

SEGURIDAD DE LA INFORMACIÓN

TEMA 2

**TÉCNICAS CRIPTOGRÁFICAS BÁSICAS
(Y SERVICIOS DE SEGURIDAD ASOCIADOS)**

Indice del tema (I)

- Introducción a la criptografía clásica
 - Cifrados por sustitución y transposición. Ejemplos
 - Cifrado producto
 - Cifrado Vernam (one-time pad)
- Algoritmos simétricos
 - Fundamentos
 - Algoritmo DES
 - Algoritmo triple-DES
 - Algoritmo AES
 - Otros algoritmos simétricos
 - Modos de operación para algoritmos simétricos
 - Ventajas y desventajas de los algoritmos simétricos

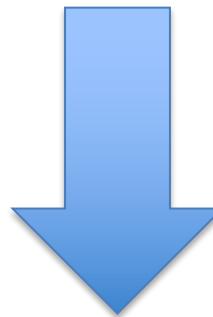
Indice del tema (II)

- Algoritmos asimétricos (o de clave pública)
 - Cifrado/descifrado
 - Firma Digital
 - Intercambio de Claves
 - Algoritmo de Diffie-Hellman
 - Algoritmo RSA
- Otras primitivas criptográficas
 - Funciones hash
 - Códigos de autenticación de mensajes
- Referencias bibliográficas

Introducción a la criptografía

Criptografía, Criptoanálisis, Criptología

- Ya se sabe por el tema anterior que un **algoritmo de cifrado** es un mecanismo fundamental para el desarrollo de servicios de seguridad, como puede ser la confidencialidad



- Criptografía: ciencia que estudia cómo mantener la seguridad en los mensajes (M)
 - usando, entre otros mecanismos, los algoritmos de cifrado
- Criptoanálisis: ciencia que estudia cómo romper los textos cifrados
- Criptología: Criptografía + Criptoanálisis

- El algoritmo de **cifrado** es un mecanismo que transforma un texto en claro en texto ininteligible
 - Su objetivo es dar cobertura al servicio de CONFIDENCIALIDAD
 - El **ALGORITMO DE CIFRADO**, caracterizado por **E** (del inglés “encrypt”), opera sobre el **texto en claro M** (mensaje) para producir el **texto cifrado C** (criptograma)



- La transformación inversa de un texto cifrado a un texto en claro, se denomina ALGORITMO DE DESCIFRADO
 - El algoritmo se denota por la letra D (“decrypt”) y opera sobre C para producir el mensaje M

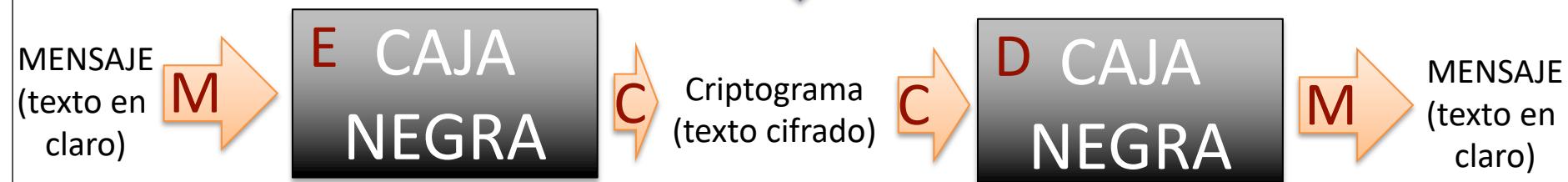


- Se cumple también que:

$$D(C) = M$$
$$D(E(M)) = M$$

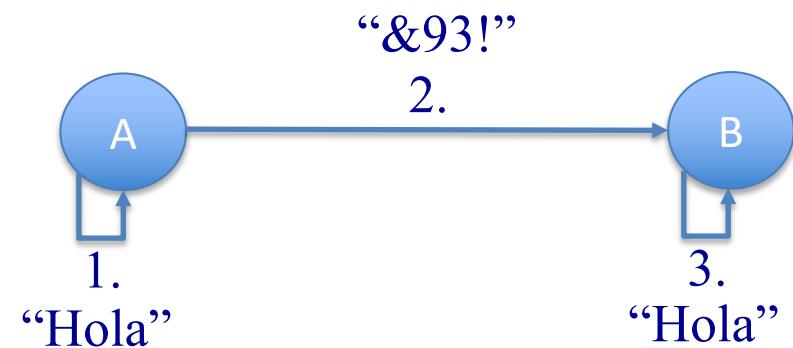
$$D(E(M)) = M$$

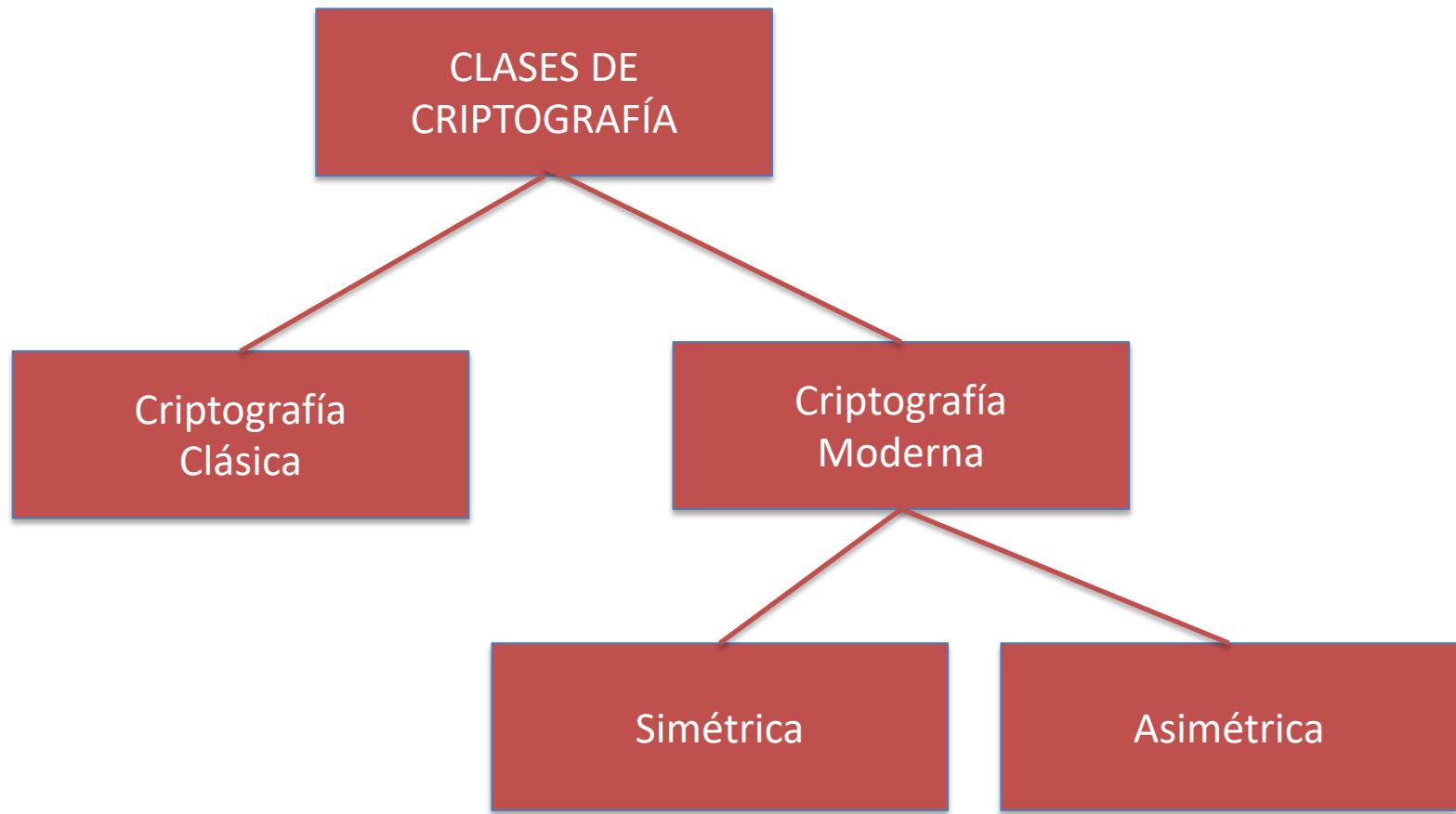
Otra forma
de verlo es



- Ejemplo:

1. A: "Hola"
2. A → B: E("Hola") → "&93!"
3. B: D("&93!") → "Hola"





Criptografía clásica



- Antes de la existencia de ordenadores, la criptografía clásica consistía en algoritmos basados en caracteres
- Estos algoritmos se basaban en dos técnicas principales:
 - **Cifrado por sustitución:**
 - Cada carácter del texto en claro se sustituye por otro carácter en el texto cifrado
 - $A \rightarrow V$
 - $V \rightarrow W$
 - ...
 - **Cifrado por transposición:**
 - Realizar una permutación con respecto a las posiciones que ocupan los símbolos en el mensaje en claro
 - HOLA \rightarrow ALHO

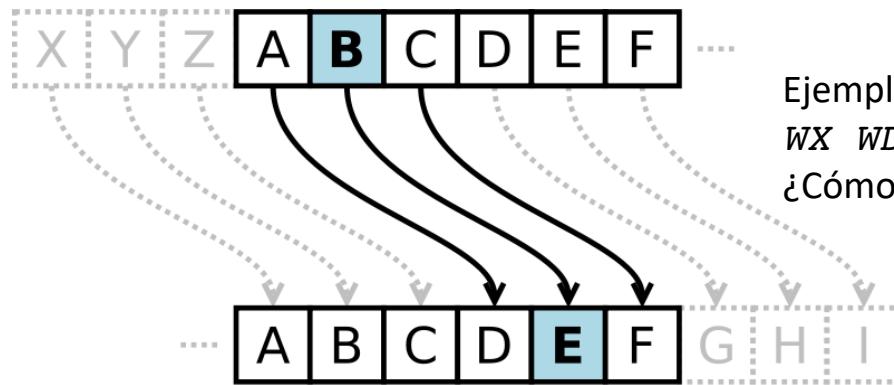
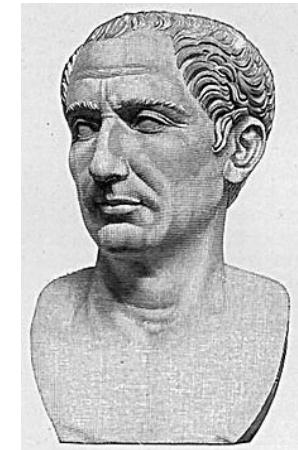
- Antes de la existencia de ordenadores, la criptografía clásica consistía en algoritmos basados en caracteres
- Estos algoritmos se basaban en dos técnicas principales:
 - **Cifrado por sustitución:**
 - Cada carácter del texto en claro se sustituye por otro carácter en el texto cifrado
 - $A \rightarrow V$
 - $V \rightarrow W$
 - ...
 - **Cifrado por transposición:**
 - Realizar una permutación con respecto a las posiciones que ocupan los símbolos en el mensaje en claro
 - HOLA \rightarrow ALHO

VAMOS A VER ALGUNOS EJEMPLOS

Ejemplo 1: cifrado por sustitución César

- Objetivo:
 - Cada carácter de texto en claro se reemplaza por aquel posicionado a tres posiciones a la derecha (módulo 27)

$$C: M \rightarrow M + 3 \text{ (mod 27)}$$



Ejemplo texto cifrado:

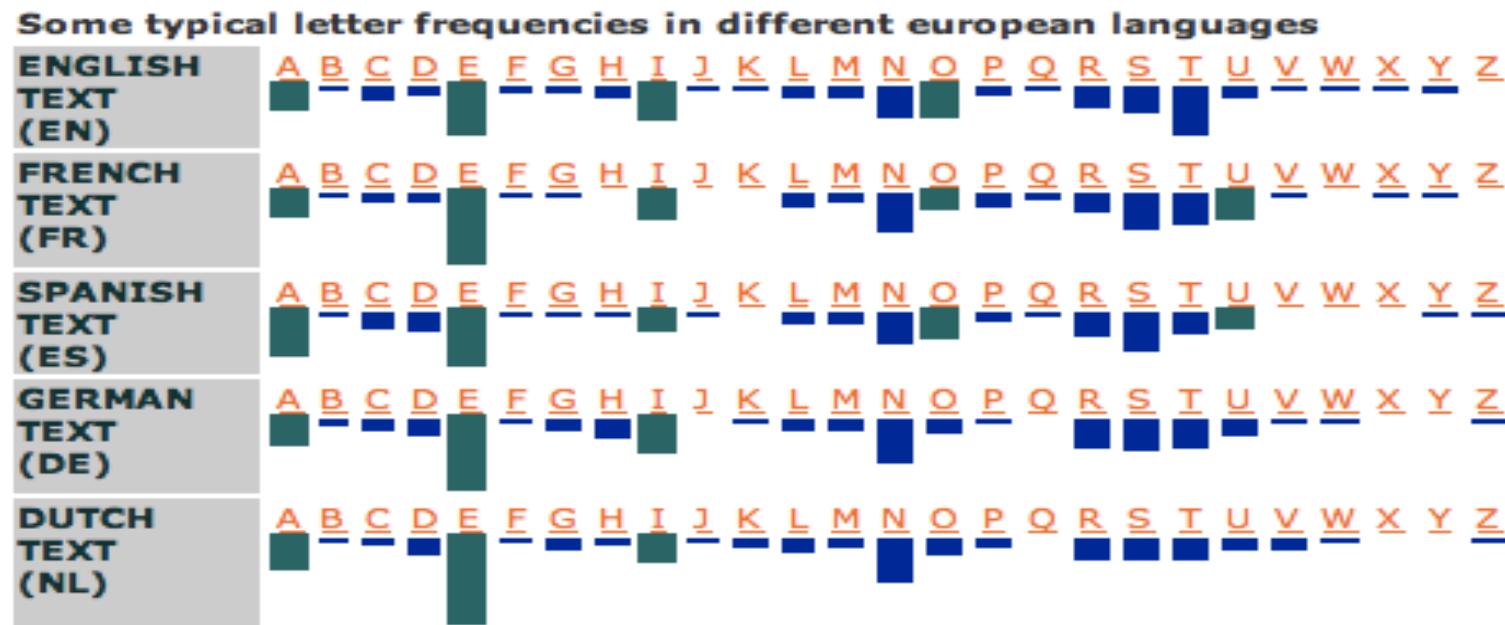
WX WDPHELHQ, EUXWR, KLMR PLR

¿Cómo sería el descifrado de este texto cifrado?

- Se puede generalizar a un sistema de cifrado con 27 posibles combinaciones

$$C: M \rightarrow M + i \text{ (mod 27)} \quad 1 \leq i \leq 27$$

- El algoritmo proporciona ventajas al criptoanalista, porque la frecuencia de aparición de las letras es bien conocida. Así:



English

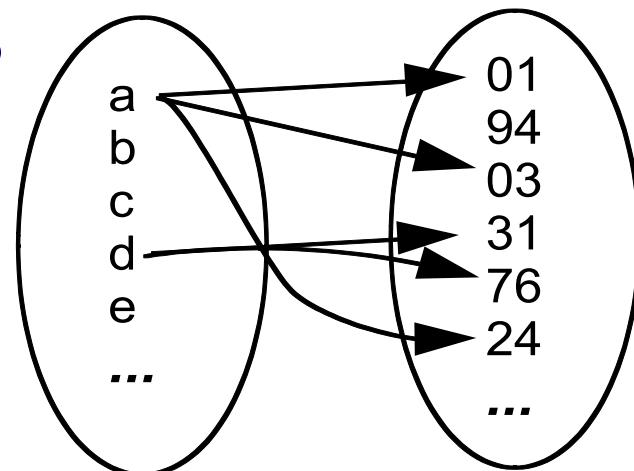
E	12.4%	H	6.5%	U	2.7%	G	2.0%	K	0.7%
T	8.9%	S	6.2%	M	2.5%	Y	2.0%	Q	0.1%
A	8.0%	R	6.1%	W	2.3%	P	1.6%	X	0.1%
O	7.6%	D	4.6%	C	2.2%	B	1.3%	J	0.1%
N	7.0%	L	3.6%	F	2.2%	V	0.8%	Z	0.0%
I	6.7%								

Spanish

E	13.0%	S	6.9%	U	3.6%	V	1.0%	J	0.3%
A	11.1%	T	5.3%	P	3.0%	F	0.8%	Z	0.3%
O	9.7%	C	5.2%	M	2.9%	Y	0.7%	X	0.2%
I	8.2%	D	4.5%	G	1.4%	H	0.6%	W	0.1%
N	8.0%	L	3.6%	B	1.3%	Q	0.6%	K	0.0%
R	7.7%								

Ejemplo 2: cifrado por sustitución homofónico

- Se basa en la idea de asignar a un símbolo del alfabeto fuente varios del alfabeto cifrado, solventando el problema de la frecuencia de letras
 - Correspondencia uno a muchos ⇒ al cifrar un mensaje podemos obtener varios criptogramas
 - Ejemplo:



Letra	% (redondeado)	Símbolos asignados
A	8	10, 11, 23, 45, 76, 79, 87, 98
L	6	02, 15, 21, 25, 56, 60
N	3	44, 63, 71
O	8	04, 16, 28, 29, 37, 52, 69, 90
P	2	30, 88
T	2	24, 77

“PLATON” se cifra como “882110772963”

Ejemplo 3: cifrado por sustitución POLIalfabética

- Alfabeto para posiciones impares:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
V	K	L	s	w	e	M	N	U	f	a	b	Q	r	S	t	o	j	I	P	x	s	Ñ	h	d	Z	W

- Alfabeto para posiciones pares:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
z	g	X	Y	a	b	D	K	L	P	q	s	t	U	O	Ñ	Q	k	e	c	H	W	M	N	f	g	i

- Cifrado del texto: “*HOLA A TODOS*”

H O L A A T O D O S

- #### • Descifrado:

Ejemplo 3: cifrado por sustitución POLIalfabética

- Alfabeto para posiciones impares:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
V	K	L	s	w	e	M	N	U	f	a	b	Q	r	S	t	o	j	I	P	x	s	Ñ	h	d	Z	W

- Alfabeto para posiciones pares:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
z	g	X	Y	a	b	D	K	L	P	q	s	t	U	O	Ñ	Q	k	e	c	H	W	M	N	f	g	i

- Cifrado del texto: “**HOLAA TODOS**”

H	O	L	A	A	T	O	D	O	S
N	Ñ	b	z	V	H	t	Y	t	c

- Descifrado:

N	Ñ	b	z	V	H	t	Y	t	c
H	O	L	A	A	T	O	D	O	S

- Antes de la existencia de ordenadores, la criptografía clásica consistía en algoritmos basados en caracteres
- Estos algoritmos se basaban en dos técnicas principales:
 - **Cifrado por sustitución:**
 - Cada carácter del texto en claro se sustituye por otro carácter en el texto cifrado
 - $A \rightarrow V$
 - $V \rightarrow W$
 - ...
 - **Cifrado por transposición:**
 - Realizar una permutación con respecto a las posiciones que ocupan los símbolos en el mensaje en claro
 - HOLA \rightarrow ALHO

VAMOS A VER ALGUNOS EJEMPLOS

Ejemplo 4: cifrado por transposición

- Objetivo: el texto en claro se escribe como secuencia de filas (con una cierta profundidad) y se lee como secuencia de columnas

Restricción
a nivel de
fila



H	O	L	A	M	U	N
D	O	Y	A	S	Í	C
O	N	T	O	D	O	.

