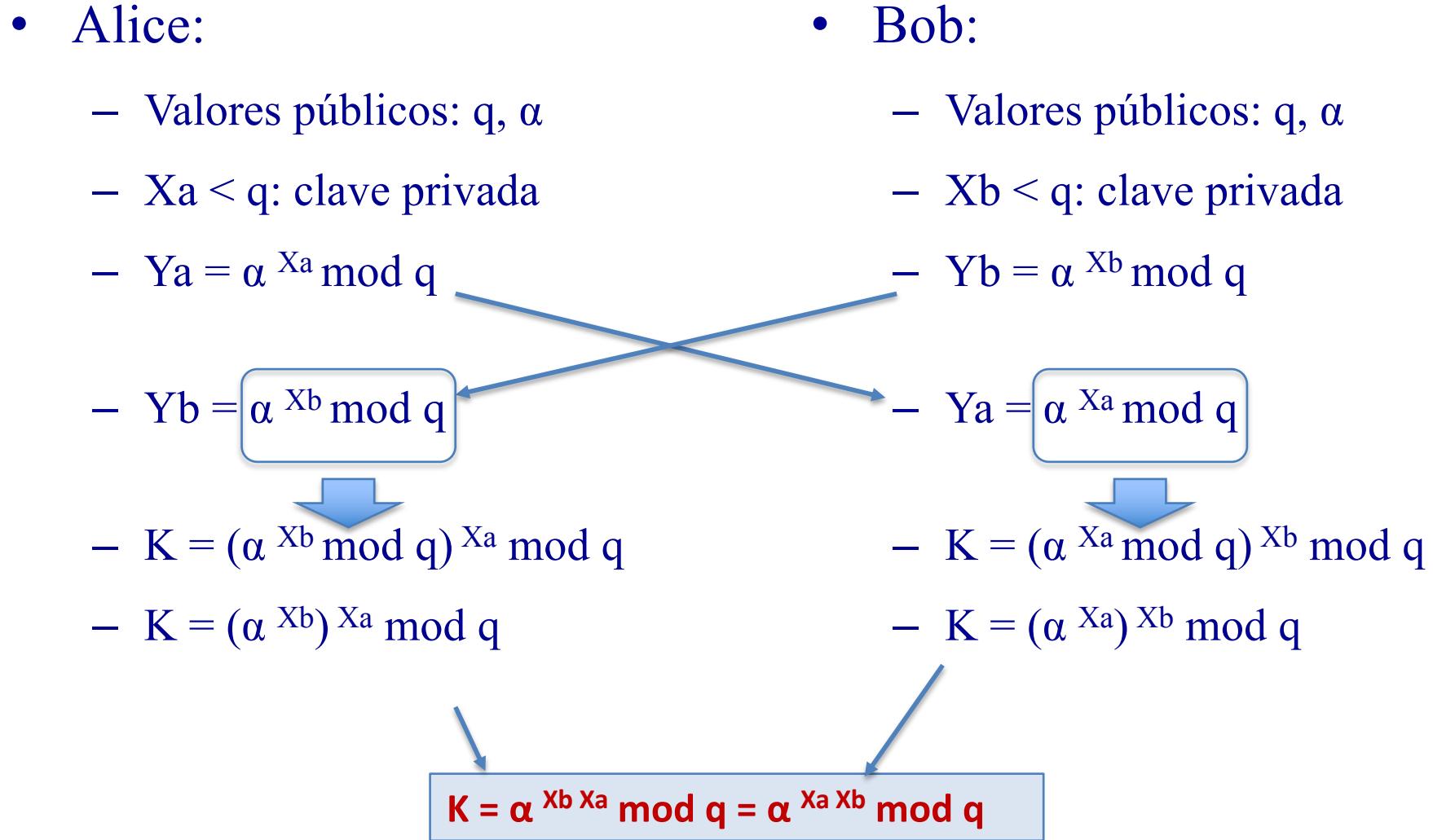
Calculation of Secret Key by User B

$$K = (Y_A)^{X_B} \text{ mod } q$$

**Algoritmo
Diffie-Hellman
de intercambio
de clave**



**Secuencia ordenada
de pasos del algoritmo**



- Ejemplo:
 - ① $q = 353$; $\alpha = 3$ (3 es raíz primitiva de 353)
 - ② A y B seleccionan sus correspondientes claves privadas: $X_A = 97$, $X_B = 233$
 - ③ A calcula su clave pública: $Y_A = 3^{97} \text{ mod } 353 = 40$
 - ④ B calcula su clave pública: $Y_B = 3^{233} \text{ mod } 353 = 248$
 - ⑤ A y B se intercambian sus claves públicas
 - ⑥ A calcula $K = (Y_B)^{X_A} \text{ mod } 353 = 248^{97} \text{ mod } 353 = 160$
 - ⑦ B calcula $K = (Y_A)^{X_B} \text{ mod } 353 = 40^{233} \text{ mod } 353 = 160$

- Ejemplo (continuación):
 - Se asume que el atacante tendrá la siguiente información:
 $q = 353$; $\alpha = 3$; $Y_A = 40$; $Y_B = 248$
 - En este ejemplo simple, sería posible usar un ataque exhaustivo para determinar que $K = 160$
 - El atacante calcula fácilmente $3^{X_a} \bmod 353 = 40$ o bien $3^{X_b} \bmod 353 = 248$
 - Con números más grandes el problema no es resoluble en un tiempo razonable

- http://dkerr.home.mindspring.com/diffie_hellman_calc.html

Example: Diffie - Hellman Define Public Values:

$n =$

$g =$

Both Alice and Bob each pick a private x and compute a public $X = g^x \bmod n$

Alice

Alice chooses a Private Value $a =$

Bob

Bob chooses a Private Value $b =$

- or -

- or -

Alice computes Public Value: $A = g^a \bmod n$ Bob computes Public Value: $B = g^b \bmod n$

(Public) $A =$

(Public) $B =$

Alice and Bob exchange Public Values

Alice and Bob each compute Same Master Value M

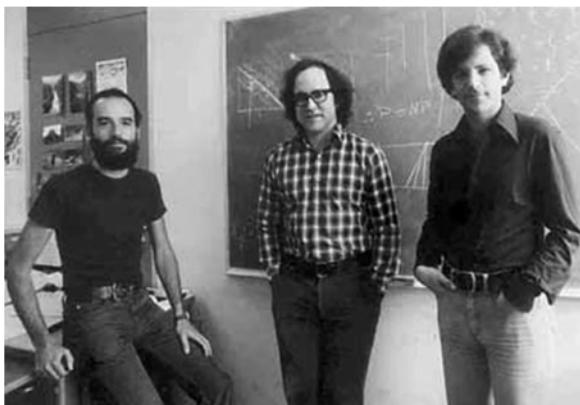
Alice computes $M = B^a \bmod n = g^{ba} \bmod n$ Bob computes $M = A^b \bmod n = g^{ab} \bmod n$

Alice computes $M =$

Bob computes $M =$

Algoritmo RSA

- Es el criptosistema de clave pública más usado
 - Su nombre procede de sus inventores: *Rivest, Shamir y Adleman* del Instituto Tecnológico de Massachusetts (MIT)
- Estos se basaron para su invención, en 1.977, en una idea bastante “simple”:
 - “*es muy fácil multiplicar dos números enteros primos grandes, pero extremadamente difícil hallar la factorización de ese producto*”
 - puede darse a conocer dicho producto sin que nadie descubra esos números de procedencia, a no ser que conozca al menos uno de ellos



Se piensa que RSA será seguro mientras no se conozcan “**formas rápidas**” de descomponer un número grande en producto de primos

- Parámetros necesarios:
 - encontrar dos números primos grandes p y q , y calcular

$$n = p * q ; p \neq q$$
 - se calcula $\varphi(n)$, de forma que

$$\varphi(n) = (p - 1) * (q - 1)$$
 - se elige aleatoriamente un número grande e tal que

$$\text{MCD}(e, \varphi(n)) = 1; e < \varphi(n)$$

(o sea, e y $\varphi(n)$ son primos relativos o coprimos)
 - Se determina el número d que cumple

$$e * d \equiv 1 \pmod{\varphi(n)}$$

(donde d debe ser el multiplicador inverso de e mod $\varphi(n)$)