- Ejemplo:
 - “EN ANDALUCIA, EL MULHACEN Y EL VELETA, SON LAS MONTAÑAS MAS ALTAS”



ENANDALUCIAELMULHACENYELVE
LETASONLASMONTAÑASMASALTAS

Hay que quitar los espacios,
los símbolos, etc.

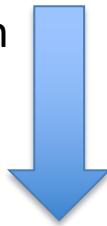
- Mensaje cifrado:

ELNEATNADSAOLNULCAISAMEOLNMTUALÑHAASCMEANSYAELLTVAES

Ejemplo 5: cifrado por transposición con CLAVE

- Se podría complicar el procedimiento anterior estableciendo una restricción en el número de columnas cuyo valor va a depender del tamaño que tenga una **clave**

Restricción
a nivel de
fila



H	O	L	A	M	U	N
D	O	Y	A	S	Í	C
O	N	T	O	D	O	.



Restricción a nivel de columna

- Ejemplo:
 - Texto en claro: “**HOLA A TODOS, QUE TENGÁIS UN BUEN DÍA**”
 - Clave: ”**SECRETO**” con un tamaño de 7

S	E	C	R	E	T	O

Ejemplo 5: cifrado por transposición con CLAVE

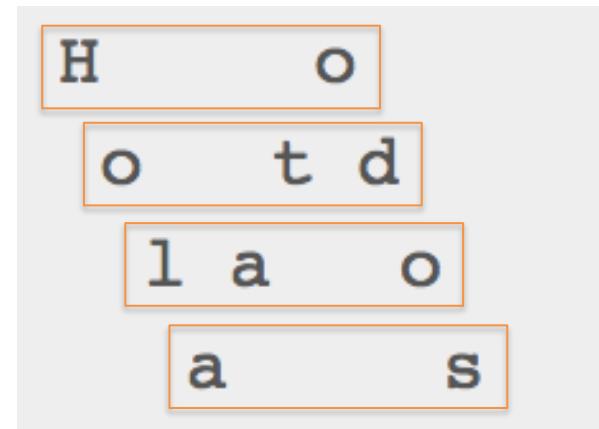
- Ejemplo:
 - Texto en claro: “**HOLA A TODOS, QUE TENGÁIS UN BUEN DÍA**”
 - Clave: ”**SECRETO**” con un tamaño de 7

S	E	C	R	E	T	O
H	O	L	A	A	T	O
D	O	S	Q	U	E	T
E	N	G	A	I	S	U
N	B	U	N	D	I	A

- Podemos fortalecer la seguridad si añadimos más restricciones:
 - Para el cifrado se puede poner la condición siguiente: se va a ir cogiendo las letras de aquellas columnas por orden alfabético del secreto, es decir: **C, E, E, O, R, S, T**, resultando en: “_____”
 - Solución: “**LSGUOONBAUIDOTUAQAQNHDENTESI**”

Ejemplo 6: cifrado por transposición Railfence

- El cifrado consiste
 - en escribir diagonalmente el texto en claro con una profundidad P específica
 - el criptograma se escribe leyendo las filas
- Ejemplo: $M = \text{"Hola a todos"}$, con una profundidad de $P=4$, entonces el criptograma es: **Hoot d laoas**
por simplemente computar:



- Antes de la existencia de ordenadores, la criptografía clásica consistía en algoritmos basados en caracteres
- Estos algoritmos se basaban en dos técnicas principales:
 - **Cifrado por sustitución:**
 - Cada carácter del texto en claro se sustituye por otro carácter en el texto cifrado
 - $A \rightarrow V$
 - $V \rightarrow W$
 - ...
 - **Cifrado por transposición:**
 - Realizar una permutación con respecto a las posiciones que ocupan los símbolos en el mensaje en claro
 - HOLA \rightarrow ALHO

SE PUEDEN COMBINAR

Cifrado Producto

- **Combinación de algoritmos sustitución y transposición**
- Se pueden considerar como la aplicación sucesiva de varios cifrados E_i

$$E = E_1 \cdot E_2 \cdot \cdots \cdot E_r$$

$$E(M) = E_1(E_2(\cdots(E_r(M))))$$

- La composición de funciones de descifrado D_i se realiza en orden inverso

$$D = D_r \cdot D_{r-1} \cdots D_1$$

$$M = D(C) = D_r(D_{r-1}(\dots(D_1(C))))$$

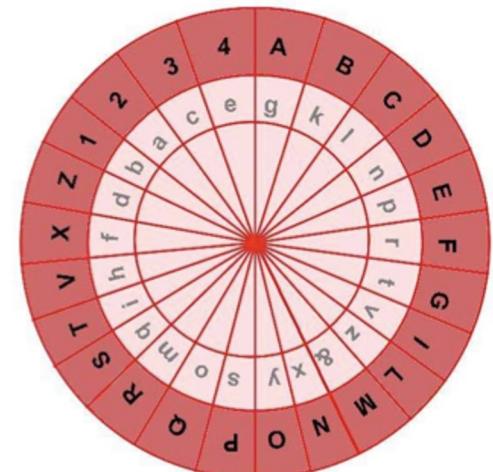
- Es un esquema utilizado para obtener un alto grado de seguridad con sistemas relativamente sencillos aplicados reiterativamente
- Dan lugar a sistemas de cifrado complejos, seguros y difíciles de atacar, así como fácilmente trasladables a un ordenador

Ejemplo 7: métodos polialfabéticos y nomenclátores

- Para complicar el proceso de cifrado, se puede hacer uso del disco de Alberti junto con nomenclátores, los cuales consisten en asociar a determinados palabras códigos específicos

Felipe II	123
Rey	124
Walshingan	122

- Se desea descifrar el siguiente texto: “*baa&hpmiyvsoiyrlxckngkl*”
- Uso del disco y condiciones:
 - Cada diez letras descifradas, se ha de girar el disco externo (de las mayúsculas) dos posiciones en el sentido de las agujas del reloj
 - En el disco de Alberti, la **u** se identifica con la **v** al cifrar
 - Al descifrar, por el sentido de la frase, se puede conocer si se ha de escribir una u otra letra

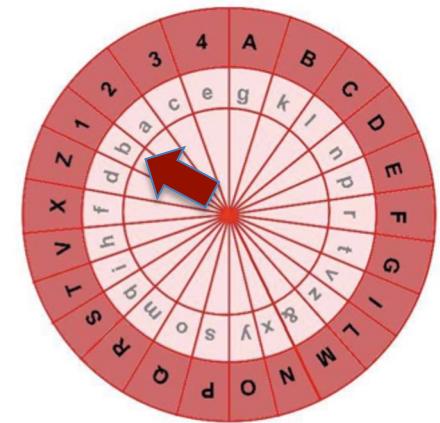


Ejemplo 7: métodos polialfabéticos y nomenclátóres

- Funcionamiento para cifrar:
 - Posicionar los disco en el estado inicial

b	a	a	&	H	p	m	i	Y	V
1	2								

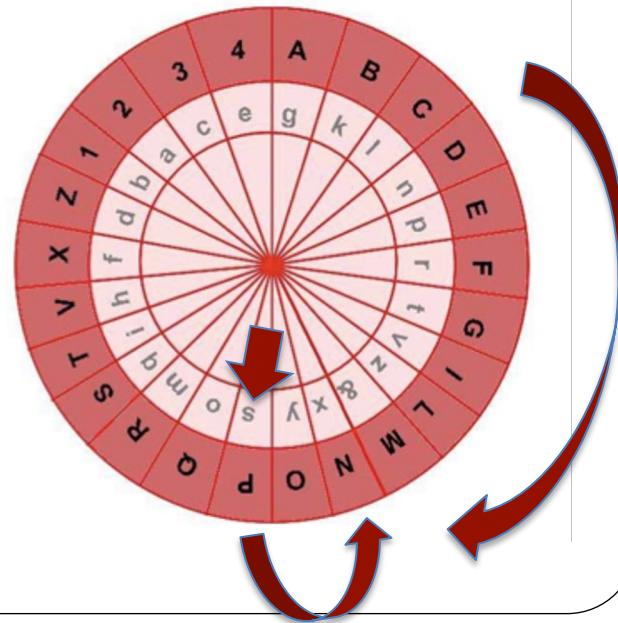
"baa&hpmiyvsvoiylrlxckngkl"



- Con el disco externo girar 2 posiciones en el sentido de las agujas del reloj (sólo en cada diez letras descifrada):

s	v	o	i	Y	l	r	l	x	c
N	F								

"baa&hpmiyvsvoiylrlxckngkl"

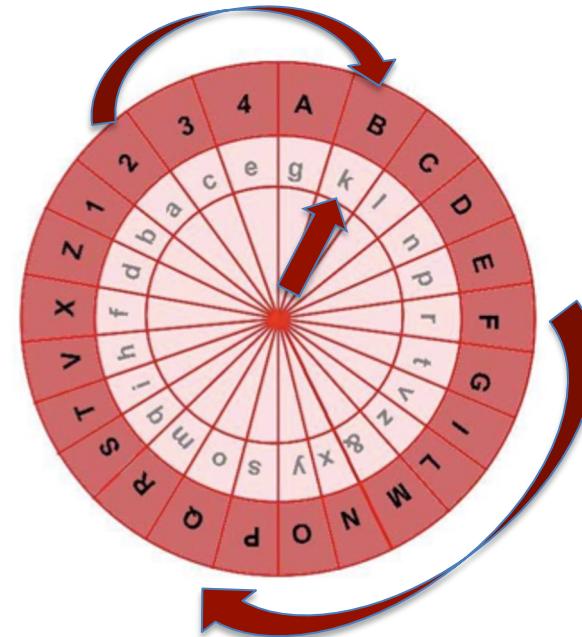


Ejemplo 7: métodos polialfabéticos y nomenclátores

- Con el disco externo volver a girar 2 posiciones en el sentido de las agujas del reloj:

k	n	g	k	L
2	4	1	2	3

"baa&hpmiyvsvoiylrlxckngkl"



- Por consiguiente, el texto en claro es:

b	a	a	&	H	p	m	i	Y	v
1	2	2	M	V	E	R	T	O	I

s	v	o	I	Y	l	r	l	x	c
N	F	O	R	M	A	D	A	L	1

k	n	g	k	L
2	4	1	2	3

"1 2 2 M V E R T O I N F O R M A D A L 1 2 4 1 2 3"

- Si, además, añadimos los nomenclátores + la restricción de la V → U:

Felipe II	123
Rey	124
Walshinan	122

"WALSHINGAN MUERTO INFORMAD AL REY FELIPE II"

Ejemplo 8: Cifrado Vernam

- Aplica el concepto de **one-time pad (OTP)**
- Un one-time pad es un *conjunto infinito y no repetitivo* de letras aleatorias
- Cada letra del pad se usa para cifrar una única letra del texto en claro, en módulo n (longitud del alfabeto)



Texto : T H I S I S S E C R E T
OTP: X V H E U W N O P G C Z

Cifrado : Q C P W C O F S R X H S

One-time pad booklet and microdot reader, concealed in a toy truck and used by an illegal agent that operated in Canada
© Canadian Security Intelligence Service

- Ejemplo:
 - Aquí se observan grupos de tres filas, que se corresponden con texto en claro (en decimal), clave y criptograma

0	321	6	3767	0	7762	6	3123	7	6487	0	267	0	6267	6	7061	
2	184	6	8432	4	6051	8	7931	7	8273	0	3033	0	6233	6	6233	
6	9140	10	3949	9	713	4	0014	4	4679	0	9280	0	5780	0	5780	
2	3777	6	8279	6	5867	0	8709	6	8395	7	6388	7	2387	7	2387	
6	773	411	67	42357	4	7435	6	2133	7	1370	9	551	9	551	9	551
8	5680	0	9338	0	7114	4	5154	1	0428	7	7778	7	7778	7	7778	
6	3075	8	7069	5	8672	7	1527	7	2843	9	370	9	9176	9	9176	
4	7791	0	7884	5	9125	0	0098	6	2782	6	6856	6	776	6	776	
6	1789	841	69	72997	3	1316	3	3422	7	1373	2	7886	2	7886	2	7886
3	1726	50	833	8	2058	2	1727	6	8626	3	1533	7	811	7	811	
4	5760	17	427	7	8213	7	669	7	1130	4	2540	6	6370	6	6370	
1	6276	6	9204	5	0291	9	311	5	6422	7	3372	3	3741	3	3741	
7	7773	2	8366	5	8776	4	6760	7	7613	0	5867	6	6233	6	6233	
1	2344	35	601	9	509	5	2060	5	781	5	5250	7	6652	7	6652	
8	7771	5	3962	4	2474	9	8220	9	4484	5	7361	3	1727	3	1727	
2	73	7	8208	8	76926	3	3976	3	26716	0	3746	4	1483	4	1483	
6	7618	0	621	0	7404	7	8573	6	7230	6	6780	6	7722	6	7722	
8	0001	7	8229	7	3324	0	3881	9	8506	0	0744	2	175	2	175	
1	7429	7	6856	9	8767	7	26796	5	9177	7	3787	6	6233	6	6233	
2	7792	30	642	3	8091	9	6167	9	6323	4	6625	7	3741	7	3741	
3	1722	2	6758	2	6189	7	7779	0	39702	7	50501	7	50501	7	50501	
5	728	73	333	3	0077	1	5832	5	5850	6	5572	6	6728	6	6728	
0	634	25	606	3	2247	8	0011	5	2773	3	3232	2	2773	2	2773	
3	4082	9	8332	3	2214	9	5293	4	7733	3	2773	3	00523	3	00523	

Fuente: <http://www.caslab.cl/che.php>

B	A	R	R	O	Y	C	Á	N	A	B	R	A	V	A
1	0	18	18	15	25	2	0	14	0	1	18	0	22	0
E	D	S	A	S	A	C	E	T	N	I	E	V	E	D
4	3	19	0	19	0	2	4	20	13	8	4	22	4	3
5	3	10	18	7	25	4	4	7	13	9	22	22	26	3
F	D	K	R	H	Y	E	E	H	N	J	V	V	Z	D

Fuente: <http://bit.ly/2cqBu8D>

Cifrado: (carácter del texto en claro + key) + mod 27

Descifrado: (carácter del criptograma - key) + mod 27

- Ejemplo:
 - En los ordenadores, el OTP aleatorio de longitud infinita se combina mediante XOR con el texto en claro. Ejemplo:

Texto en claro	1	1	0	0	1	0	1	1	0	0	0	1	1	0	1	0	0	1	1	1	0	1	1	\oplus
OTP	1	0	0	1	1	0	1	0	1	0	1	1	0	1	0	0	1	1	0	0	1	0	=	
Criptograma	0	1	0	1	0	0	0	1	1	0	1	0	1	1	1	0	1	0	1	0	0	1	\oplus	
OTP	1	0	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	=	
Texto en claro	1	1	0	0	1	0	1	1	0	0	0	1	1	0	1	0	0	1	1	1	0	1	1	

- Inconvenientes del cifrado Vernam:
 - las letras del OTP (o bits si se usa en ordenador) han de generarse aleatoriamente
 - el OTP no se vuelve a usar

Relación de ejercicios

1. Considerando el alfabeto inglés (sin incluir la ñ) y un desplazamiento de 3 posiciones para el proceso de cifrado o descifrado, aplicar la técnica de sustitución Caesar para cifrar el siguiente texto:

“EL PATIO DE MI CASA ES PARTICULAR”

SOLUCIÓN: HO SDWLR GH PL FDVD HV SDUWLFXODU

Relación de ejercicios

2. Dado el criptograma $C = \text{"FMIRZIRMHS E PE EW MKREXYVE HI WIKYVMHEH HI PE MRJSVQEGMSR"}$ descifrar el contenido del mismo, sabiendo, además, que hay que usar la técnica de sustitución Caesar con un desplazamiento de 4 posiciones modulo $n=26$ (Alfabeto inglés)

SOLUCIÓN: BIENVENIDO A LA ASIGNATURA DE SEGURIDAD
DE LA INFORMACIÓN

Relación de ejercicios

3. El siguiente algoritmo aplicará una sustitución monoalfabética, pero esta vez teniendo en cuenta la siguiente regla: $C_i = M_i + K_i \text{ mod } 26$ donde K representa una clave de longitud L . El objetivo es cifrar el texto original usando el alfabeto inglés

¿Cuál sería el criptograma del mensaje $M = \text{"HOLA AMIGOS"}$ usando una clave $K = \text{CIFRA}$?

Nota: se empieza a contar desde la posición 0 (A del alfabeto)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

H	O	L	A	A	M	I	G	O	S
7	14	11	0	0	12	8	6	14	18
C	I	F	R	A	C	I	F	R	A
+2	+8	+5	+17	+0	+2	+8	+5	+17	+0
J	W	Q	R	A	O	Q	L	F	S
9	22	16	17	0	14	16	11	31→5	18

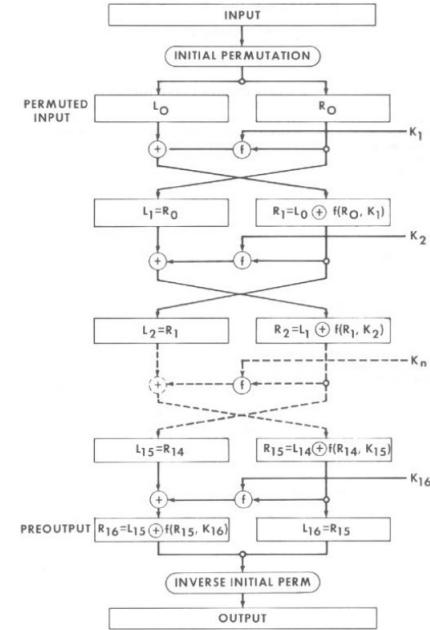
SOLUCIÓN: JWQR AOQLFS

Relación de ejercicios

4. Mediante la técnica Railfence, determinar el criptograma correspondiente al mensaje “*Nos han descubierto, debemos huir*” con una profundidad $P=7$ (alfabeto inglés) y sin contar espacios