Para extraer los primos,
donde $\varphi(n)$ se define como
el número de enteros positivos
menores o iguales a n y coprimos de n

Calcular
las
claves

- n, d y e (y por supuesto, p y q) son la base del sistema
 - n módulo
 - d clave privada
 - e clave pública
- Las claves tienen el siguiente uso:
 - la **clave privada**: para descifrar o firmar mensajes
 - la **clave pública**: para cifrar o verificar la firma
- Para cifrar: $C = M^e \pmod{n}$
- Para descifrar: $M = C^d \pmod{n}$
- RSA se trata, por lo tanto, de un algoritmo exponencial



$$C^d = (M^e)^d \equiv M^{ed} \pmod{n}$$

- Ejemplo del cálculo de la clave pública y privada:

① Seleccionar primos: $p = 17$, $q = 11$

① Calcular $n = pq = 17 \times 11 = 187$

② Calcular $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$

③ Seleccionar e : $\text{mcd}(e, 160) = 1$ y $e < 160$
se selecciona $e = 7$

⑤ Determinar d : $d \cdot e \equiv 1 \pmod{160}$
 $d = 23$ dado que $23 \times 7 = 161 \pmod{160} = 1$

⑥ Clave pública = 7 y módulo = 187

⑦ Clave privada = 23

- El texto original, M (y también el cifrado, C) debe tomarse como un número decimal. De esta forma, si se trabaja con el código ASCII:

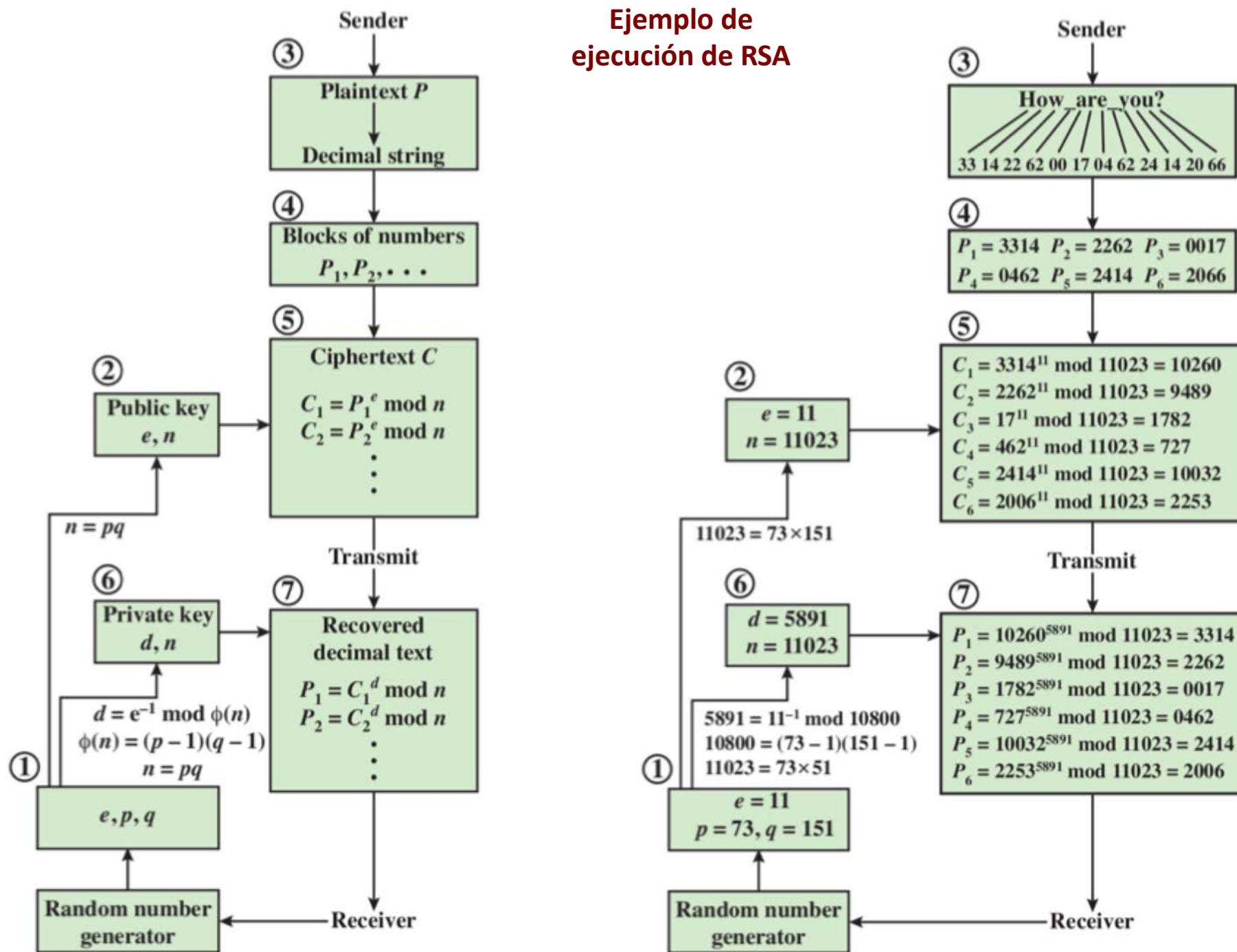
$M = \text{"Hola"}$ equivaldría al valor decimal 72 111 108 097

- Además, como se está trabajando en aritmética modular, todos los valores usados deben ser menores que el módulo n
 - Esto significa que si el valor decimal del mensaje es superior al módulo, $M > n$, entonces M debe ser troceado en otros menores que n
 - Suponiendo $n = 100000$, daría lugar a los bloques

$$m_0 = 72111, m_1 = 10809 \text{ y } m_2 = 7$$

- Los bloques m_i no deben/pueden ser pequeños, pues entonces el criptoanalista puede construirse una tabla donde tenga todos los posibles m_i y sus respectivos c_i
- Para seleccionar p ó q debe testearse si es primo o no con cualquiera de los tests existentes (se recomienda *Miller-Rabin*)
- También hay que tener cuidado al elegir el tamaño
 - **cuanto mayor sea, más seguro, pero también más lento** será el sistema en sus cálculos
- Se han propuesto sistemas en los que se busca un valor e muy pequeño, con lo cual hacer más rápido el proceso de cifrado

Ejemplo de ejecución de RSA



Comparativa: Criptografía Simétrica vs. Asimétrica

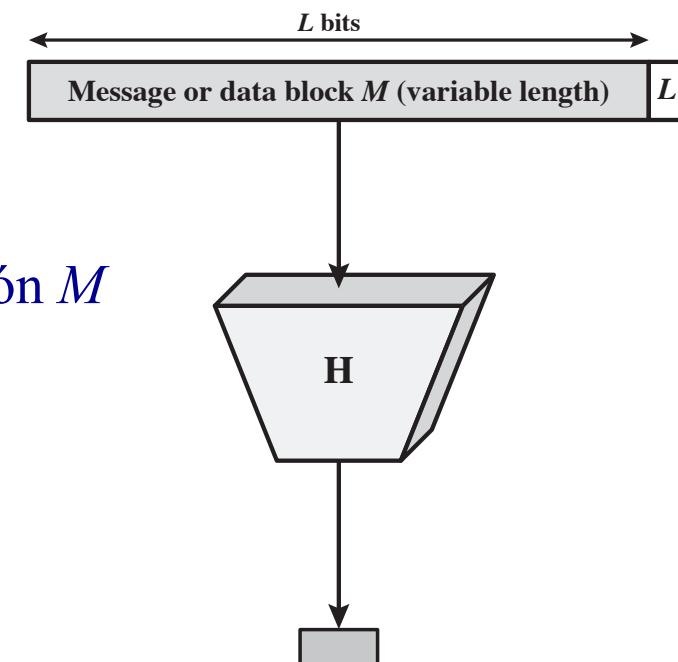
Atributo	Clave simétrica	Clave asimétrica
Años en uso	Miles	Menos de 50
Uso principal	Cifrado de grandes volúmenes de datos	Intercambio de claves; firma digital
Estándar actual	DES, Triple DES, AES	RSA, Diffie-Hellman, DSA
Velocidad	Rápida	Lenta
Claves	Compartidas entre emisor y receptor	Privada: sólo conocida por una persona Pública: conocida por todos
Intercambio de claves	Difícil de intercambiar por un canal inseguro	La clave pública se comparte por cualquier canal La privada nunca se comparte
Longitud de claves	56 bits (vulnerable) 256 bits (seguro)	1024 – 2048 (RSA) 172 (curvas elípticas)
Servicios de seguridad	Confidencialidad Integridad Autenticación	Confidencialidad Integridad Autenticación, No repudio