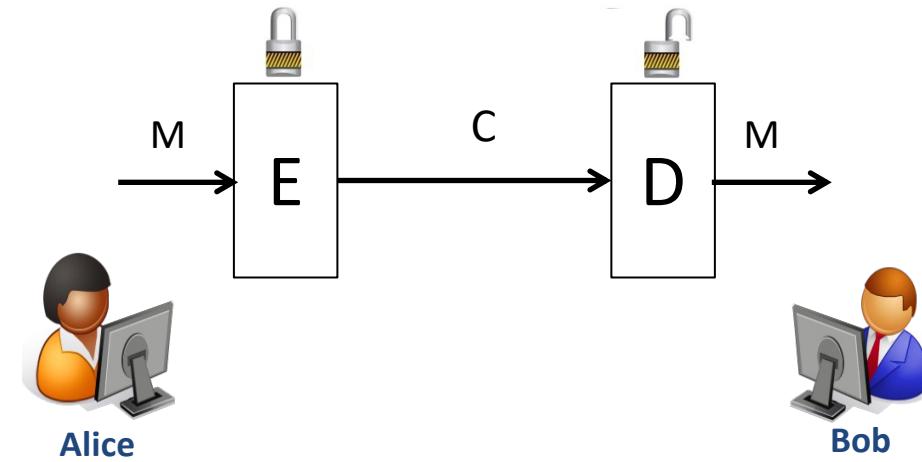
SOLUCIÓN: Nihobesusuroihctmrasoenedbde

Algoritmos simétricos

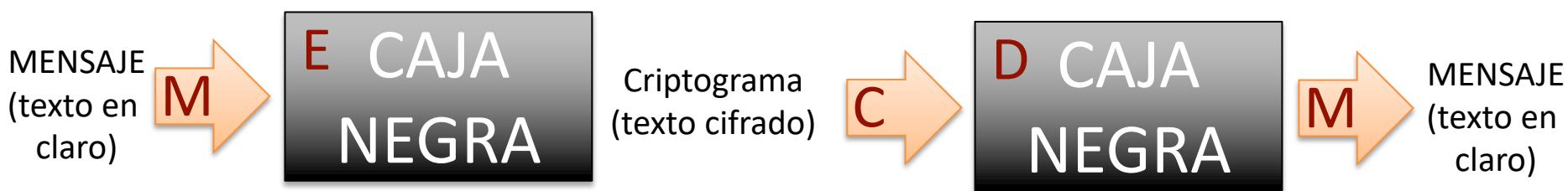


Fundamentos

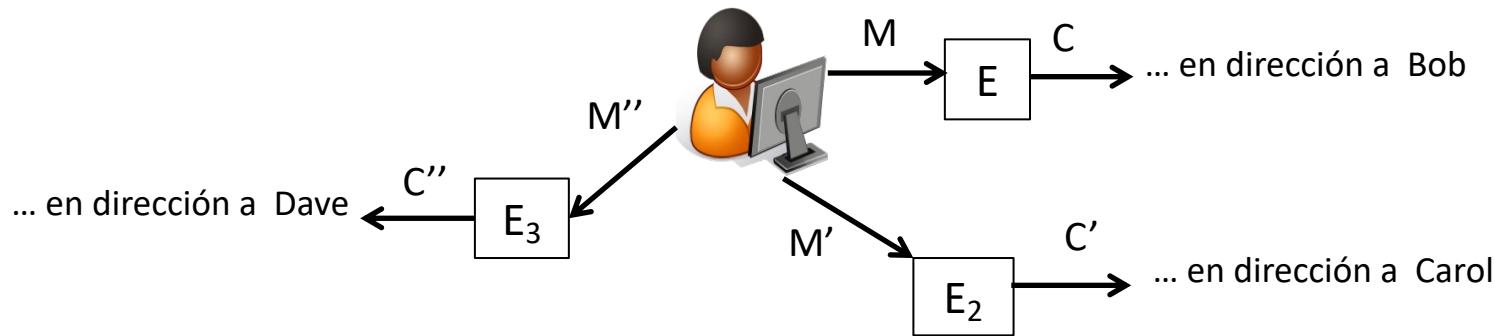
- En la mayoría de los ejemplos de la sección anterior la comunicación entre los usuarios puede representarse como sigue:



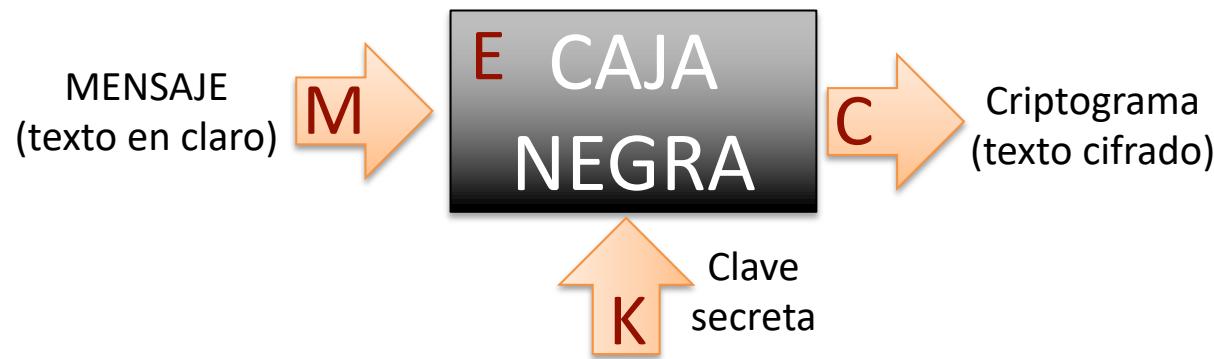
- Equivalente a lo que ya vimos antes:



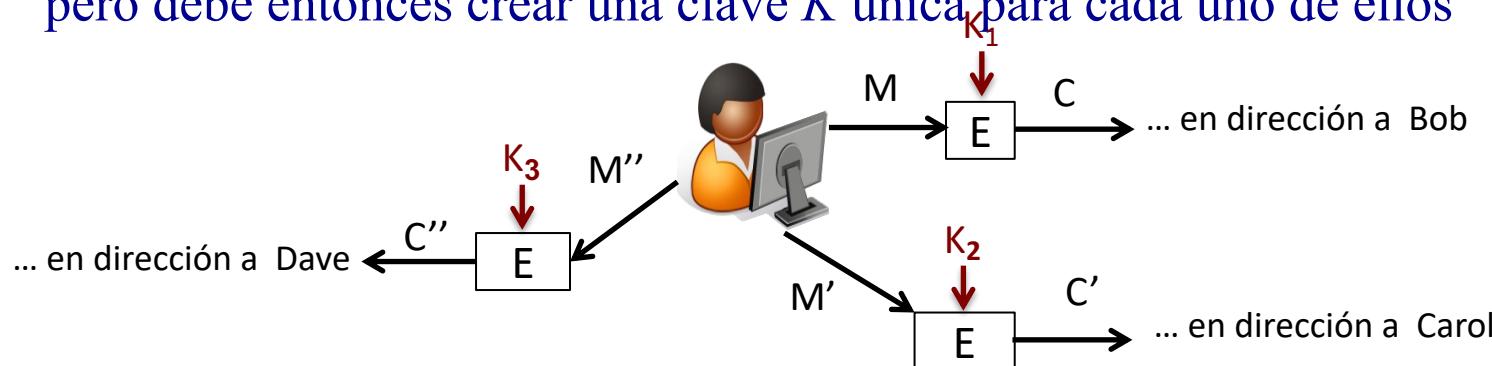
- El esquema anterior es útil siempre que se mantengan en “secreto” la transformación E y su inversa D
 - Esto es factible para un intercambio de información entre dos usuarios específicos (por ejemplo, *Alice* y *Bob*)
 - Sin embargo, esta forma de funcionamiento resulta **no escalable**
 - Cuando *Alice* necesita comunicar con alguien distinto de *Bob*, tendría que usar un algoritmo distinto (o una caja negra distinta) de tipo E como muestra la figura inferior
 - Esto también significa que *Alice* necesitará aplicar un algoritmo distinto (o una caja negra distinta) de tipo E PARA CADA USUARIO con quien contacta



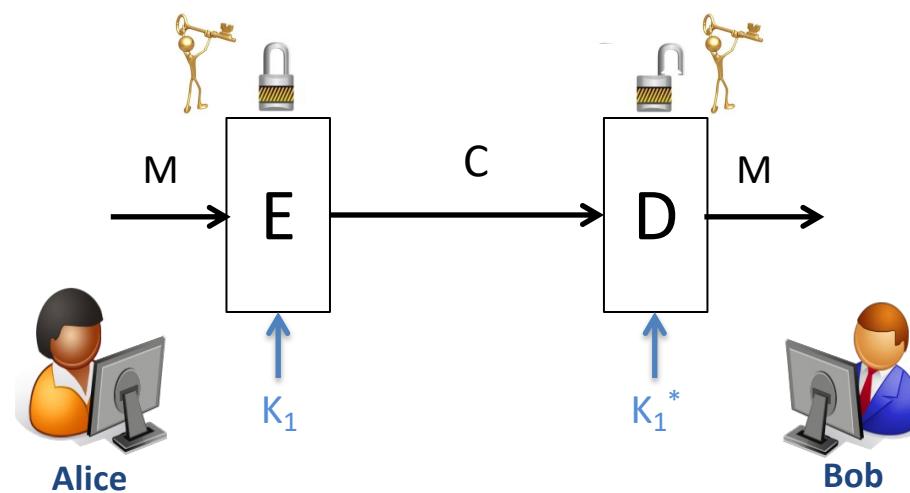
- El problema anterior se puede solucionar aplicando una **CLAVE SECRETA** K y aplicada con el algoritmo de cifrado E



- De esta forma, *Alice* puede usar el mismo algoritmo (o la misma caja negra) E en sus comunicaciones y con todos los usuarios (*Bob*, *Carol*, *Dave*, ...), pero debe entonces crear una clave K única para cada uno de ellos



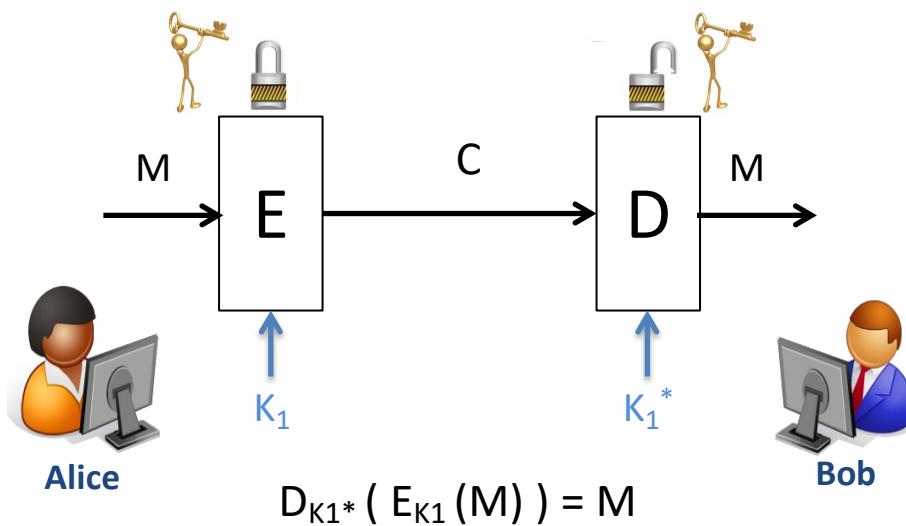
- Por tanto, el mismo algoritmo de descifrado D será usado por todos los receptores, pero cada uno necesitará la clave correspondiente de descifrado ($K_1^*, K_2^*, K_3^* \dots$)
- En resumen, para la comunicación específica entre *Alice* y *Bob*:



$$D_{K1^*} (E_{K1} (M)) = M$$

Recordatorio...

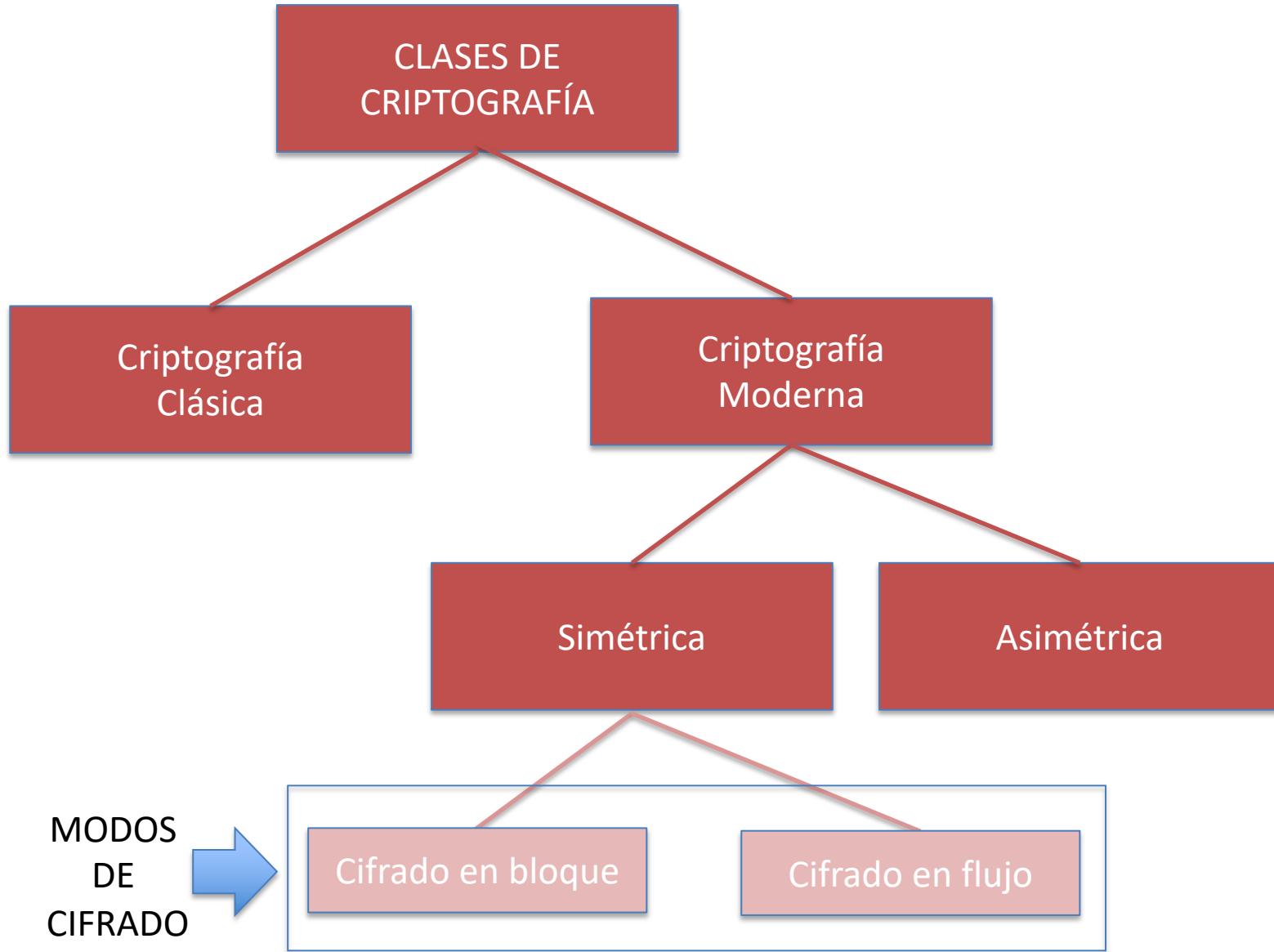
- En esta situación, el mismo algoritmo de descifrado D será usado por todos los receptores, pero cada uno necesitará la clave correspondiente de descifrado (K_1^* , K_2^* , K_3^* ...)
- En resumen, para la comunicación específica entre *Alice* y *Bob*:



- Si los algoritmos E y D son públicos, entonces el secreto se encuentra en K_1 y K_1^*
- **Pero, ¿por qué son los algoritmos públicos? ...**



- Por lo tanto, en las nuevas condiciones anteriores,
es posible hacer públicos los algoritmos E y D
 - De hecho, se pueden evaluar públicamente para detectar posibles fallos
 - En caso de no tener fallos, entonces se pueden introducir en herramientas comerciales, etc.
 - Esto se formaliza en el **segundo principio de Kerckhoffs**:
 - *“The system must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience”*
- Por lo tanto, la seguridad del sistema dependerá finalmente de que Alice y Bob mantengan en secreto las claves secretas K y K^*
 - Los **algoritmos simétricos** son aquellos en los que K y K^* son la misma clave, y se denomina **clave de sesión**
 - En los **algoritmos asimétricos**, las claves K y K^* son distintas



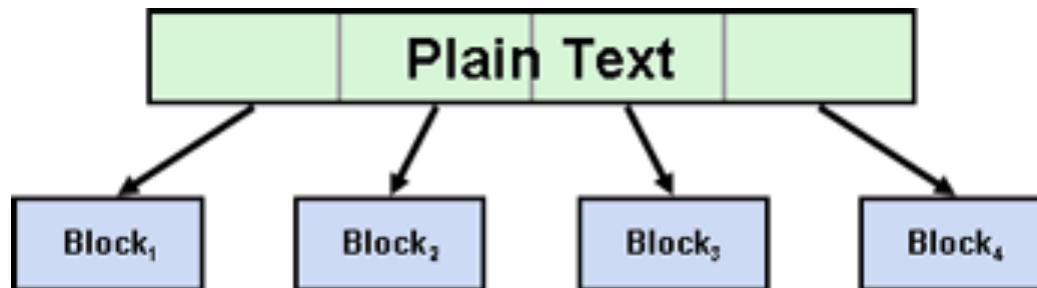
Modos de cifrado - cifrado en bloques

- A partir de la expresión

$$E(M) = C$$

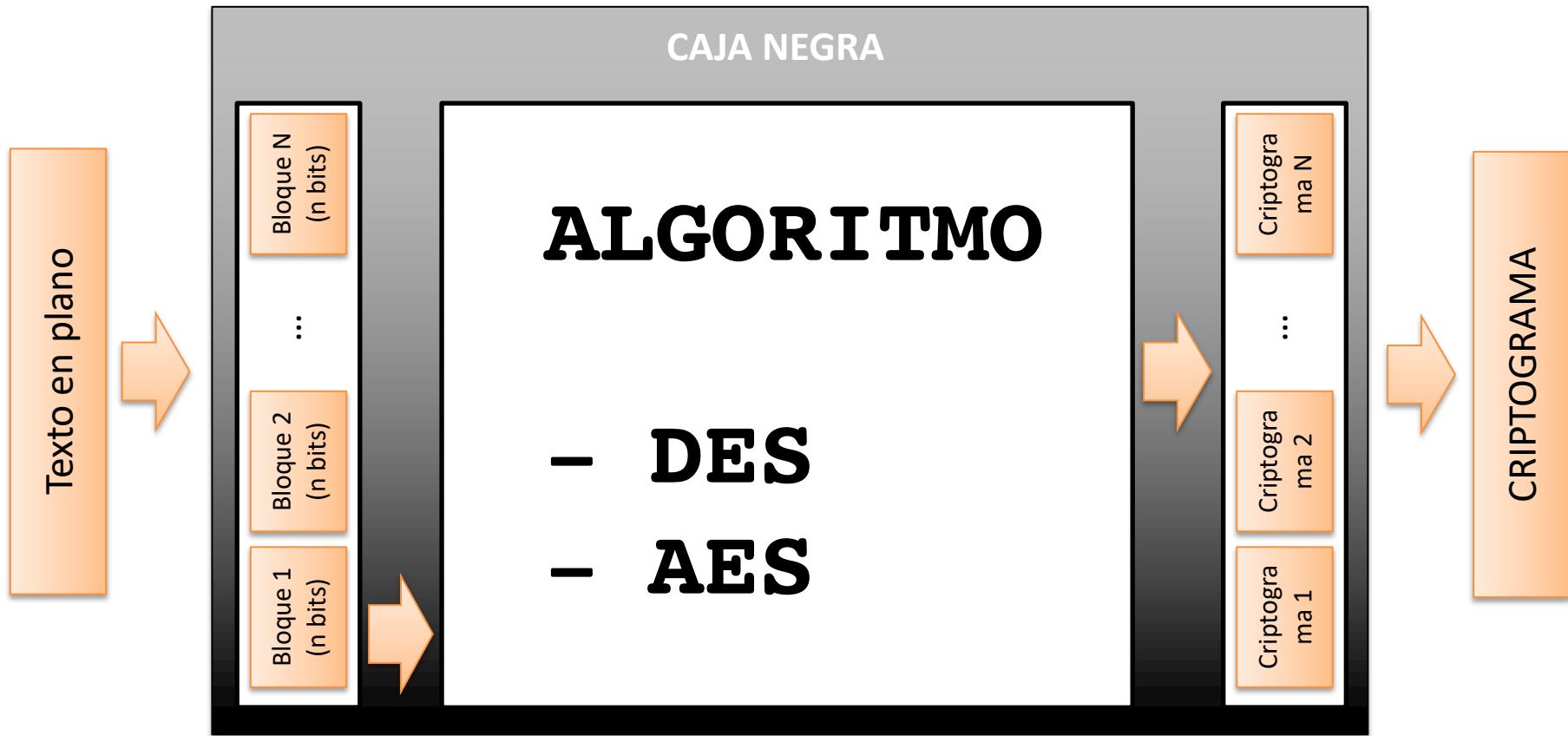
se podría pensar que el algoritmo de cifrado procesa todo el mensaje de una sola vez