Otras primitivas criptográficas



Funciones Hash

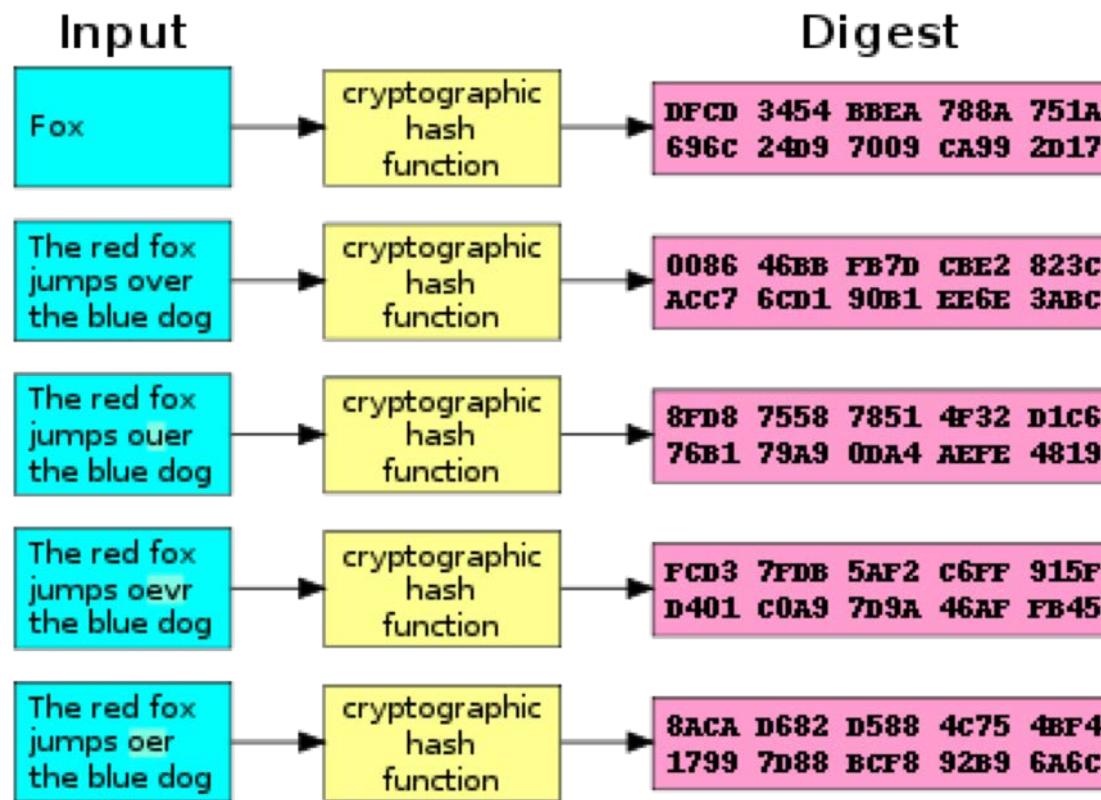
- Una función unidireccional (o de sentido único) es un bloque de construcción para muchos protocolos criptográficos
 - Son fáciles de computar, pero muy difíciles de invertir
 - Por lo tanto, no son útiles de cara al cifrado
 - porque un mensaje cifrado mediante una función unidireccional no se podría descifrar
 - No hay pruebas ni evidencia real de que existan, pero muchas funciones tienen apariencia de serlo



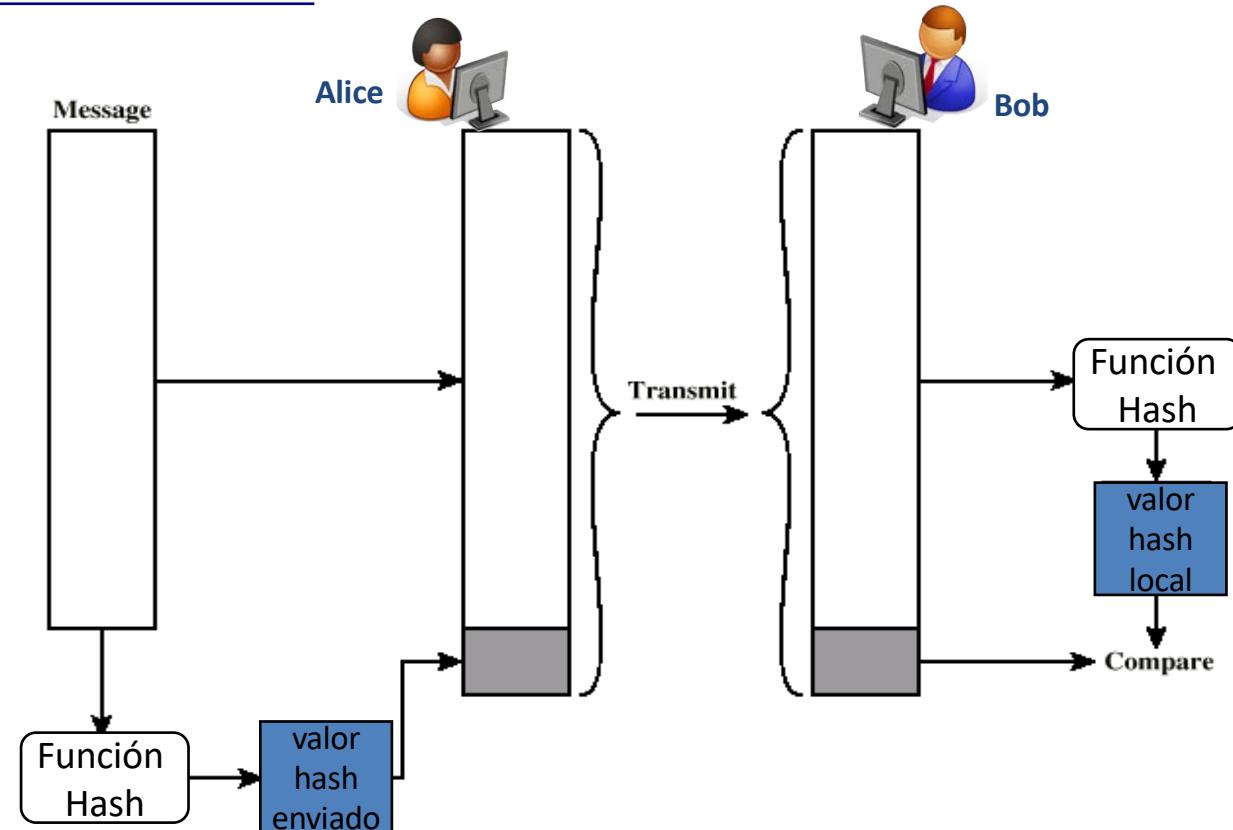
- Una función hash es una función que:
 - acepta como entrada un bloque de información M (preimagen) de longitud variable
 - produce un bloque de salida h (valor hash) de longitud fija
 - Ese valor se denomina huella digital de M