- Sin embargo, por cuestiones de diseño, es raro que ocurra eso
- De hecho, son muchos los algoritmos que necesitan procesar el mensaje M en bloques de n bits, denominándose entonces **cifrados en bloque**



- La longitud específica n de los bloques viene determinada por el propio diseño interno del algoritmo

Si lo vemos como cajas negras

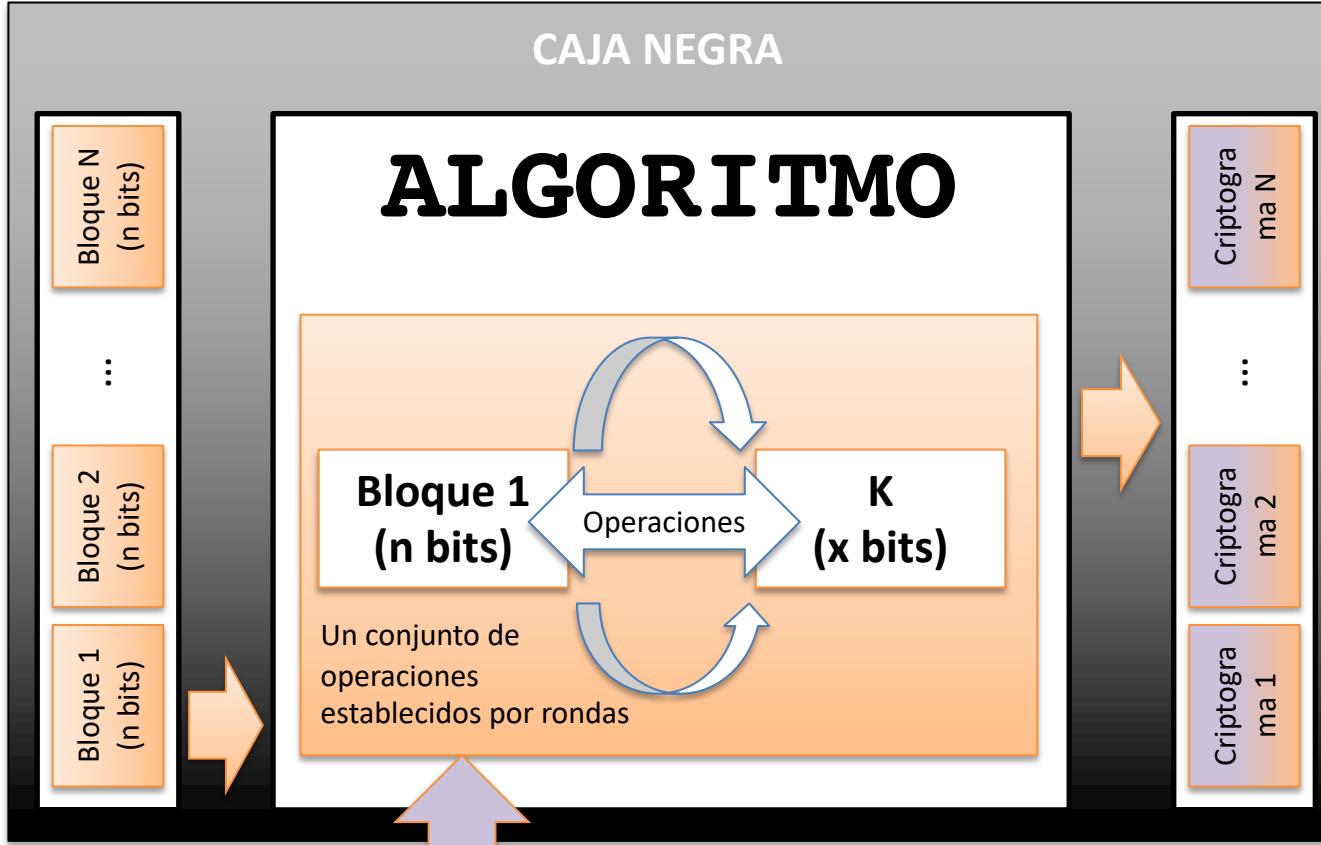


Pero, ¿cuándo se produce el cifrado?

CAJA NEGRA

ALGORITMO

Texto en plano



Se produce el cifrado cuando:

- el bloque de datos se entremezcla con el secreto (es decir, con los datos de la clave K), aplicando la operación **XOR**
- Cuanto más operaciones adicionales haya, más ruido se producirá en el criptograma final y más robusto será éste (frente ataques)

Otra forma de verlo (1)

- Por tanto:

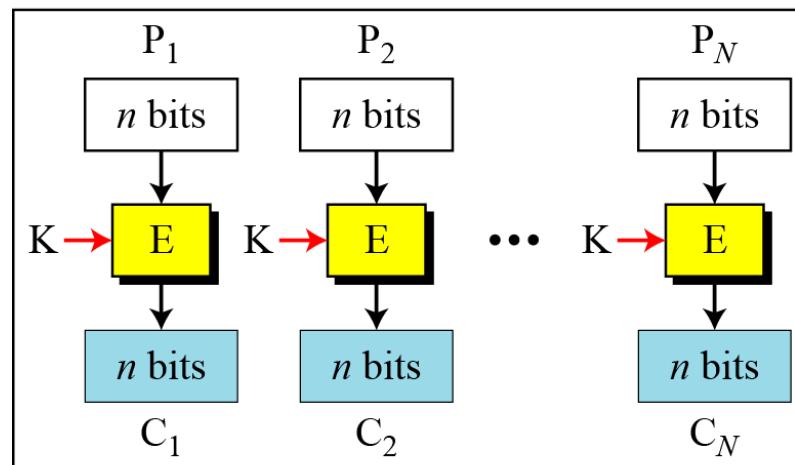
E: Encryption

P_i : Plaintext block i

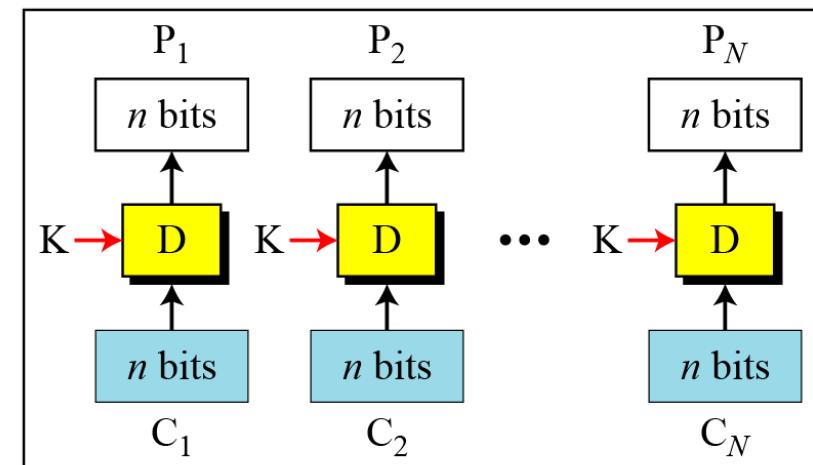
K: Secret key

D: Decryption

C_i : Ciphertext block i

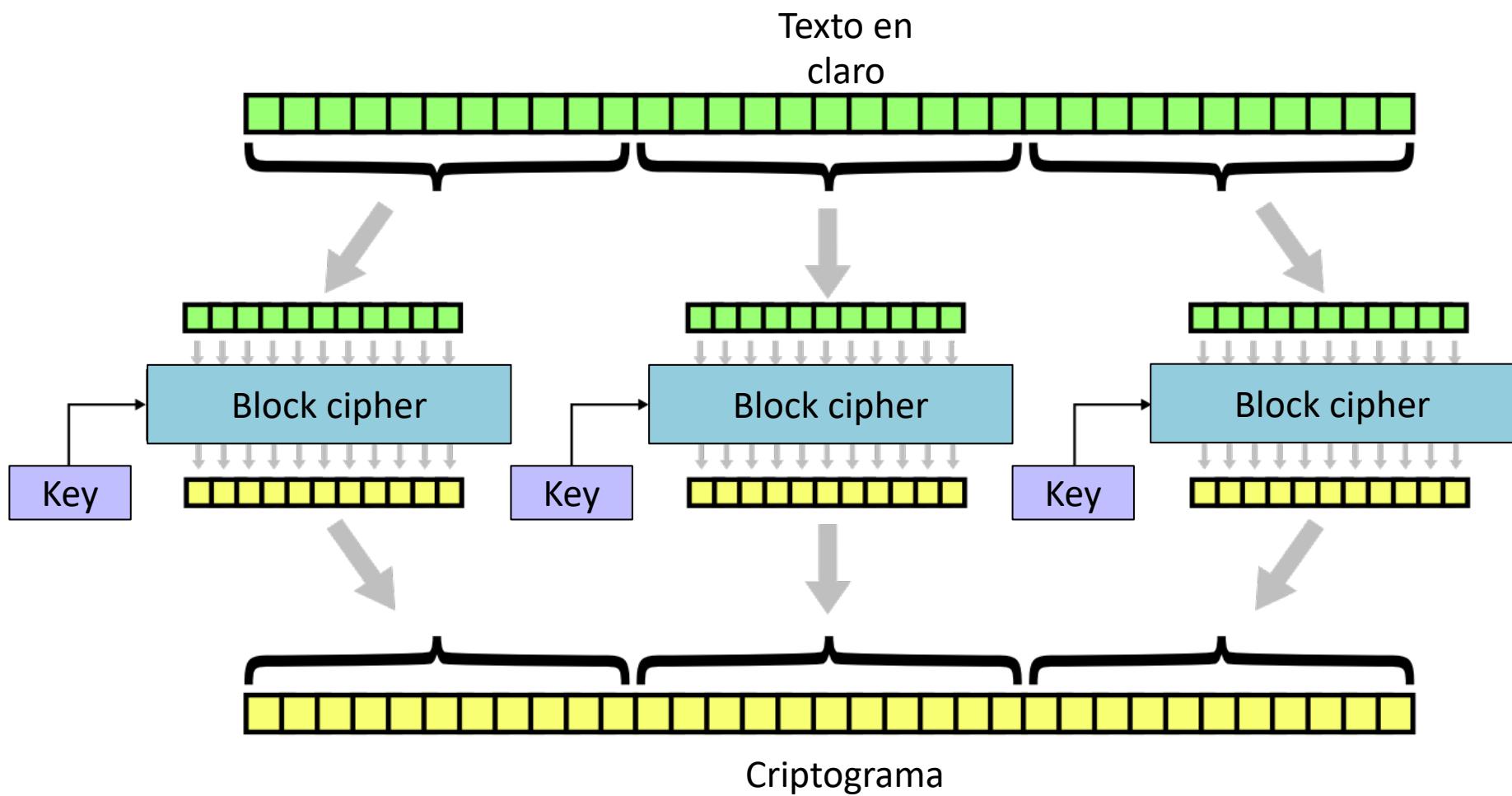


Encryption



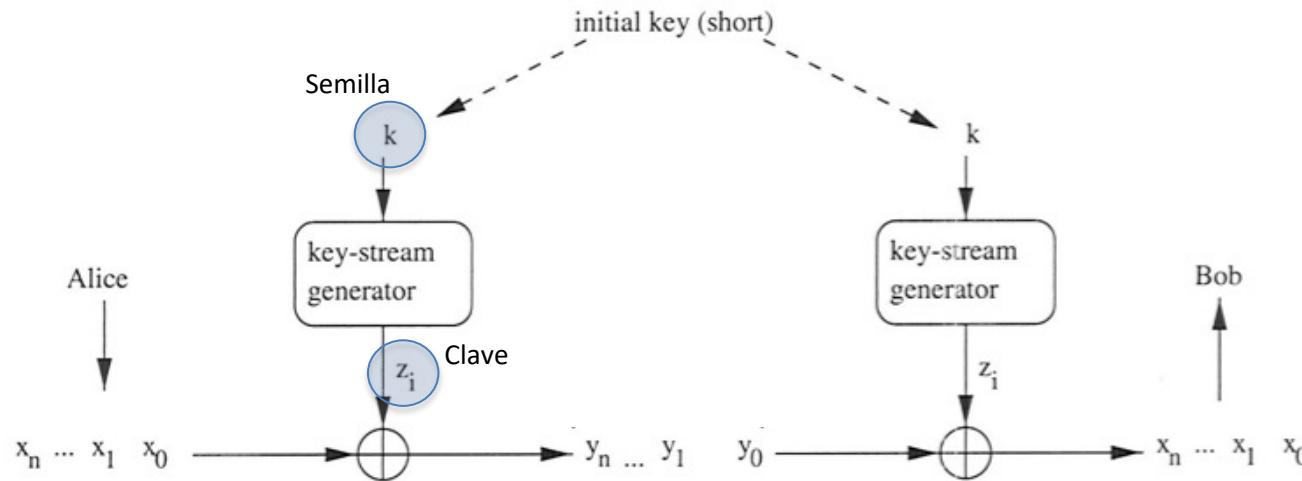
Decryption

Otra forma de verlo (2)

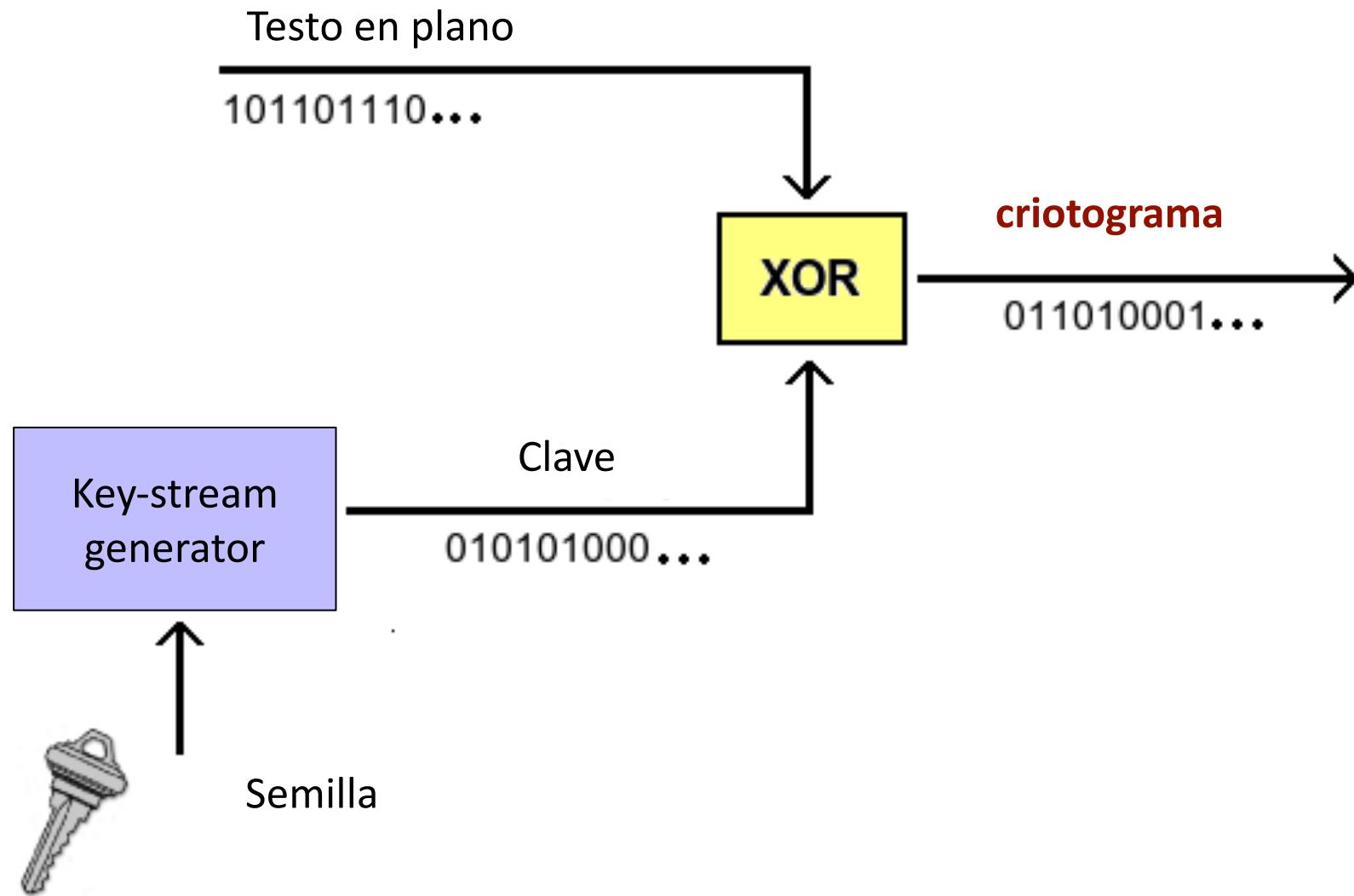


Modos de cifrado - cifrado en flujo

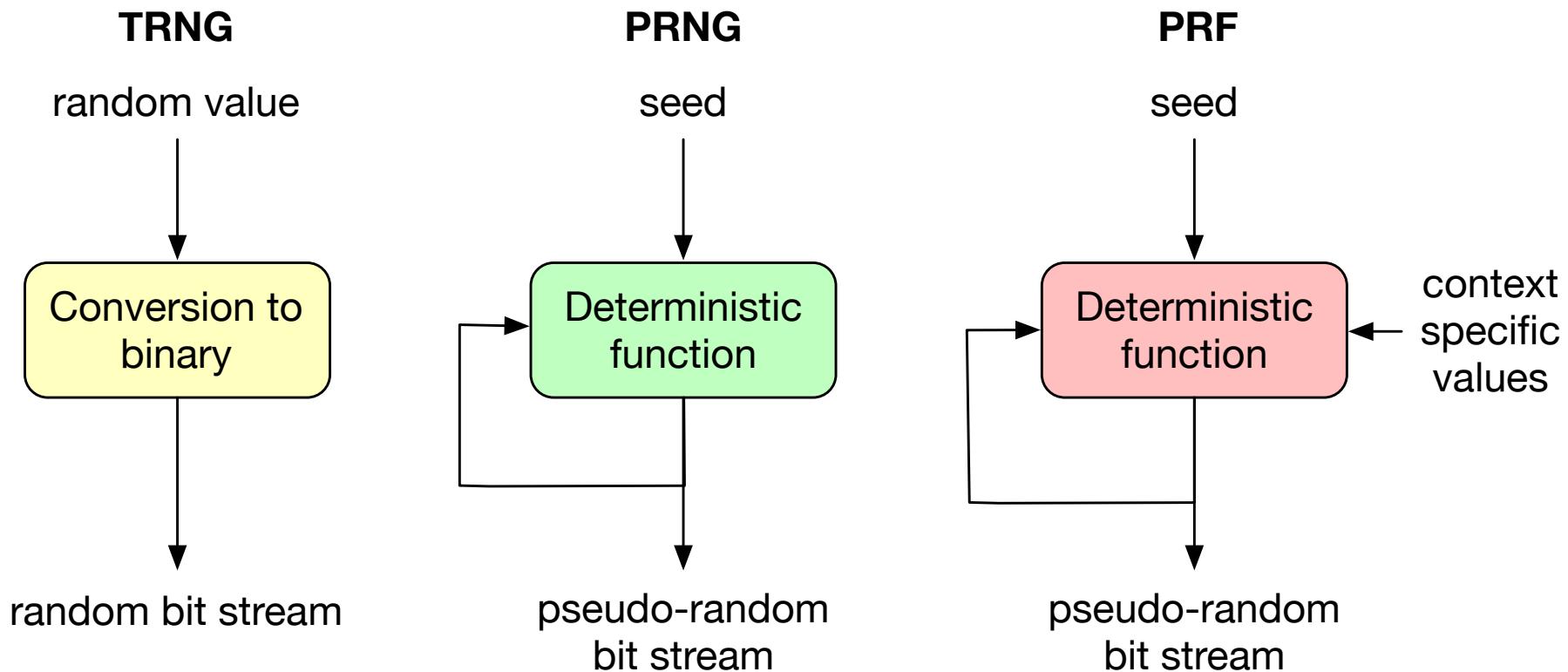
- Otros algoritmos, en lugar de procesar M particionándolo en bloques, necesitan procesarlo bit a bit, denominándose entonces **cifrados en flujo**
- Para ello, se opera en **XOR**:
 - cada bit del mensaje se realiza un XOR con el bit correspondiente del flujo de clave
 - El flujo de clave depende de la clave inicial **K (la semilla)**