- Una función hash criptográfica es un algoritmo con el que es fácil computar el valor hash a partir de una preimagen, pero para el que resulta computacionalmente imposible:
 1. encontrar la preimagen M para un valor hash h predeterminado (propiedad: **unidireccionalidad**),
 2. encontrar dos bloques M y M' que produzcan el mismo valor hash h (propiedad: **libre de colisiones**)
- Para cualesquiera dos entradas M y M' a una función hash criptográfica, se producen dos salidas h y h' diferentes, pero no se puede discernir la relación entre esas variaciones
 - Concretamente, un cambio en un único bit de la preimagen da lugar a un cambio de, como media, la mitad de los bits de salida

- Efecto de una función hash (caso de *SHA-1*):

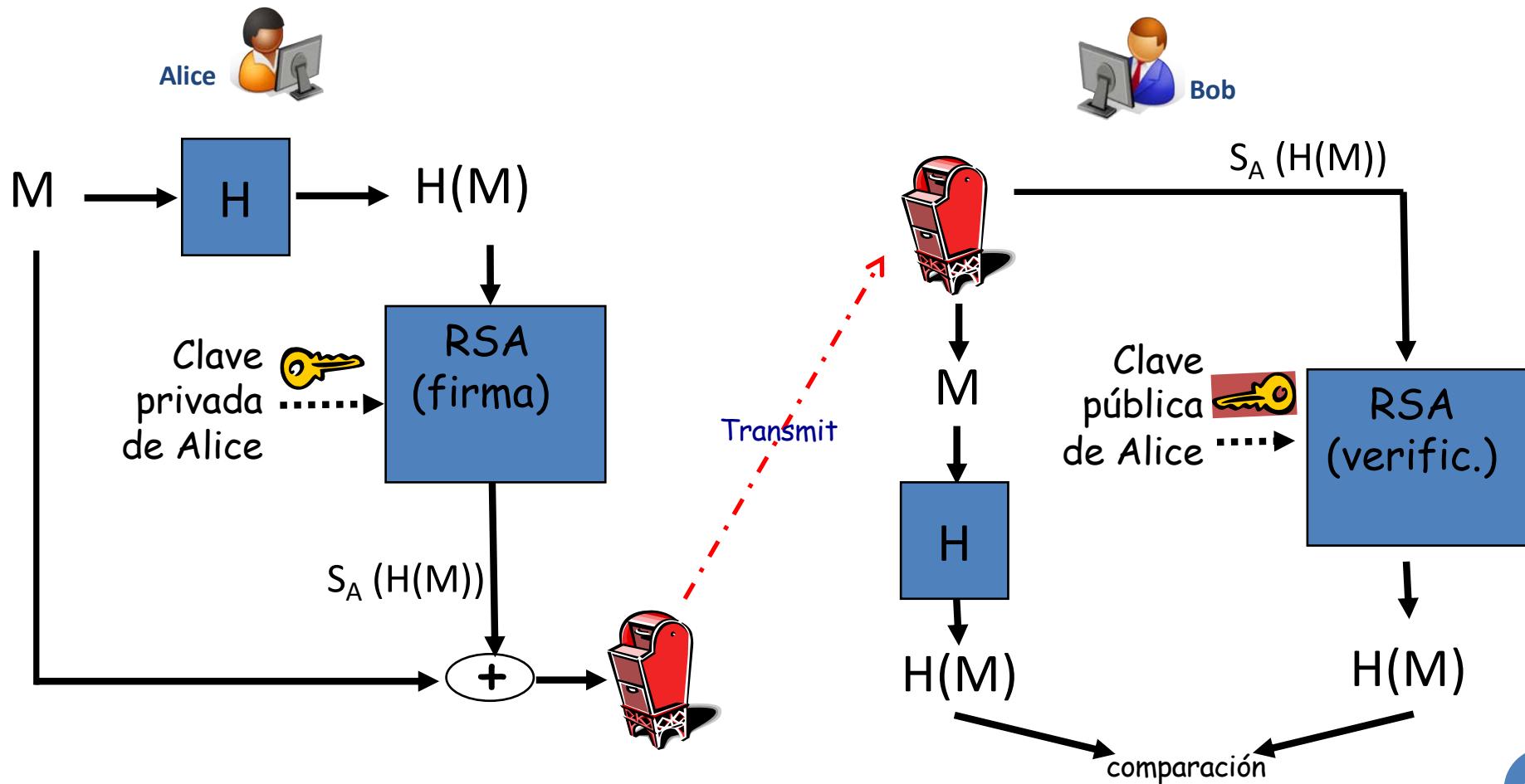


- La utilidad más inmediata de una función hash es proporcionar el servicio de integridad de datos
- Ejemplo de uso:

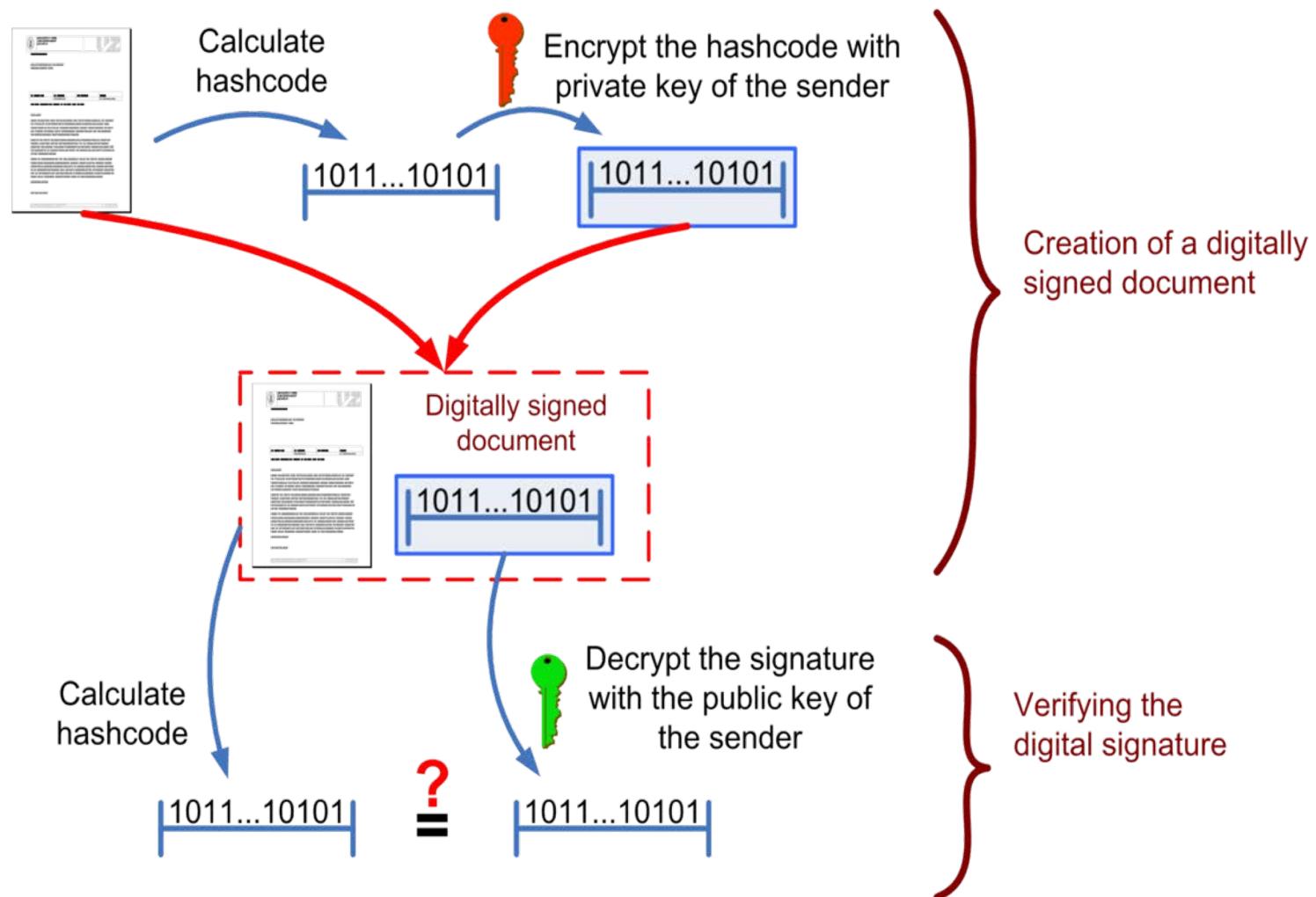


- Sin embargo, se observa que *no hay autenticidad* (*Bob* no tiene garantías de que *Alice* sea el origen)

- El uso combinado de las funciones hash con la criptografía de clave pública mejora sustancialmente la eficiencia de uso de esta última (en el procedimiento específico de firma digital)

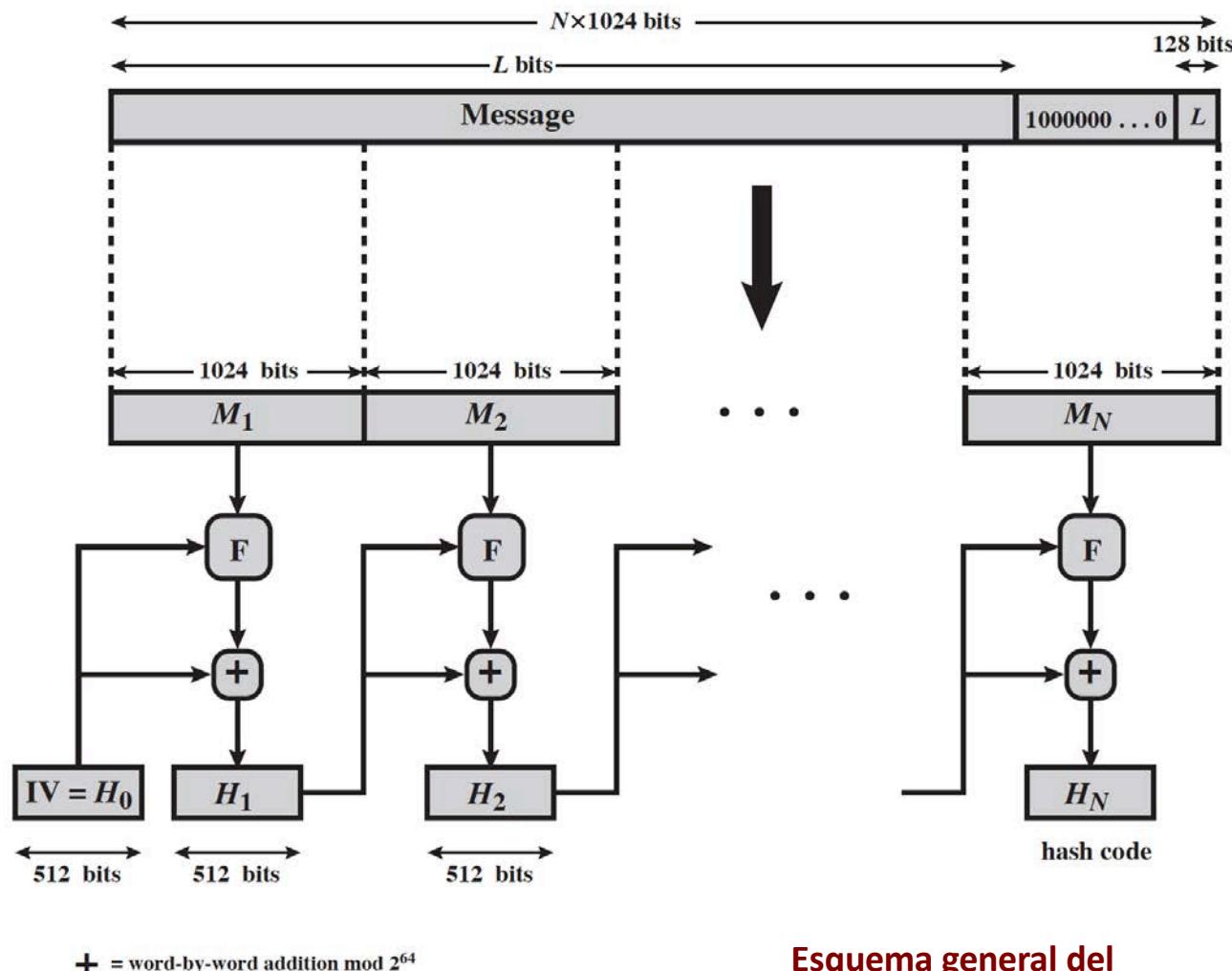


Creating and verifying a digital signature

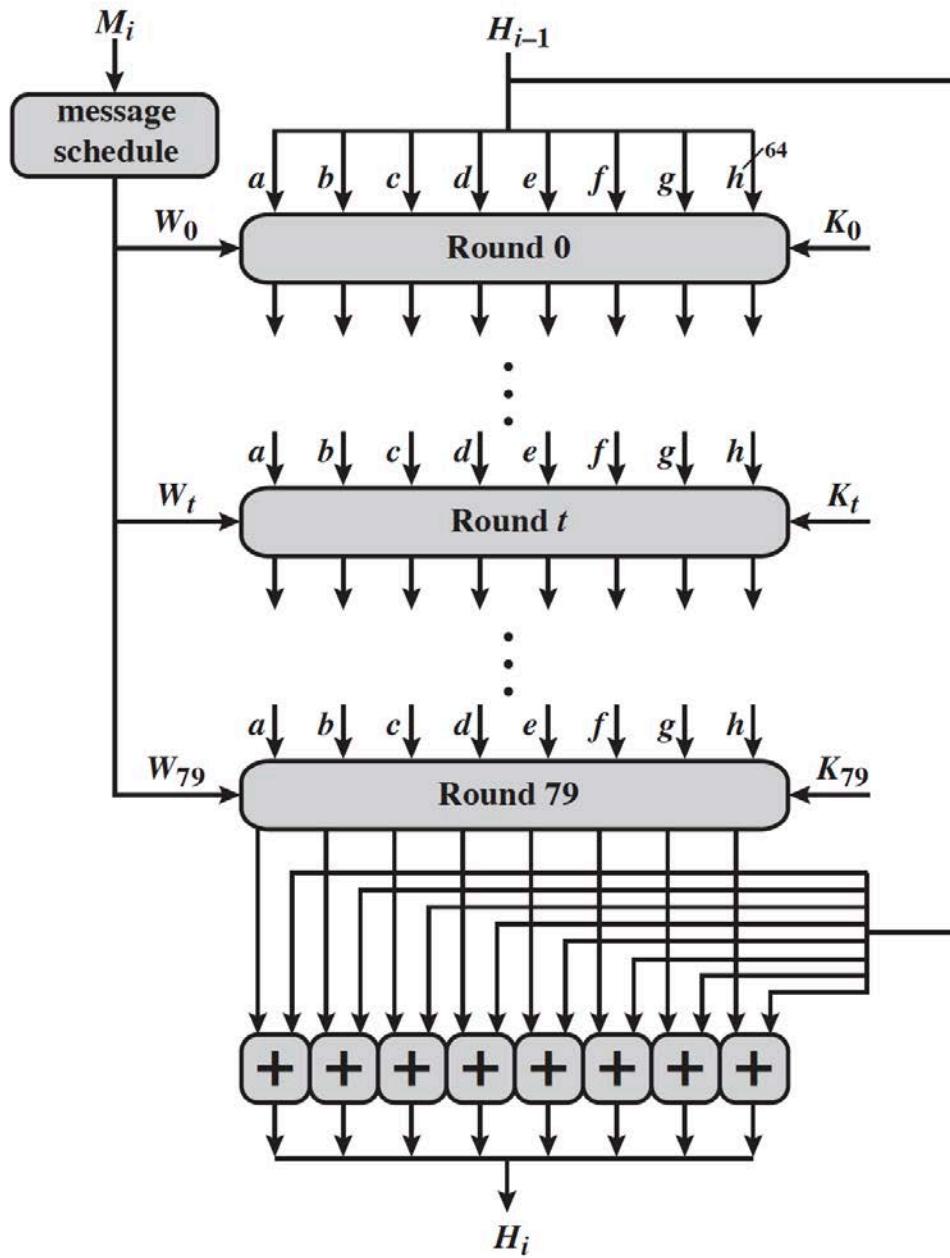


- Los beneficios del esquema anterior son:
 - la firma se puede mantener separada del documento
 - los requisitos de almacenamiento y firma son mucho menores
- Un sistema de archivos puede usar este tipo de esquema para verificar la existencia de documentos sin tener que almacenar sus contenidos