Otra forma de verlo (1)



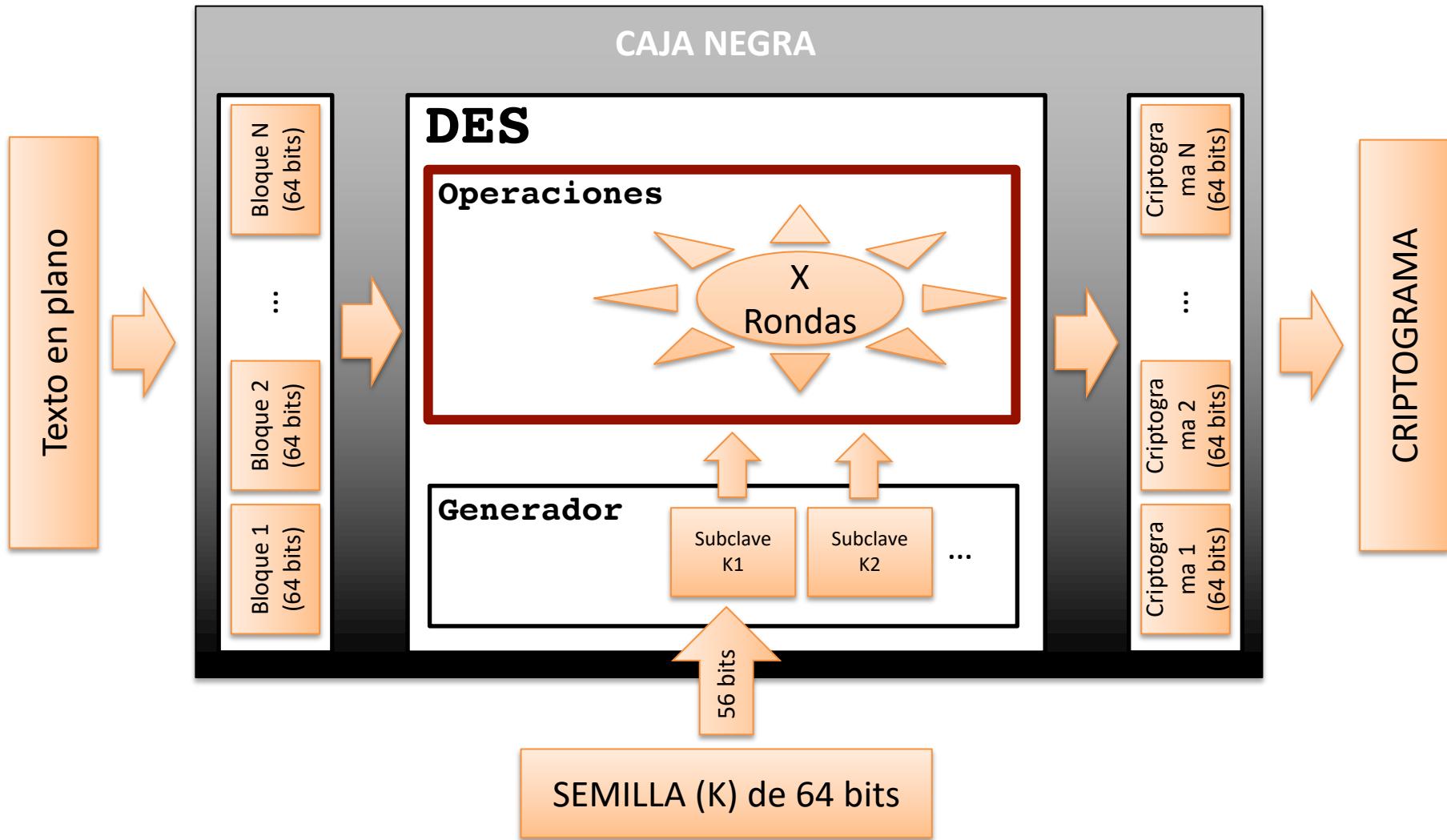
Tipos de generadores pseudo-randoms



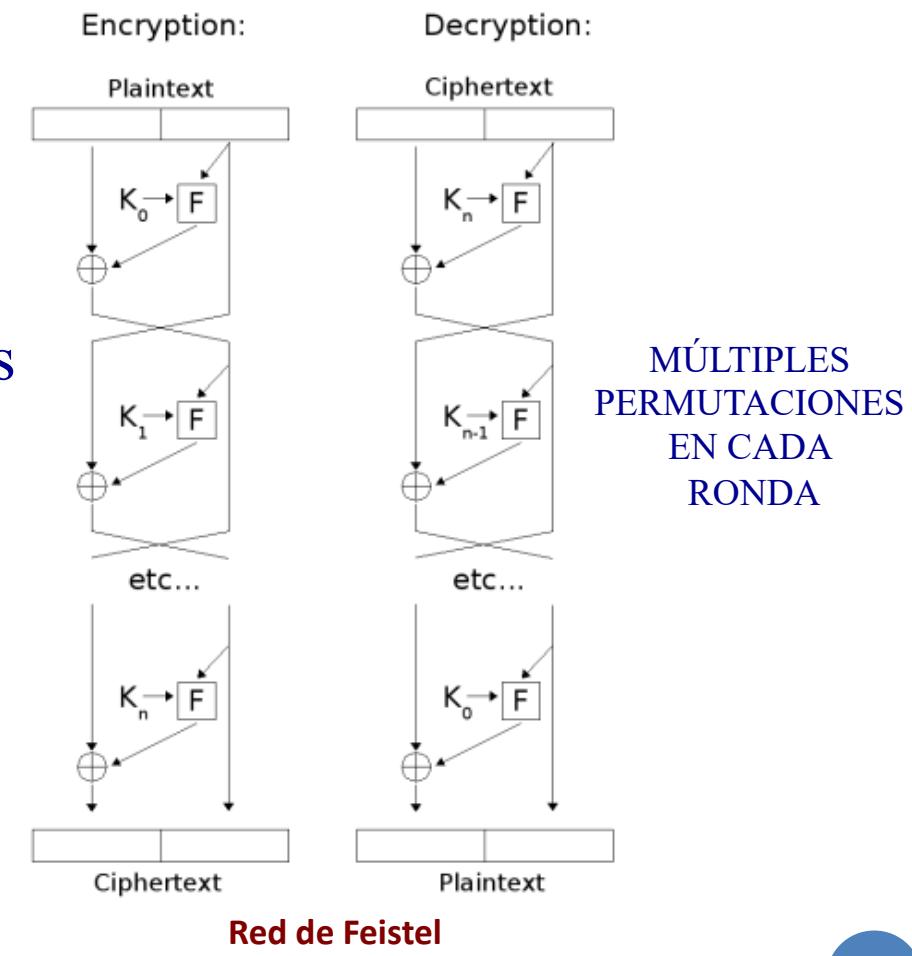
- Un **true random number generator (TRNG)**:
 - Toma como entrada un valor aleatorio
 - Su valor se puede extraer del entorno físico del ordenador (patrones de sincronización de las teclas, actividad eléctrica del disco, movimientos del ratón, valores del reloj del sistema, ...)
- Un **pseudorandom number generator (PRNG)**:
 - Toma como entrada un valor fijo (semilla) y produce una secuencia abierta utilizando un algoritmo determinista
 - La semilla suele ser generada por el TRNG
- Una **pseudorandom function (PRF)**:
 - se utiliza para producir una cadena pseudoaleatoria de bits de cierta longitud fija (por ejemplo, claves simétricas y nonces) + información de contexto

Algoritmo DES (Data Encryption Standard)

- *DES* es un algoritmo de cifrado simétrico que:
 - Usa bloques de **texto en claro de 64 bits**, y produce **bloques cifrados de igual tamaño**
 - Usa una **clave** que también es **de 64 bits** (8 octetos) de longitud
 - El último bit de cada octeto de la clave se usa como bit de paridad, por lo que la longitud efectiva de la clave (a efectos de seguridad) es de, en realidad, **56 bits**
- Diseñado por *IBM* para la competición del *National Bureau of Standards* (ahora *NIST*), en la que se solicitaban propuestas de algoritmos que pudiesen usarse como estándares para:
 - *cifrado de datos en transmisión*
 - *cifrado de datos en almacenamiento*por parte de el Gobierno americano, las empresas privadas y, en general, de cualquier tipo de usuario

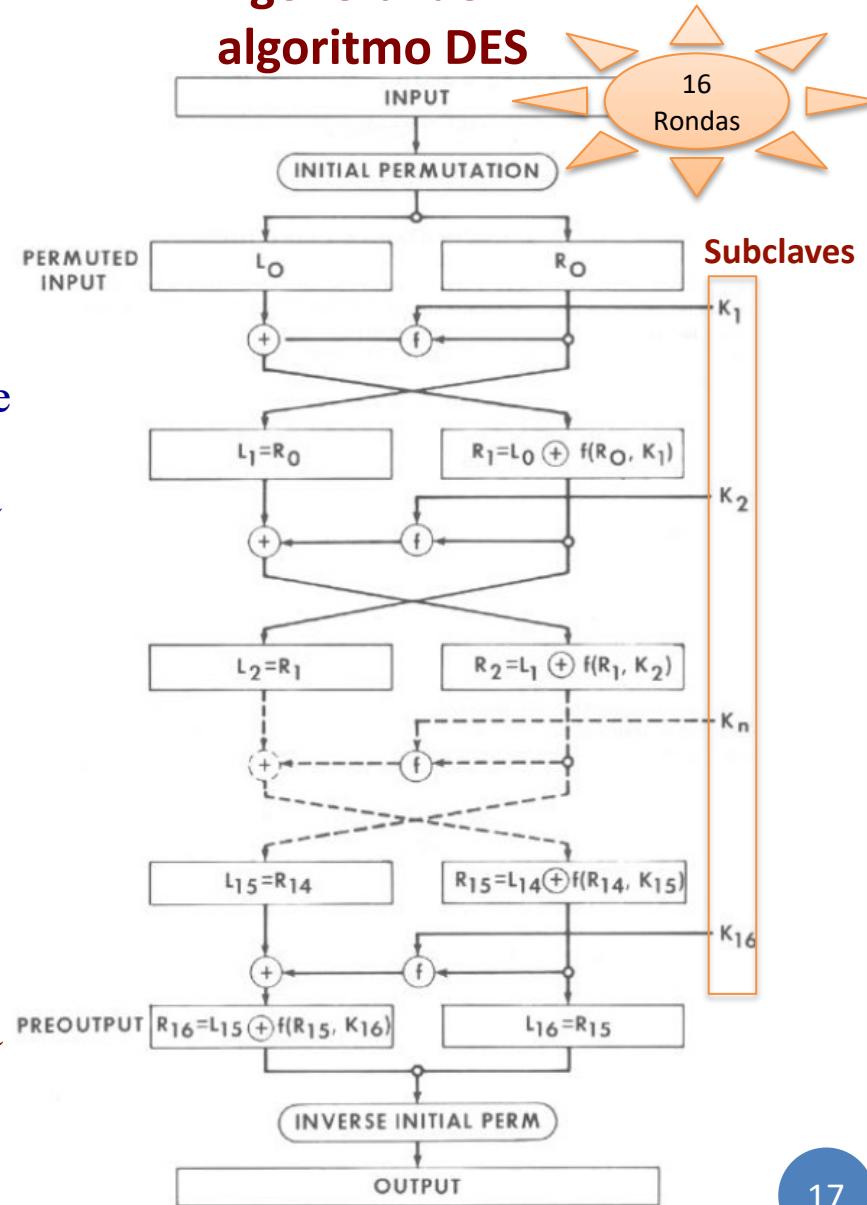


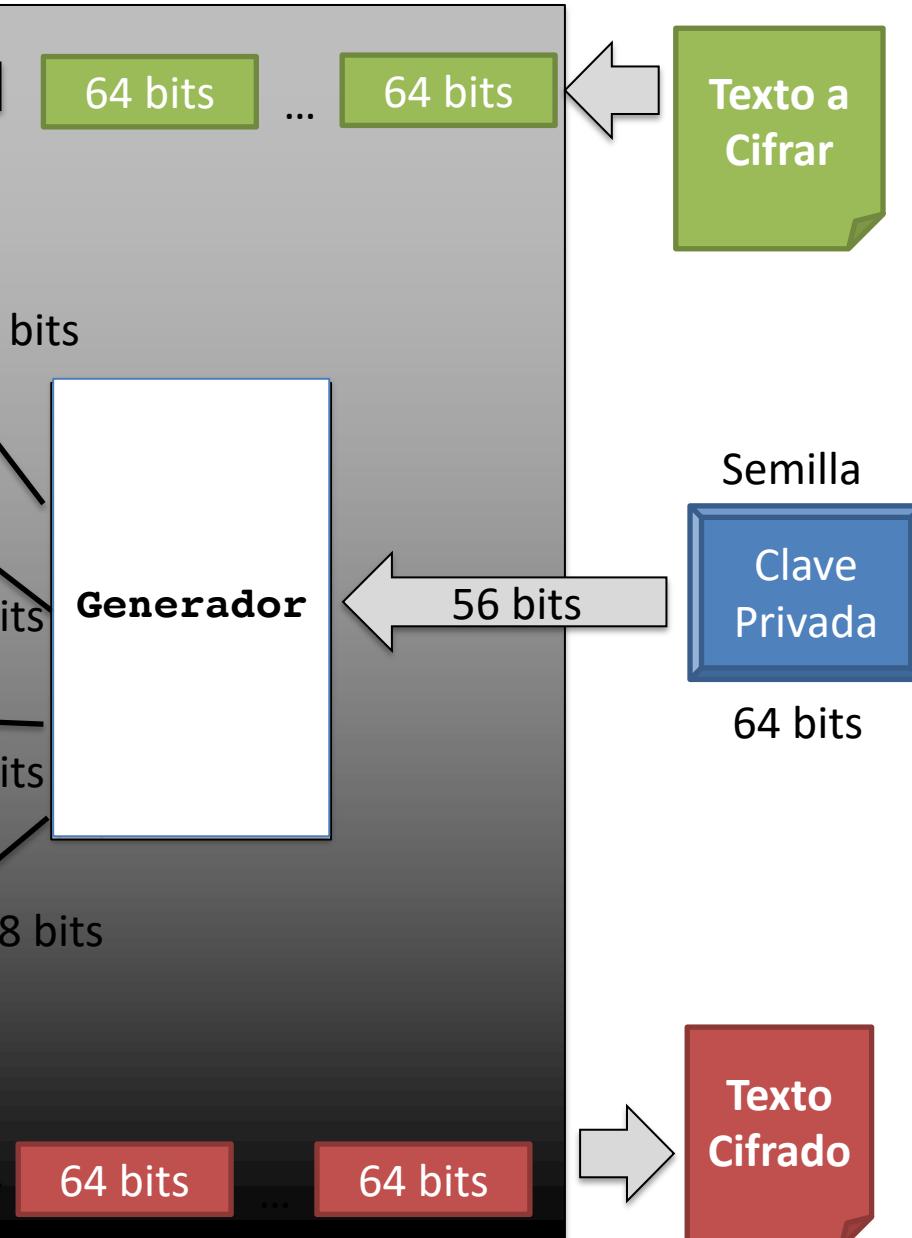
- Para el desarrollo del *DES*, *IBM* partió de *Lucifer*, un algoritmo propio desarrollado con anterioridad y usado principalmente en entornos bancarios
- Lucifer se basaba en el uso de una técnica denominada **red de Feistel**, y usaba una longitud de clave de 128 bits



- La idea de la red de Feistel queda reflejada en el propio ***DES***, como muestra la figura
- El esquema corresponde a la operación de cifrado (**16 etapas/rondas**) que se realiza para cada uno de los bloques del texto en claro
 - donde cada ronda aplica una subclave de 48 bits, generada por un generador interno y propio de DES y mediante una semilla de 56 bits
- ¿Qué extraemos de este esquema?
 1. Se aplica **16 rondas de operaciones (permutaciones y XOR ⊕)** para el cifrado
 2. Se usa **16 subclaves en cada etapa** – luego, ¡ las subclaves son el secreto !
 3. La robustez de DES se encuentra en la cantidad de permutaciones que se realizan dentro del algoritmo

Esquema general del algoritmo DES

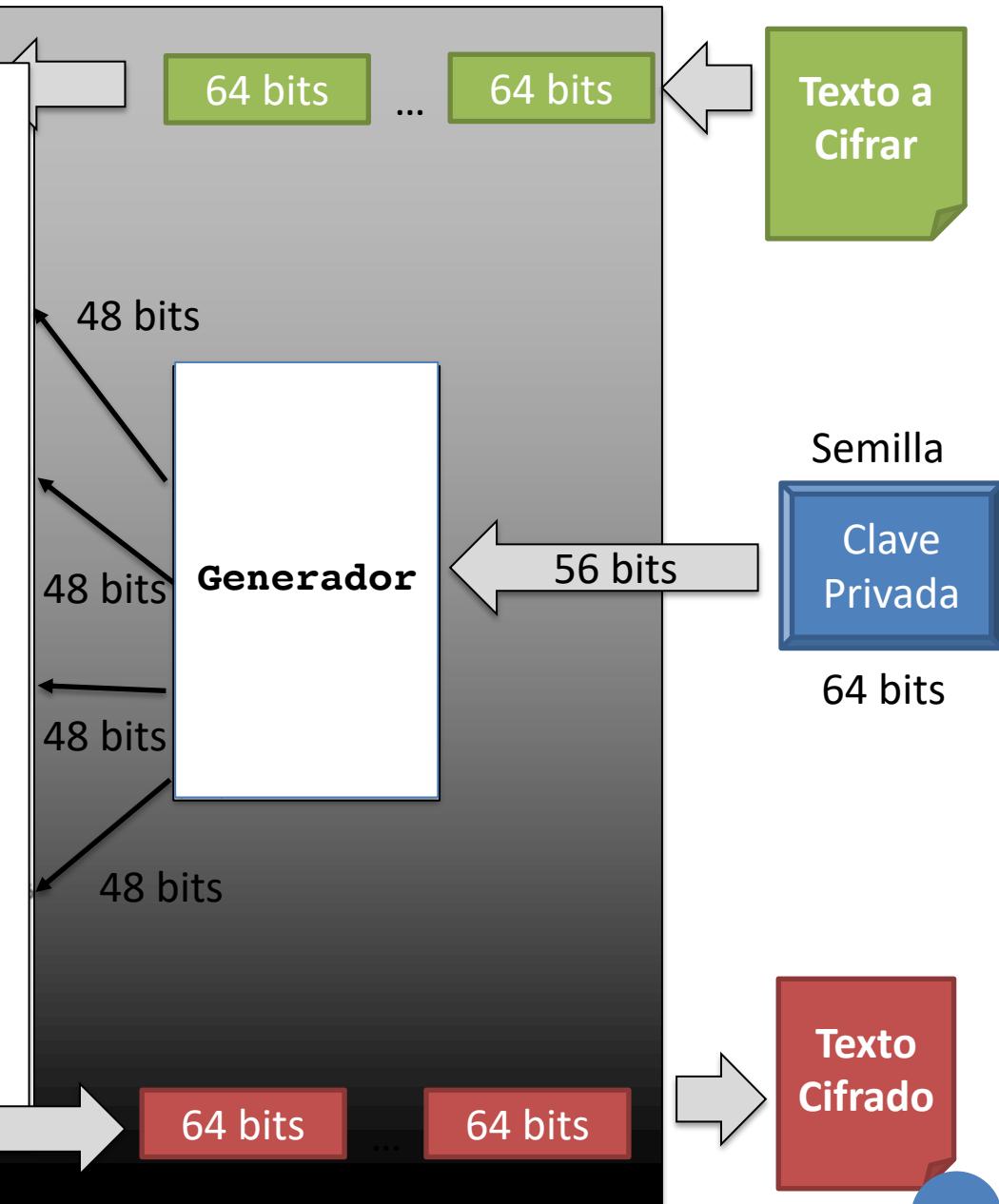




F

Se realizan una serie de operaciones internas:

- 1) Expandir los datos de entrada (siguiendo un proceso estandarizado) a 48 bits para que se pueda operar con las subclaves
- 2) CIFRAR con XOR (48 bits + 48 bits)
- 3) Comprimir el criptograma a 32 bits siguiendo un proceso estandarizado



INCISO - Efecto avalancha

- Una característica eficiente de *DES*, y deseable en cualquier algoritmo de cifrado, es el **efecto avalancha**
 - Es decir, un cambio pequeño en el texto en claro, o en la clave, produce un cambio significativo en el texto cifrado
 - Si, por el contrario, el cambio fuera pequeño, el cripto-analista tendría mucha ventaja, porque se reduciría el número de posibles textos en claro o de posible claves
 - En otras palabras: impide reducir el espacio de búsqueda de claves para un ataque de fuerza bruta
- Ejemplo de efecto avalancha en *DES*:

Plaintext: 0000000000000000

Key: 22234512987ABB23

Ciphertext: 4789FD476E82A5F1

Plaintext: 0000000000000001

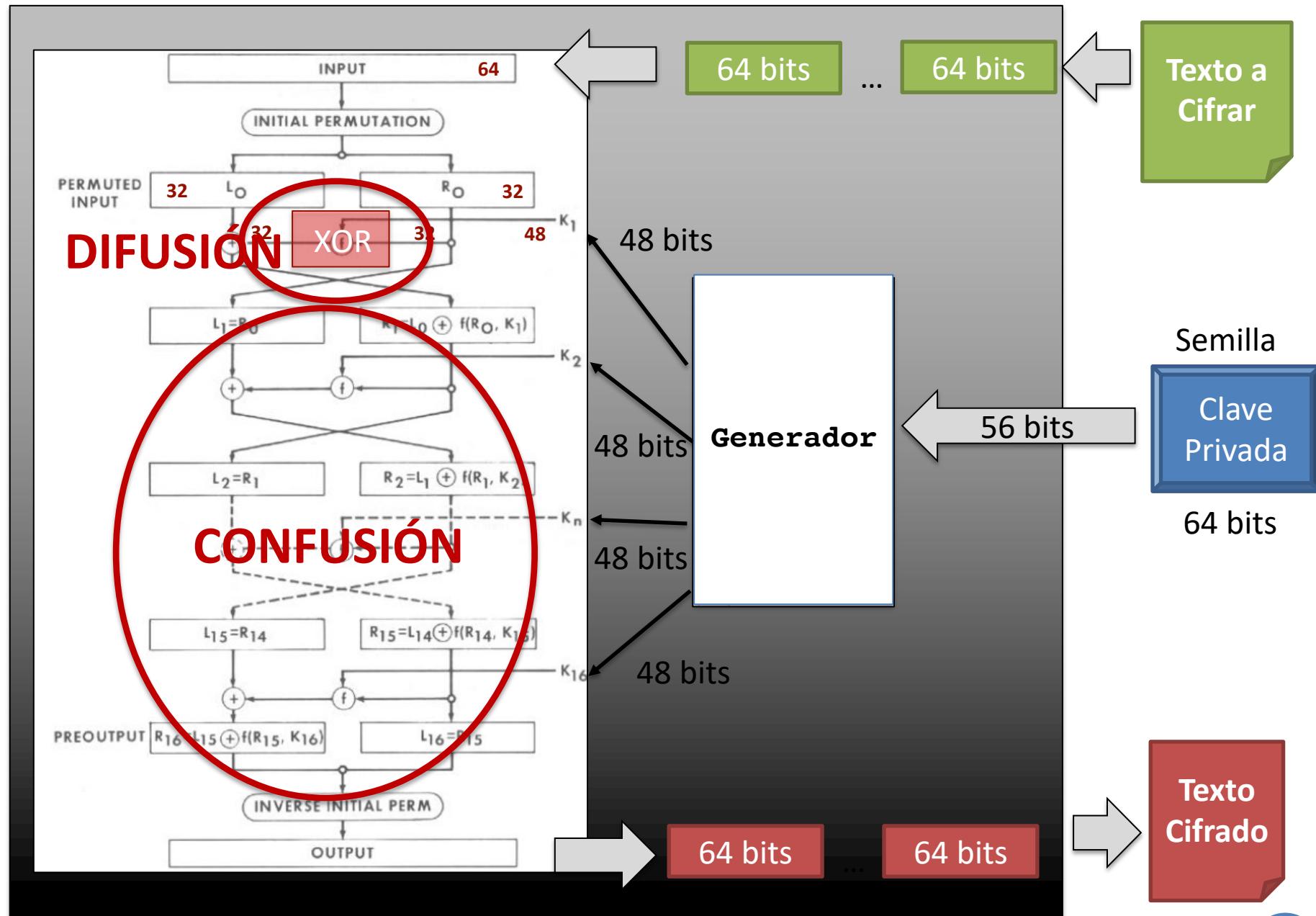
Key: 22234512987ABB23

Ciphertext: 0A4ED5C15A63FEA3

INCISO - Efecto avalancha

- El concepto de efecto avalancha se deriva de las tesis de Claude Shannon, el padre de la **Teoría de la Información** en su artículo:
 - *Communication Theory of Secrecy Systems*, 1949
- En ese artículo definía, entre otros conceptos, las propiedades de **difusión** y **confusión** para evitar (o dificultar) los ataques basados en análisis estadísticos
 - **difusión**: cada carácter del texto cifrado ha de depender de diferentes partes de la clave → ej: $M \text{ XOR } K$
 - **confusión**: la relación entre el texto cifrado y la clave ha de ser tan complicada como sea posible → ej: *PERMUTACIONES*
- Nota: es un criterio que se aplica a cualquier algoritmo de cifrado (DES, 3DES, IDEA, Camellia, etc.)





Seguimos con el algoritmo DES ...

- Hasta el momento no se conoce ningún ataque al algoritmo *DES* en sí mismo, y que haya sido completamente efectivo
- Por otro lado, un **ataque exhaustivo a la clave (brute-force attack)** podría parecer impracticable si suponemos, por ejemplo, una operación de descifrado por microsegundo
 - con longitud de clave de 56 bits, existen 2^{56} posibles claves ($= 7.2 \times 10^{16}$)



Algoritmo DES

- Sin embargo, se puede suponer que el criptoanalista va a disponer de capacidad de descifrado con microprocesadores en paralelo, y, por lo tanto, la situación cambia drásticamente:

Key size (bits)	Number of alternative keys	Time required at 1 decryption/ μs	Time required at 10^6 decryptions/ μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

- Por lo anterior, se deduce que la longitud de clave del *DES* resulta demasiado corta si el criptoanalista dispone del hardware adecuado

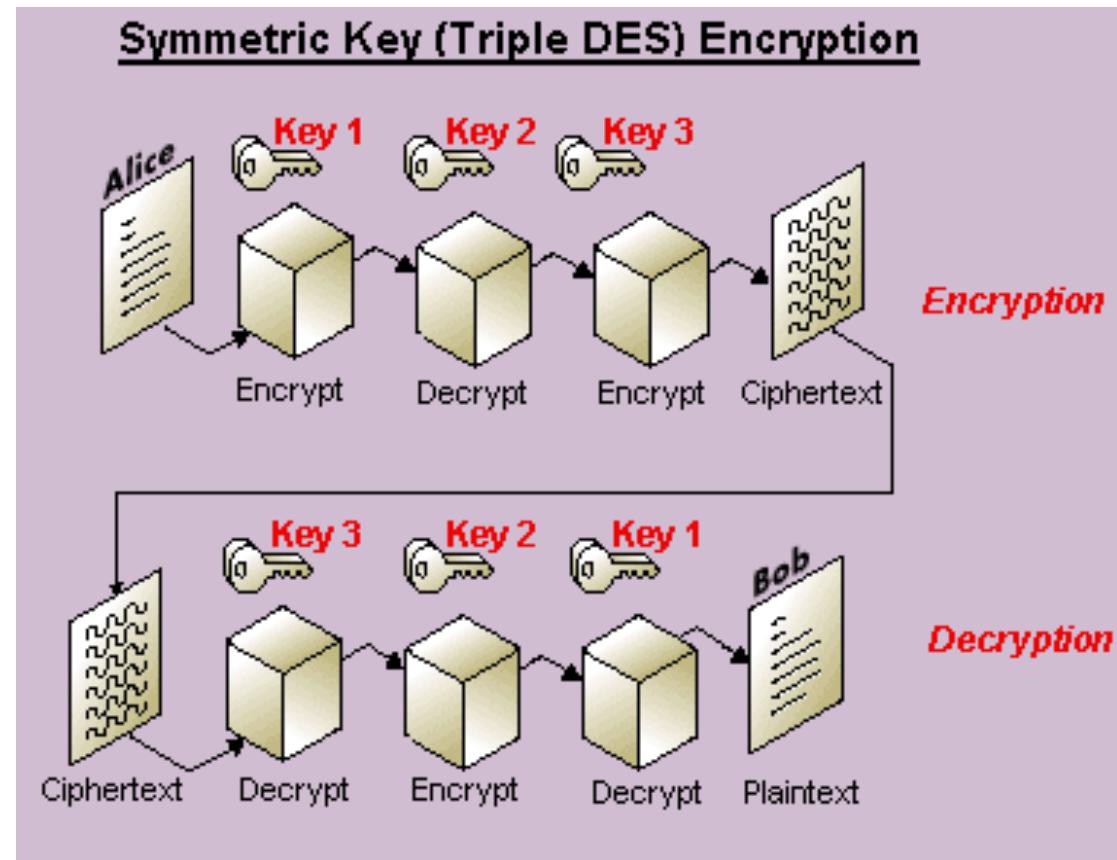


DES Key Search Machine

6 estaciones SUN-2, 27 placas de circuitos, 1800 chips customizados,
24 unidades de búsqueda por cada chip

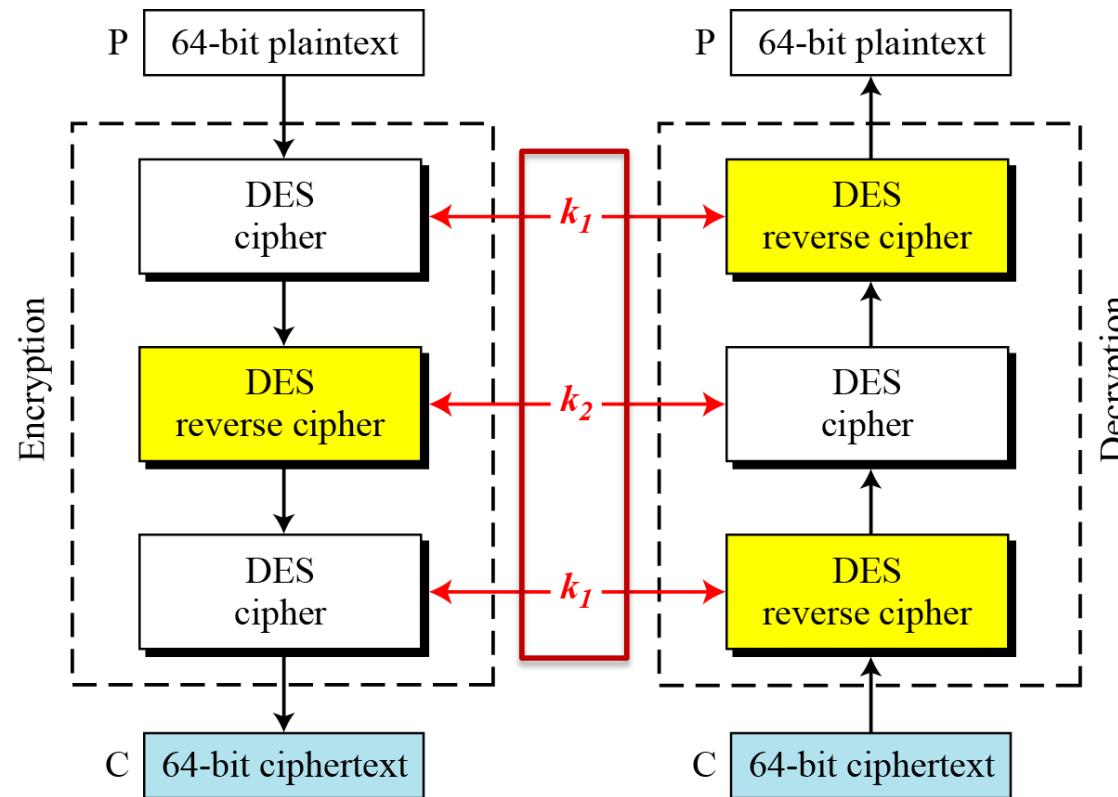
Algoritmo Triple-DES

- Para aprovechar las ventajas del *DES*, y a la vez contrarrestar su escasa longitud de clave, los organismos de estandarización han adoptado el criptosistema ***Triple-DES*** (ó ***3DES***)
- Consiste en usar una secuencia de tres operaciones *DES*, con **3 claves distintas** (168 bits)



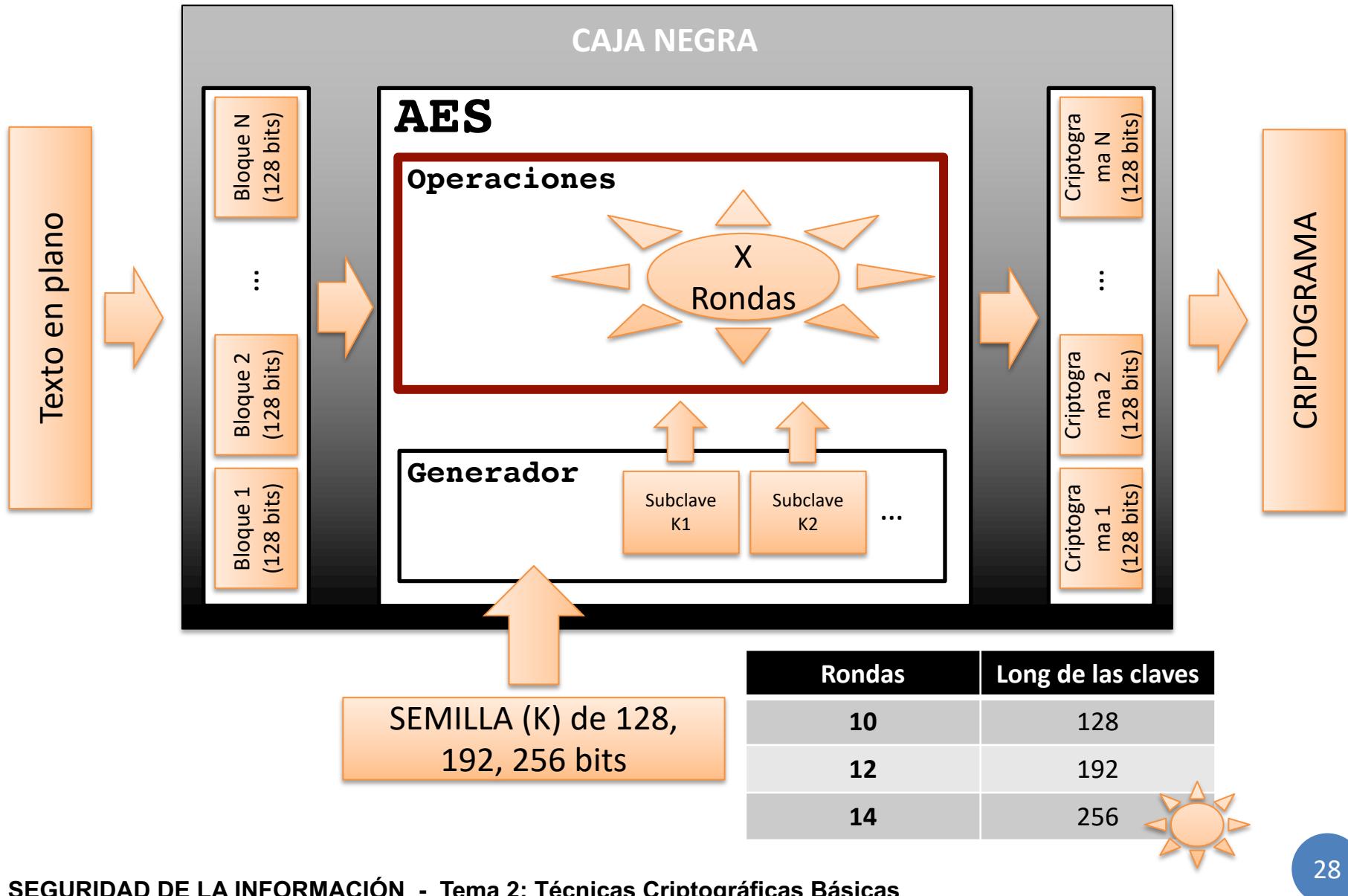
Algoritmo Triple-DES (variante de 2 keys)