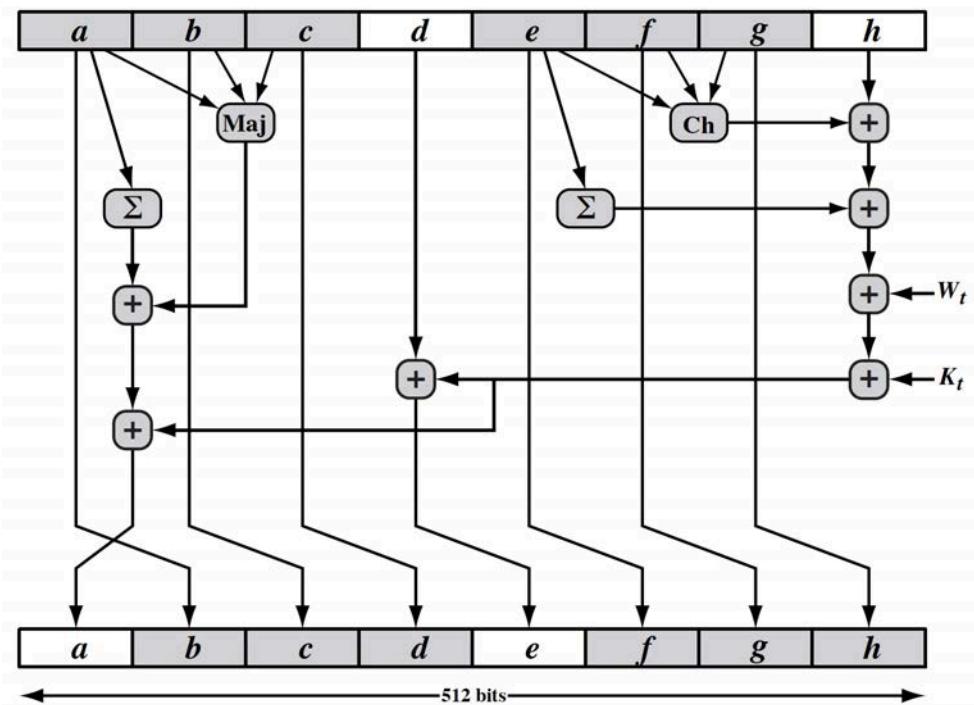
- Ejemplo de función hash: *SHA-512 (SHA-2)*



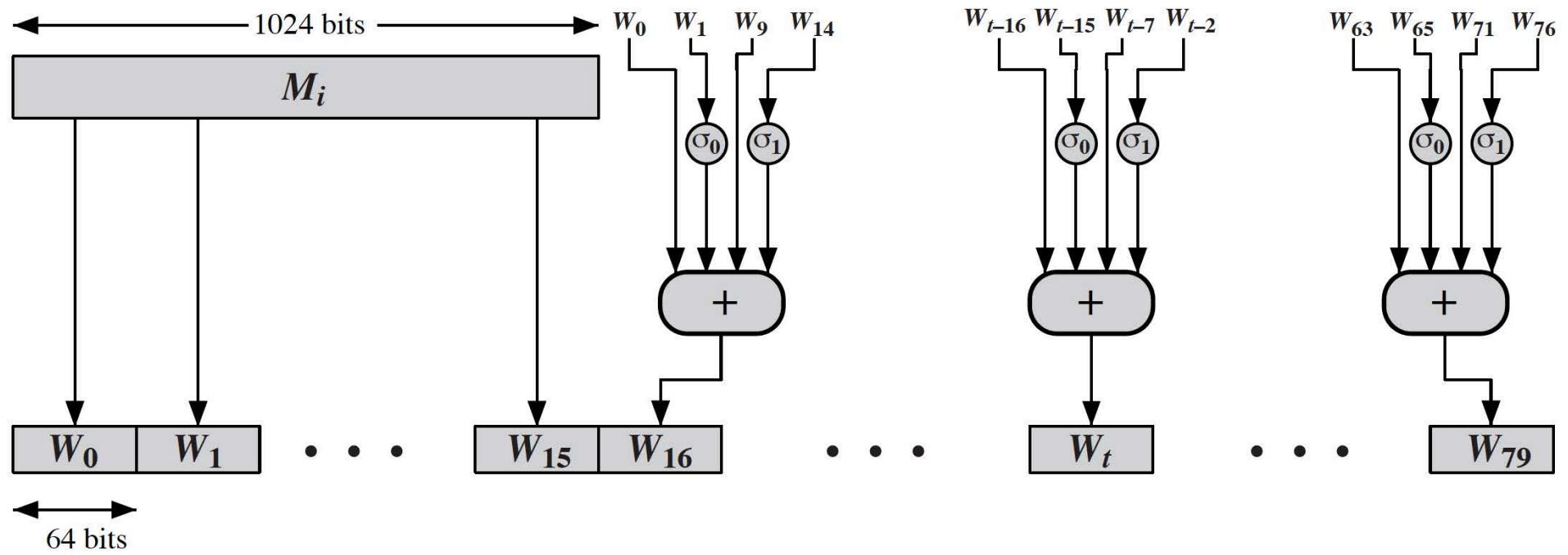
Esquema general del
algoritmo SHA-512



Procesamiento de un
bloque de 1024 bits
en SHA-2



Operación elemental (una etapa) en SHA-2



Creación de secuencia de entrada de $W_0..W_{79}$ para procesamiento del bloque M_i en SHA-2

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fdb4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90beffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceeaa26619c	d186b8c721c0c207	eadaddd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9beb9	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cf657e2a	5fc6fab3ad6faec	6c44198c4a475817

Constantes $K_0..K_{79}$ en SHA-2

Funciones hash

- Estas son algunas de las funciones hash más conocidas:
 - MD-5
 - A pesar de estar muy extendida, **MD-5 no se debería considerar segura**, porque se puede atacar criptoanalíticamente, encontrando colisiones en un margen de varios segundos
 - RIPEMD-128
 - **No se puede considerar segura** en la actualidad con independencia de los **ataques criptoanalíticos** que puedan reducir la complejidad global
 - SHA-1
 - Se utiliza extensivamente pero **ya no se considera segura**
 - El criptoanálisis de *MD-5* se ha aplicado también a *SHA-1* (dados que son de la misma familia) y se han encontrado colisiones no sólo del punto de vista teórico sino también práctico

SHA-1

Criptoanalistas 'rompen' SHA1

theresister.co.uk 17-Feb-2005

 Twittear



Share

El rumor que se ha estado corriendo, es ahora oficial, el algoritmo popular SHA-1 ha estado siendo atacado exitosamente por investigadores en China y Estados Unidos. Se ha descubierto una "colisión" en la versión completa en 2^{69} operaciones de hash, haciendo posible intentar un ataque de fuerza bruta exitoso con las computadoras más potentes de la actualidad.

Esto no significa desastre en términos prácticos, ya que la cantidad de poder computacional y conocimiento matemático necesario para poder llevar a cabo un ataque exitoso es elevado. Pero se ha demostrado que el algoritmo SHA-1 no está más allá del alcance de las supercomputadoras, como se había creído o por lo menos esperado. Teóricamente, se requeriría aproximadamente 2^{80} operaciones para poder encontrar una colisión.

Usando versiones de rendondeo-reducido del algoritmo, y la técnica del equipo, fue posible atacar el SHA-1 con menos de 2^{33} operaciones. Usando la misma técnica, el algoritmo completo SHA-0 pudo ser atacado en 2^{39} operaciones.

SHA-1 se había presumido ser más seguro que MD5, en donde las colisiones fueron encontradas el último año por algunas personas que reportaron el descubrimiento reciente. Además, en el último año, se han encontrado colisiones en SHA-0 por un equipo francés.



The Continent's Number One Infosec Event
#INFOSEC17
Register Now ▶

The Register®

Biting the hand that feeds IT

DATA CENTRE SOFTWARE SECURITY TRANSFORMATION DEVOPS BUSINESS PERSONAL TECH SCIENCE

EMERGENT TECH BOOTNOTES

Security

'First ever' SHA-1 hash collision calculated. All it took were five clever brains... and 6,610 years of processor time

Tired old algo underpinning online security must die now



23 Feb 2017 at 18:33, John Leyden, Thomas Claburn and Chris Williams



Google researchers and academics have today demonstrated it is possible – following years of number crunching – to produce two different documents that have the same SHA-1 hash signature.

This proves what we've long suspected: that SHA-1 is weak and can't be trusted. This is bad news because the SHA-1 hashing algorithm is used across the internet, from Git repositories to file deduplication systems to HTTPS certificates used to protect online banking and other websites. SHA-1 signatures are used to prove that blobs of data – which could be software source code, emails, PDFs, website certificates, etc – have not been tampered with by miscreants, or altered in any other way.

Now researchers at CWI Amsterdam and bods at Google have managed to alter a PDF without changing its SHA-1 hash value. That makes it a lot easier to pass off the meddled-with version as the legit copy. You could alter the contents of, say, a contract, and make its hash match that of the original. Now you can trick someone into thinking the tampered copy is the original. The hashes are completely the same.

https://www.theregister.co.uk/2017/02/23/google_first_sha1_collision/

[Twitter](#) [Facebook](#) [8+](#) [Link](#)

Specifically, the team has successfully crafted what they say is a practical technique to generate a SHA-1 hash collision. As a hash function, SHA-1 takes a block of information and produces a short 40-character summary. It's this summary that is compared from file to file to see if anything has changed. If any part of the data is altered, the hash value should be different. Now, in the wake of the research revealed today, security mechanisms and defenses still relying on the algorithm have been effectively kneecapped.



Google's illustration how changes made to a file can sneak under the radar by not changing the hash value

The gang spent two years developing the technique. It took 9,223,372,036,854,775,808 SHA-1 computations, 6,500 years of CPU time, and 110 years of GPU time, to get to this point. The team is made up of Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam), Ange Albertini (Google), and Yarik Markov (Google), and their paper on their work can be found here [PDF]. Its title is: "The first collision for full SHA-1."

For all the gory details, and the tech specs of the Intel CPU and Nvidia GPU number-crunchers used, you should check out the team's research paper. On a basic level, the collision-finding technique involves breaking the data down into small chunks so that changes, or disturbances, in one set of chunks is countered by twiddling bits in other chunks. A disturbance vector [PDF] is used to find and flip the right bits.

A description of Google's SHA-1 colliding PDFs can be found here. We note that the files essentially each contain a large JPEG, and the hash collision is focused on that image data. We also note that you don't have to burn another few thousand years of CPU and GPU time to create more SHA-1 collisions for simple files: thanks to Google's computations, and quirks of the PDF file format, you can from here on out produce PDFs that are visually different but still have the same SHA-1 hash value. This online tool that popped up today will easily help you create colliding PDF files.

In other words, it is now trivial for anyone to alter PDFs, webpages, and certain other simple documents, and keep the SHA-1 hash values the same, thanks to Google and co's research.

SHA-1

Sunny View School
The Costa del Sol's Leading Private British School

Free IT Service Desk Tool
Handle your IT Support better. Start with a Free Service Desk!
freshservice.com/free-

Spotlight

myspace
Search
Discover
Featured
Music

Speaking in Tech: A chat with Web 2.0 MySpace worm dude Samy Kamkar



The Register's guide to protecting your data when visiting the US



SHA1 collider

Quick-and-dirty PDF maker using the collision from the SHAttered paper.

Choose two image files (must be JPG, roughly the same aspect ratio). For now, each file must be less than 64kB.

Aspect ratio 512 x 512

Seleccionar archivo: nada seleccionado
Seleccionar archivo: nada seleccionado

Enviar

(this one is actually even simpler -- just a single comment with the entire first file in it, hence the 64k limit)

<https://alf.nu/SHA1>

Complaints to @steike.



We have broken SHA-1 in practice.

This industry cryptographic hash function standard is used for digital signatures and file integrity verification, and protects a wide spectrum of digital assets, including credit card transactions, electronic documents, open-source software repositories and software updates.

It is now practically possible to craft two colliding PDF files and obtain a SHA-1 digital signature on the first PDF file which can also be abused as a valid signature on the second PDF file.

For example, by crafting the two colliding PDF files as two rental agreements with different rent, it is possible to trick someone to create a valid signature for a high-rent contract by having him or her sign a low-rent contract.

Infographic | Paper

Collision attack: same hashes

	Sha-1	
Good doc	3713.42	!
Bad doc	3713.42	!

Attack proof

Here are two PDF files that display different content, yet have the same SHA-1 digest.



PDF 1 | PDF 2

File tester

Upload any file to test if they are part of a collision attack. Rest assured that we do not store uploaded files.

Drag some files here

Or, if you prefer..

Choose a file to upload

<https://shattered.io>

Hola a tod@s

Hola a tod@s



2 PDFs (a.pdf y b.pdf) con el resultado de la colisión



We have broken SHA-1 in practice.

This industry cryptographic hash function standard is used for digital signatures and file integrity verification, and protects a wide spectrum of digital assets, including credit card transactions, electronic documents, open-source software repositories and software updates.

It is now practically possible to craft two colliding PDF files and obtain a SHA-1 digital signature on the first PDF file which can also be abused as a valid signature on the second PDF file.

For example, by crafting the two colliding PDF files as two rental agreements with different rent, it is possible to trick someone to create a valid signature for a high-rent contract by having him or her sign a low-rent contract.

Infographic | Paper

Collision attack: same hashes

	Sha-1	
Good doc	3713.42	!
Bad doc	3713.42	!

Attack proof

Here are two PDF files that display different content, yet have the same SHA-1 digest.



PDF 1 | PDF 2

File tester

Upload any file to test if they are part of a collision attack. Rest assured that we do not store uploaded files.

Collision found in a.pdf
Collision found in b.pdf

SHA-1

- Según <https://shattered.io>, se puede crear colisiones a diferentes tipos de artefactos:
 - Digital Certificate signatures
 - Email PGP/GPG signatures
 - Software vendor signatures
 - Software updates
 - ISO checksums
 - Backup systems
 - Deduplication systems
 - GIT (a version control system (VCS) for tracking changes in computer files and coordinating interactive work based on files)
 - ...

– SHA-2

- Es en la actualidad una familia de cuatro algoritmos hash: *SHA-224*, *SHA-256*, *SHA-384* y *SHA-512*
- *SHA-224* (resp. *SHA-384*) es una variante de *SHA-256* (resp. *SHA-512*), en los que se trunca la salida
- Para *SHA-224/SHA-256* (resp. *SHA-384/SHA-512*) se han encontrado algunas colisiones reducidas