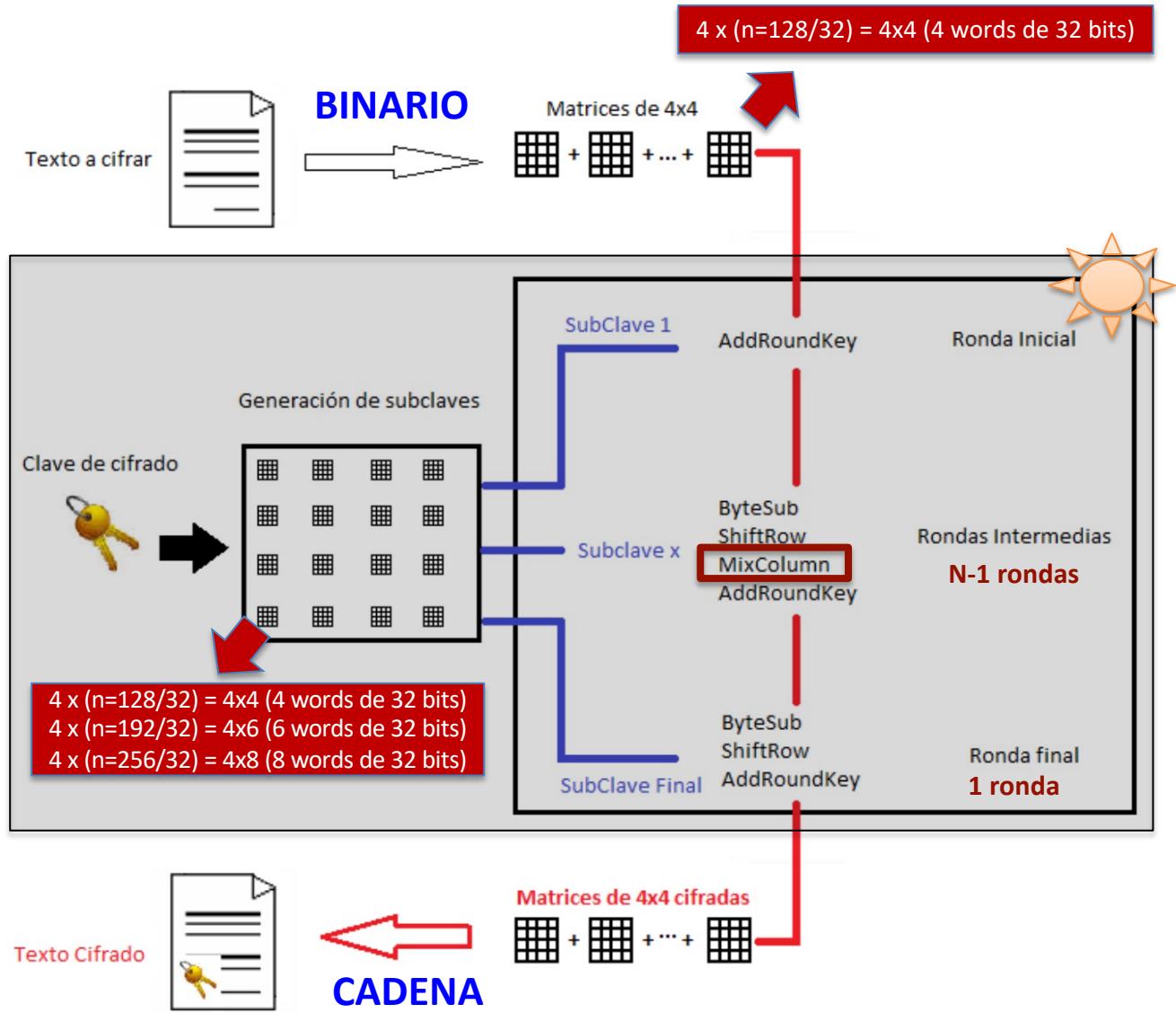
- Existe una variante en la que se utilizan sólo 2 claves, pero este esquema ha sido objeto de ataques, de forma que la **K1 = K3 (112 bits)**



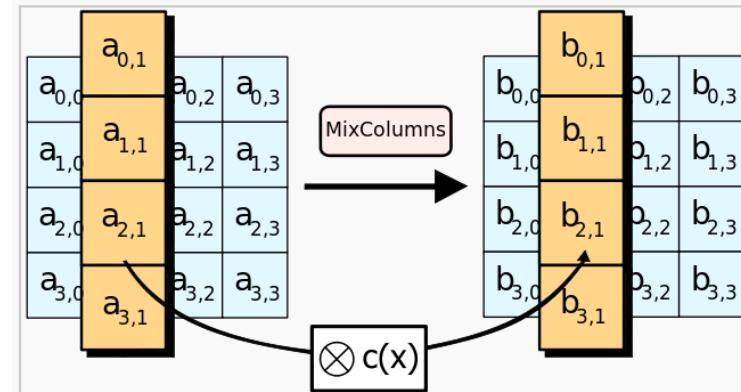
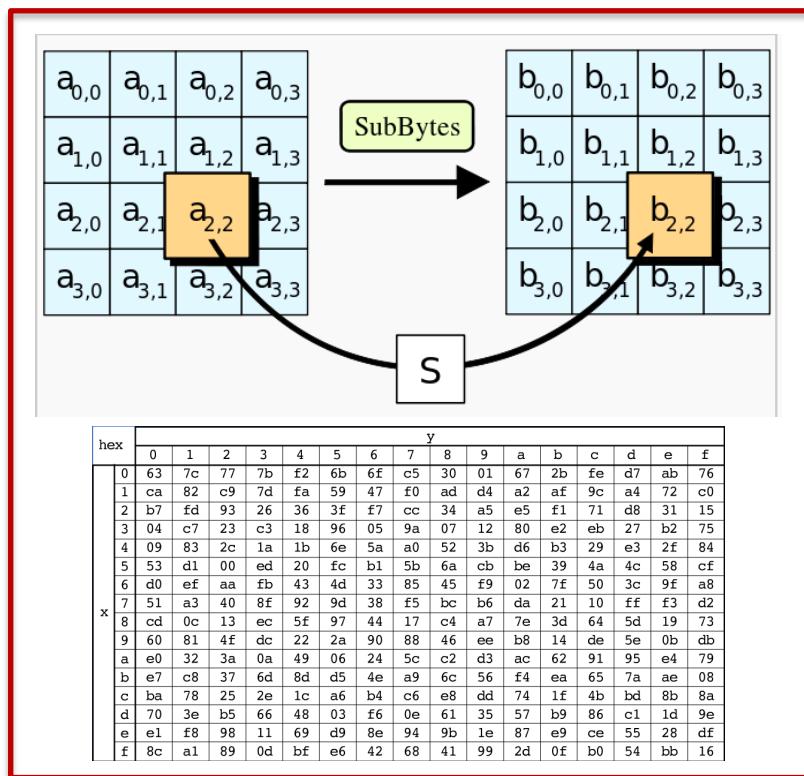
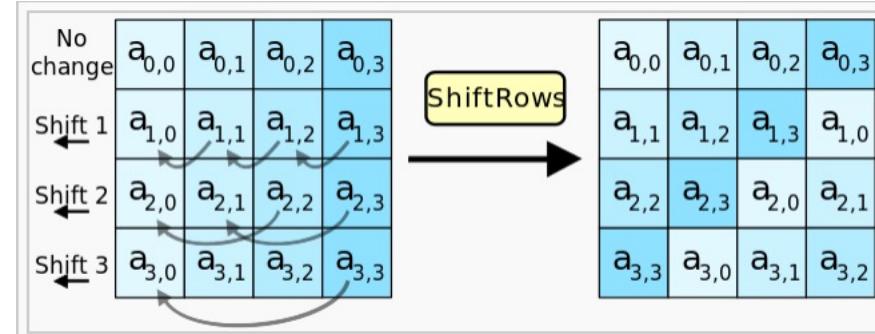
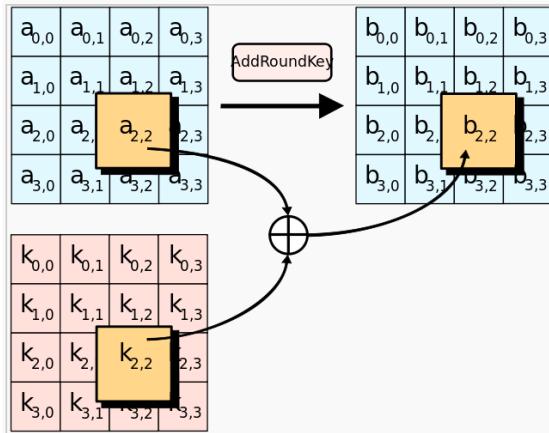
Algoritmo AES (Advanced Encryption Standard)

- AES fue publicado por NIST en 2001 como estándar de **cifrado simétrico en bloque** para sustituir a DES, especialmente en aplicaciones comerciales
 - su nombre original es *Rijndael*, por sus autores *Rijmen* y *Daemen*
 - utiliza **una clave de 128, 192 o 256 bits**
 - la longitud n de cada bloque de datos en los que se subdivide M es **128 bits**
- AES no utiliza una red de Feistel, sino una
 - **red de sustitución-permutación-operaciones aritméticas**
 - Cada etapa de AES se compone de cuatro funciones distintas:
 - **sustitución de byte**
 - **permutación**
 - **operaciones aritméticas en campo finito**
 - **XOR**

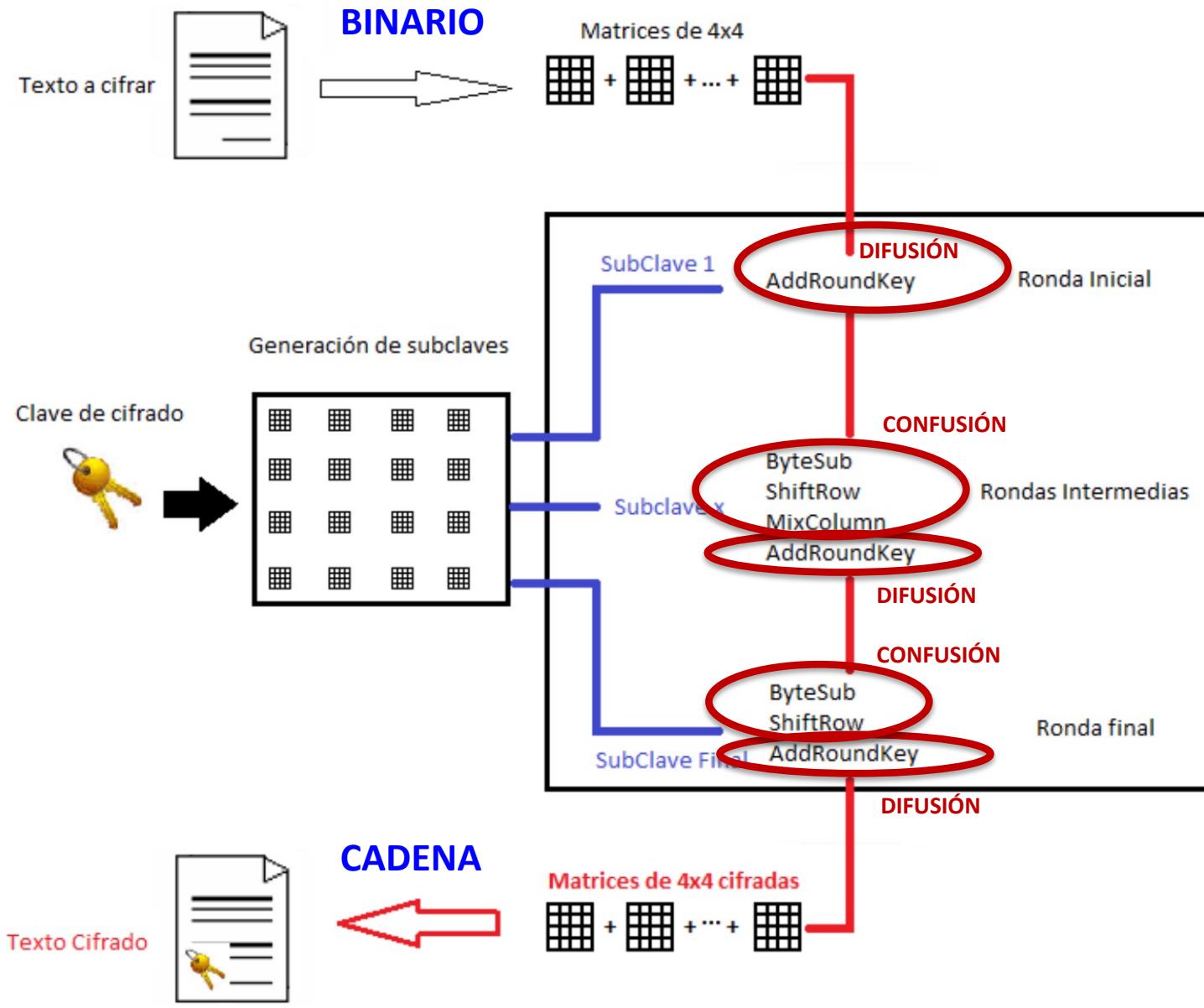




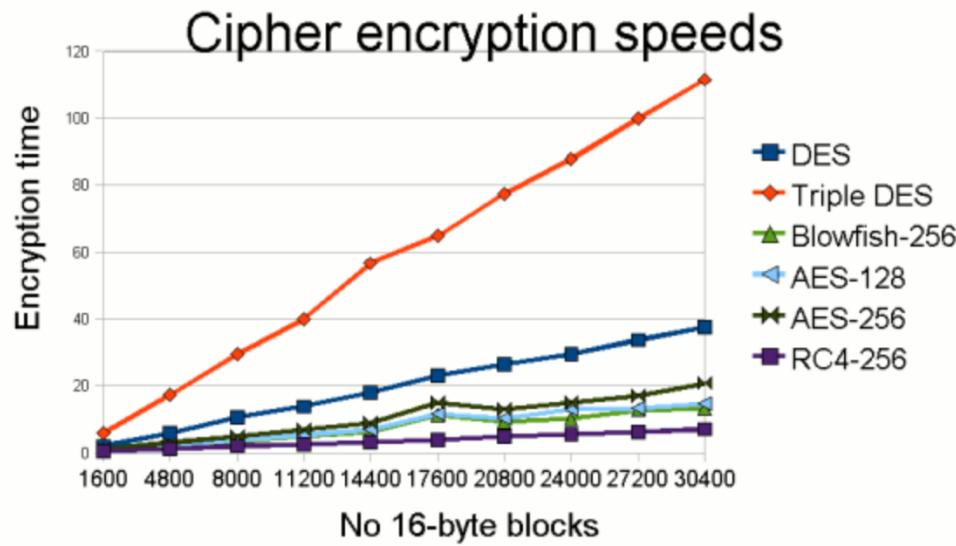
- La figura muestra el proceso de cifrado en AES para **128 bits** de clave (**10 etapas**)
- Otras posibilidades:
 - **192 bits** (**12 etapas**)
 - **256 bits** (**14 etapas**)



MixColumn multiplica la matriz que se está computando con matrices preestablecidas



- Rendimiento de AES comparado con otros algoritmos simétricos



Date	Minimum of Strength	Symmetric Algorithms
2010 (Legacy)	80	2TDEA*
2011 - 2030	112	3TDEA
> 2030	128	AES-128
>> 2030	192	AES-192
>>> 2030	256	AES-256

- Existen algunas recomendaciones generales:
 - En general, la **longitud mínima de clave** para un cifrado simétrico en bloque debería ser **128 bits**
 - **El tamaño mínimo de los bloques de datos** dependerá de la aplicación específica en la que se use el algoritmo, pero en la mayoría de ocasiones el mínimo **debería ser 128 bits**
 - La **cantidad máxima de información a cifrar con una misma clave** debería limitarse a $2^{n/2}$, donde n es el tamaño del bloque de datos
 - En cuanto a los algoritmos: ...

CLAVES:
 ≥ 128 bits

Bloques:
 ≥ 128 bits

Rekeying:
cada $2^{n/2}$

Otros algoritmos simétricos relevantes

– Blowfish

- Requiere una **clave de entre 32 y 448 bits** (pero sólo se recomienda su uso con **más de 80 bits**)
- Se utiliza en algunas configuraciones de IPSEC
- La longitud de cada **bloque de datos es de 64 bits**, demasiado pequeño para algunas aplicaciones
 - Por ese motivo, sólo se recomienda en uso heredado (*legacy use*)

– Kasumi

- Requiere una **clave de 128 bits** de longitud, y la longitud de los **bloque de datos es de 64 bits**
- Se usa en UMTS (con el nombre UIA1) y en GSM (con el nombre A5/3)
- Presenta una serie de problemas que no afectan a su uso práctico en esas aplicaciones
 - sin embargo, no se recomienda para aplicaciones futuras

– Camellia

- Requiere una **clave de 128 bits**, pero también da soporte a claves **de 192 y 256 bits**
 - Las versiones con 192 o 256 bits de clave son un 33% más lentas que la de 128 bits
- Se utiliza como **uno de los posibles cifrados en TLS**
- Por el momento, no se han encontrado ataques efectivos a este algoritmo

- Existen algunas recomendaciones generales:

- En general, la **longitud mínima de clave** para un cifrado simétrico en bloque debería ser **128 bits**
- **El tamaño mínimo de los bloques de datos** dependerá de la aplicación específica en la que se use el algoritmo, pero en la mayoría de ocasiones el mínimo **debería ser 128 bits**
- La **cantidad máxima de información a cifrar con una misma clave** debería limitarse a $2^{n/2}$, donde n es el tamaño del bloque de datos
- En cuanto a los algoritmos:

CLAVES:
>= 128 bits

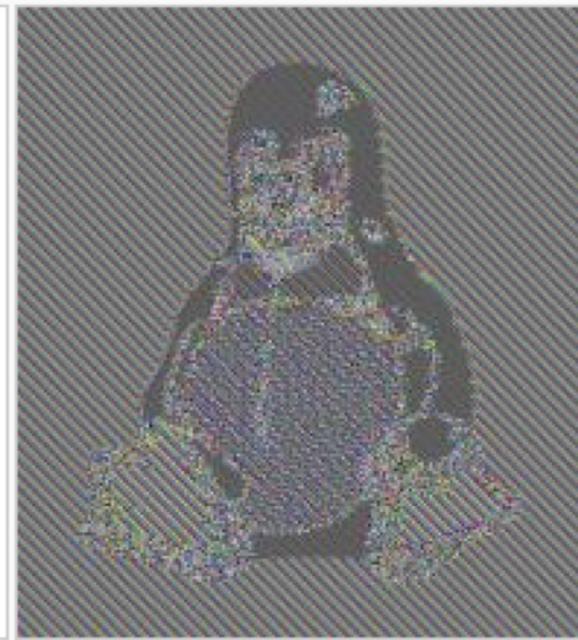
Bloques:
>= 128 bits

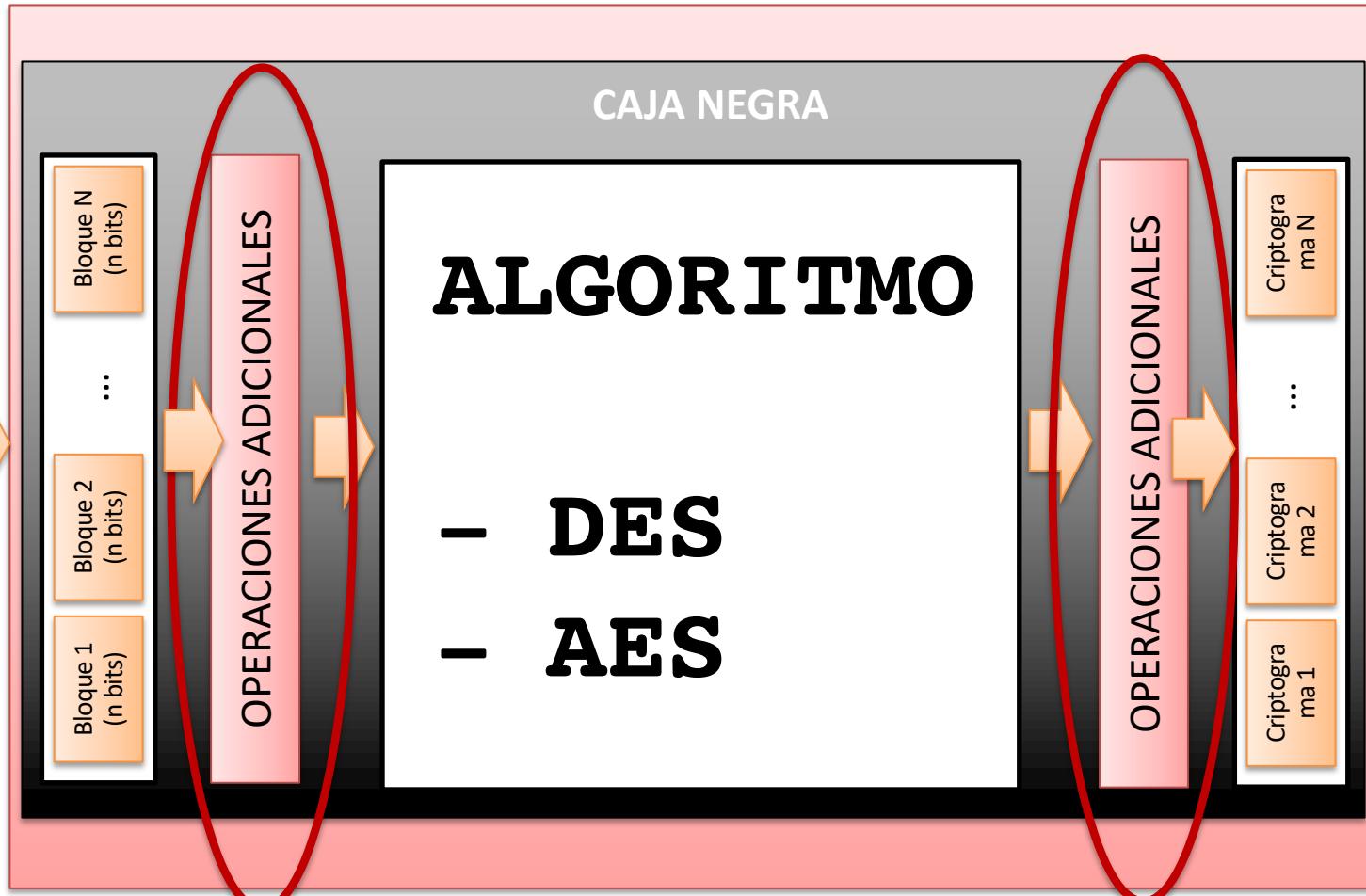
Rekeying:
cada $2^{n/2}$

Primitive	Recommendation	
	Legacy	Future
AES	✓	✓
Camellia	✓	✓
Three-Key-3DES	✓	✗
Two-Key-3DES	✓	✗
Kasumi	✓	✗
Blowfish \geq 80-bit keys	✓	✗
DES	✗	✗