– SHA-3

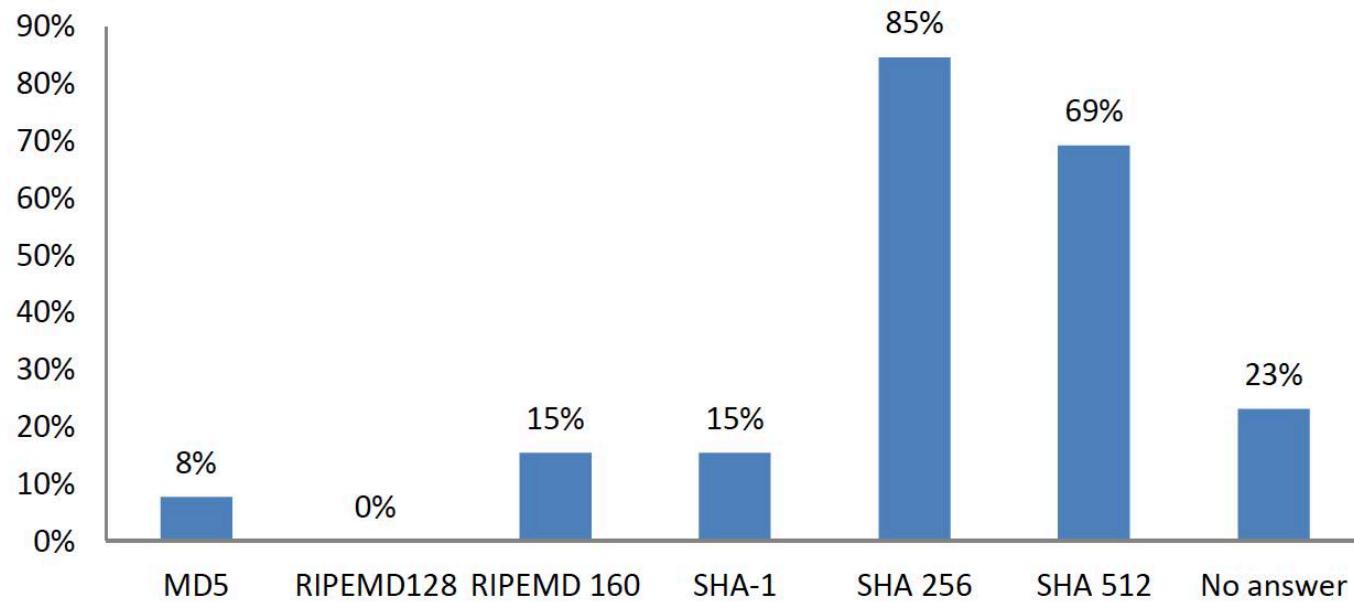
- La competición organizada por *NIST* para elegir el estándar *SHA-3* finalizó en Octubre de 2012, seleccionando el algoritmo *Keccak*

– Whirlpool

- Produce una salida de 512 bits, y no pertenece a la familia *MD-X*
- Se construye a partir de métodos similares a los usados en *AES*, y es una buena alternativa de uso para garantizar la diversidad de algoritmos

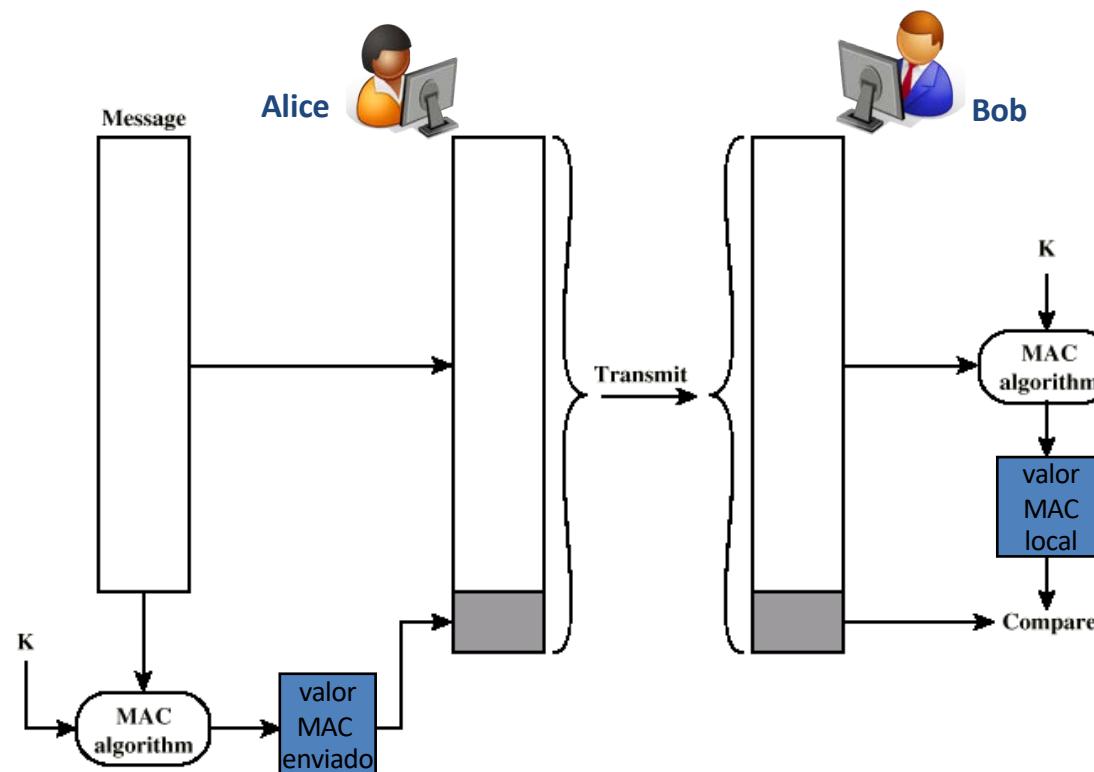
- Comentarios generales:
 - El desarrollo de las funciones hash es, probablemente, el área de la Criptografía que ha atraído más atención durante la pasada década
 - debido a las mejoras realizadas en el criptoanálisis de funciones hash, así como por la competición del *SHA-3* para diseñar un sustituto al conjunto de funciones existentes
 - La mayoría de las funciones hash existentes derivan mucho de los criterios de diseño de la función hash *MD-4* (familia *MD-X*)
 - Esta familia incluye *MD-4*, *MD-5*, *RIPEMD-128*, *RIPEMD-160*, *SHA-1* y *SHA-2*
 - Las salidas de las funciones hash deberían ser como mínimo de 160 bits de longitud para las aplicaciones heredadas, y de 256 bits para todas las aplicaciones nuevas

Primitive	Output Length	Recommendation Legacy	Recommendation Future
SHA-2	256, 384, 512	✓	✓
SHA-3	?	✓	✓
Whirlpool	512	✓	✓
SHA-2	224	✓	✗
RIPEMD-160	160	✓	✗
SHA-1	160	✓	✗
MD-5	128	✗	✗
RIPEMD-128	128	✗	✗



Funciones MAC

- Un código de autenticación de mensaje, o función MAC, es un concepto que evoluciona a partir del concepto de función hash
- Concretamente, una función MAC toma como entrada un mensaje M y una clave K , y produce un valor hash (que en este caso se denomina **valor MAC**)
- Ejemplo de uso:



- El esquema anterior soluciona el problema anteriormente mencionado de que la función hash, aunque proporciona integridad de datos, no proporcionan autenticación
 - Ahora la autenticación se consigue porque *Alice* y *Bob* comparten la clave simétrica K
 - *Bob* está convencido de que la información proviene de *Alice* porque es la única (además de él) que conoce K
- Las funciones MAC más conocidas entran en dos categorías:
 - Basadas en funciones hash (por ejemplo: *HMAC*, *UMAC*)
 - Basadas en algoritmos de cifrado en bloque (por ejemplo: *EMAC*, *AMAC*, *CMAC*)

Referencias bibliográficas

Bibliografía básica

- “*Cryptography and Network Security: Principles and Practice*”
William Stallings
Prentice Hall, 2014 (6^a edición)
- "Handbook of Applied Cryptography"
Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone,
CRC Press, 1996
- “Applied Cryptography: Protocols, Algorithms, and Source Code in C”
Bruce Schneier
Wiley, 1996 (2^a edición)

Referencias

- “*Data Encryption Standard (DES)*”, FIPS PUB 46-3, NIST, 1999
- “*Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*”, NIST, 2012
- “*Advanced Encryption Standard (AES)*”, Federal Information Processing Standards Publication 197, NIST, 2001
- “*Recommendation for Block Cipher Modes of Operation Methods and Techniques*”, NIST, 2001
- “*Record Breaking DES Key Search Completed*”
<http://www.cryptography.com/technology/applied-research/research-efforts/des-key-search.html>

Otras referencias

- “The Washington-Moscow Hot Line. A Compilation of Extracts”, edited by Jerry Proc
 - <http://www.jproc.ca/crypto/hotline.html>