Modos de operación para algoritmos simétricos

- Un modo de operación es una técnica para mejorar el efecto final de un algoritmo criptográfico
 - También se usa para adaptar el algoritmo a un tipo de aplicación concreta
- En ningún caso supone la modificación del algoritmo de cifrado en sí, sino de la forma en que se opera con los bloques de datos

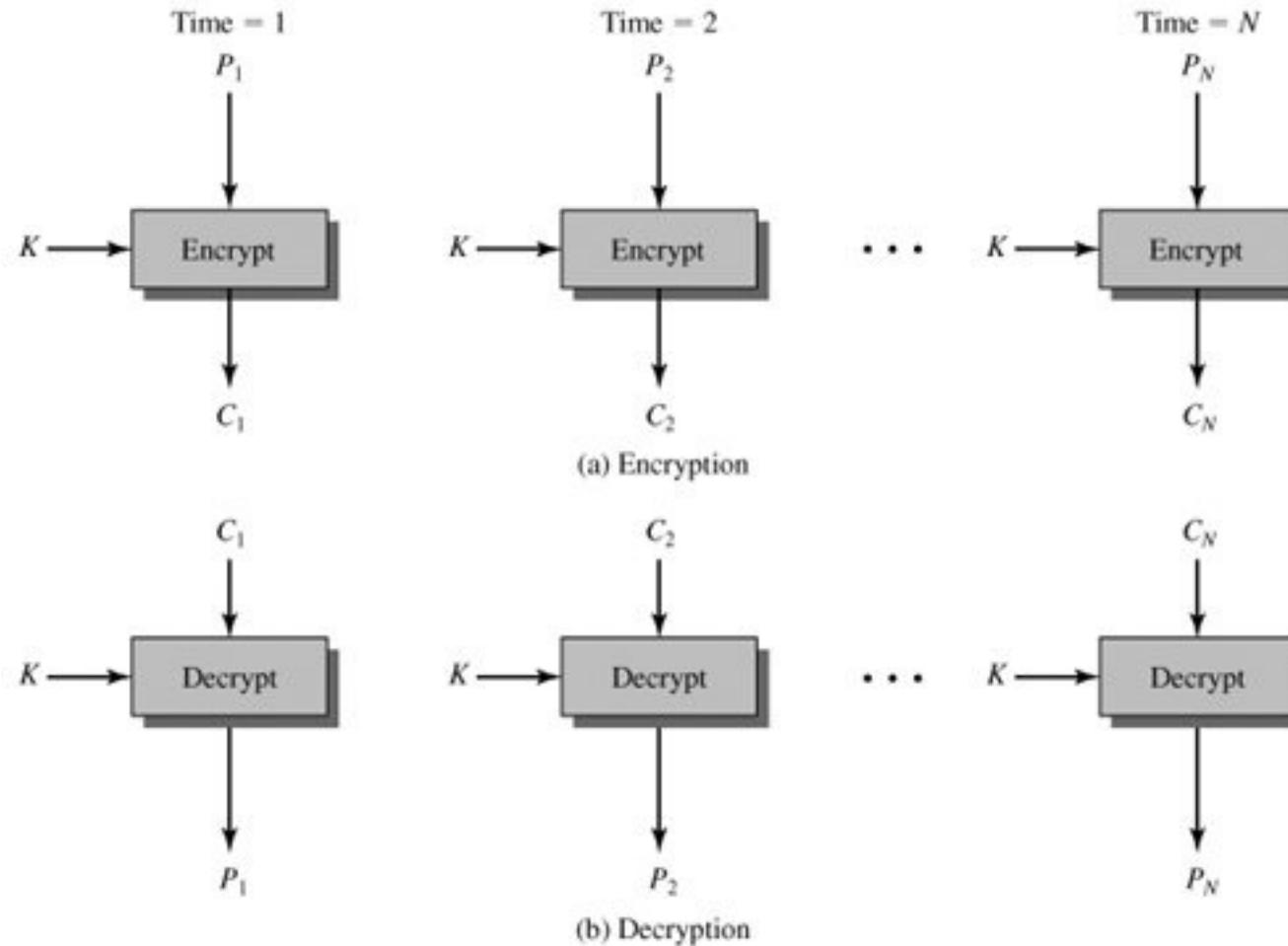


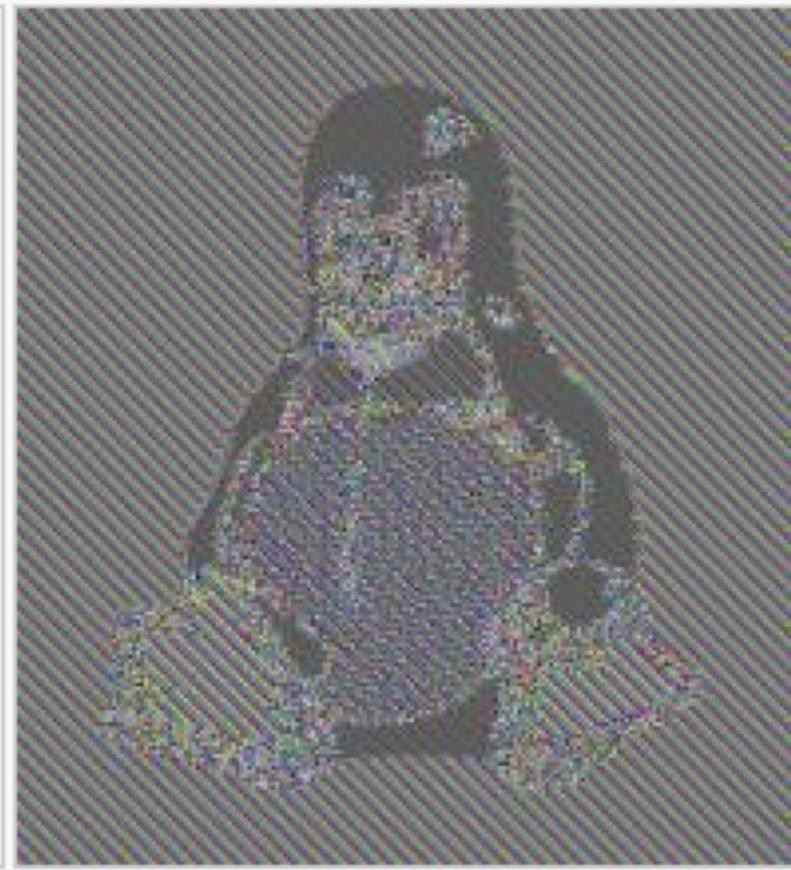


Modos de operación para algoritmos simétricos

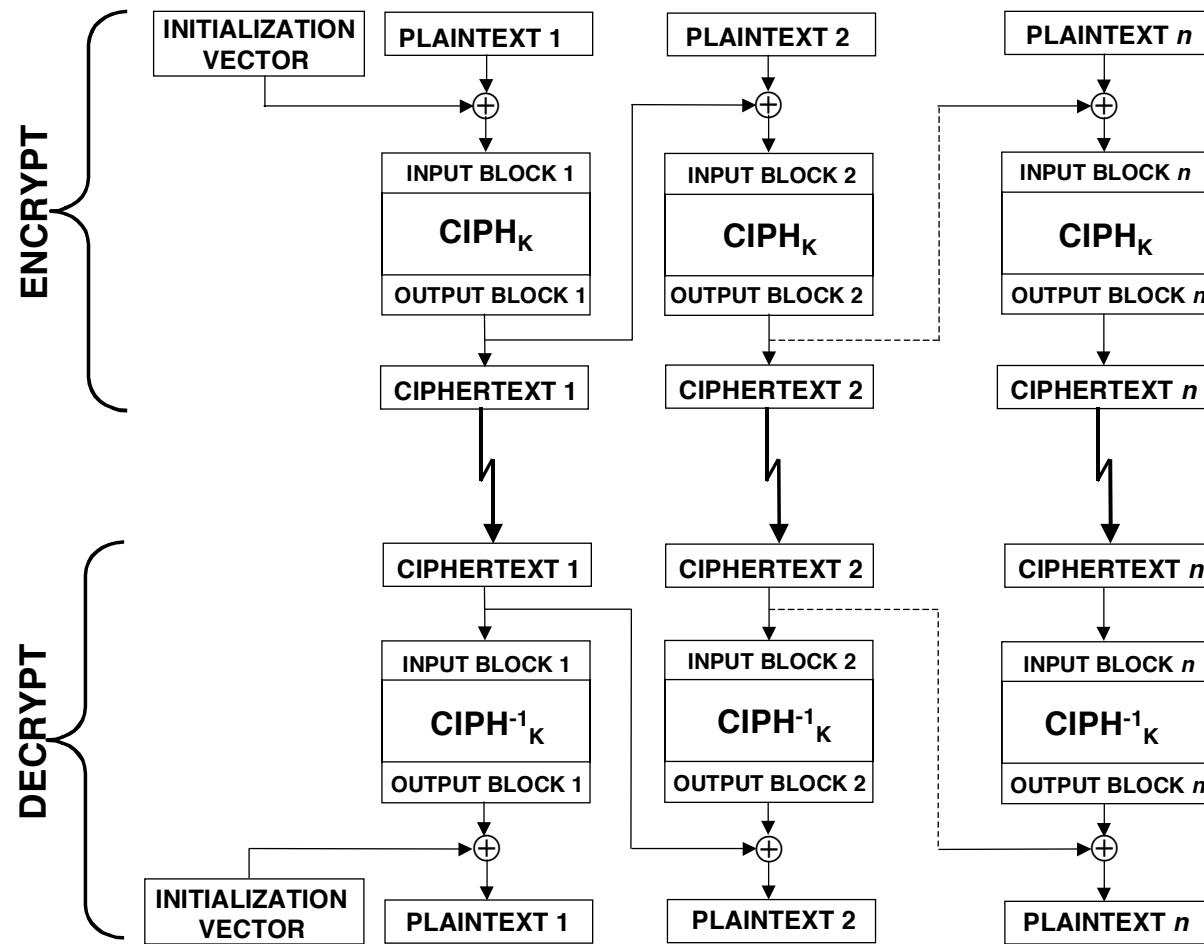
- Hay diferentes modos de operación, o lo que es lo mismo, diferentes formas de aplicar un mensaje M a un algoritmo de cifrado en bloque
 - cada una de ellas con sus ventajas y desventajas
- NIST ha definido cinco modos de operación, y cualquiera de ellos se puede utilizar con cualquier algoritmo simétrico (DES , $3DES$, AES , ...)
 - Electronic Codebook (ECB)
 - Cipher Block Chaining (CBC)
 - Cipher Feedback (CFB)
 - Output Feedback (OFB)
 - Counter (CTR)
 - **Galois-CTR (GCM)** – muy importante por su aplicación actual en la última versión de TLS

- Electronic Codebook (ECB)



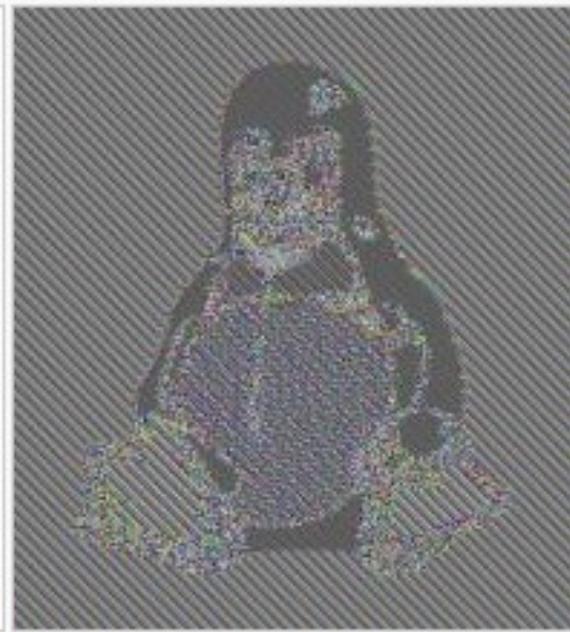


- Cipher Block Chaining (CBC)

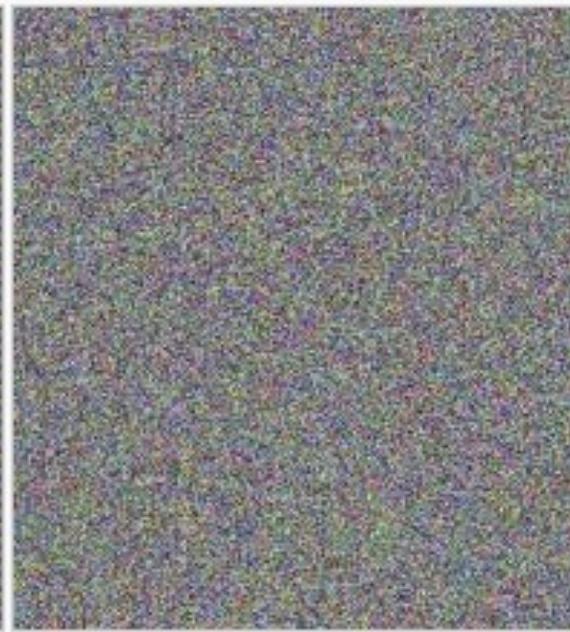




Original image

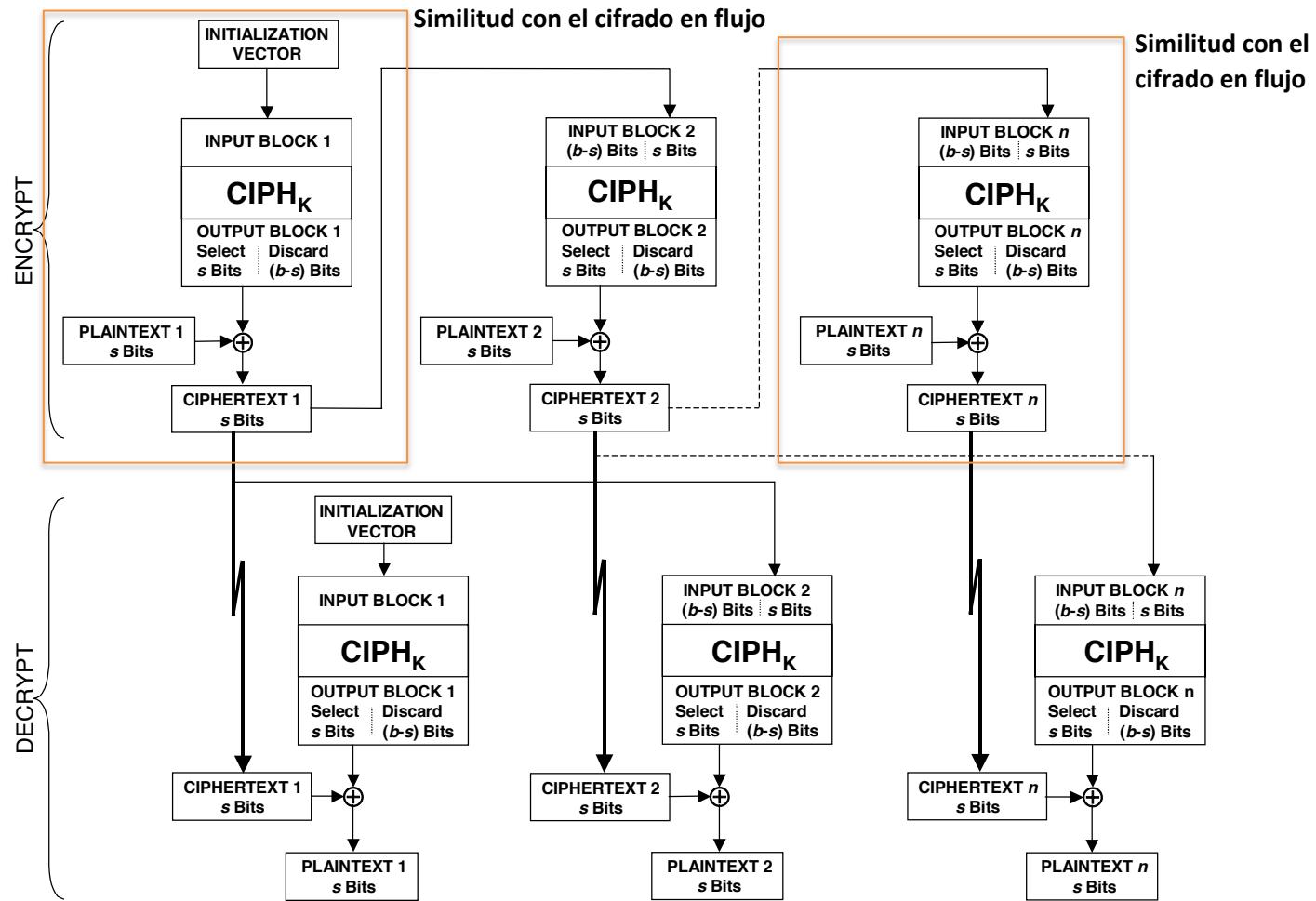


Encrypted using ECB mode

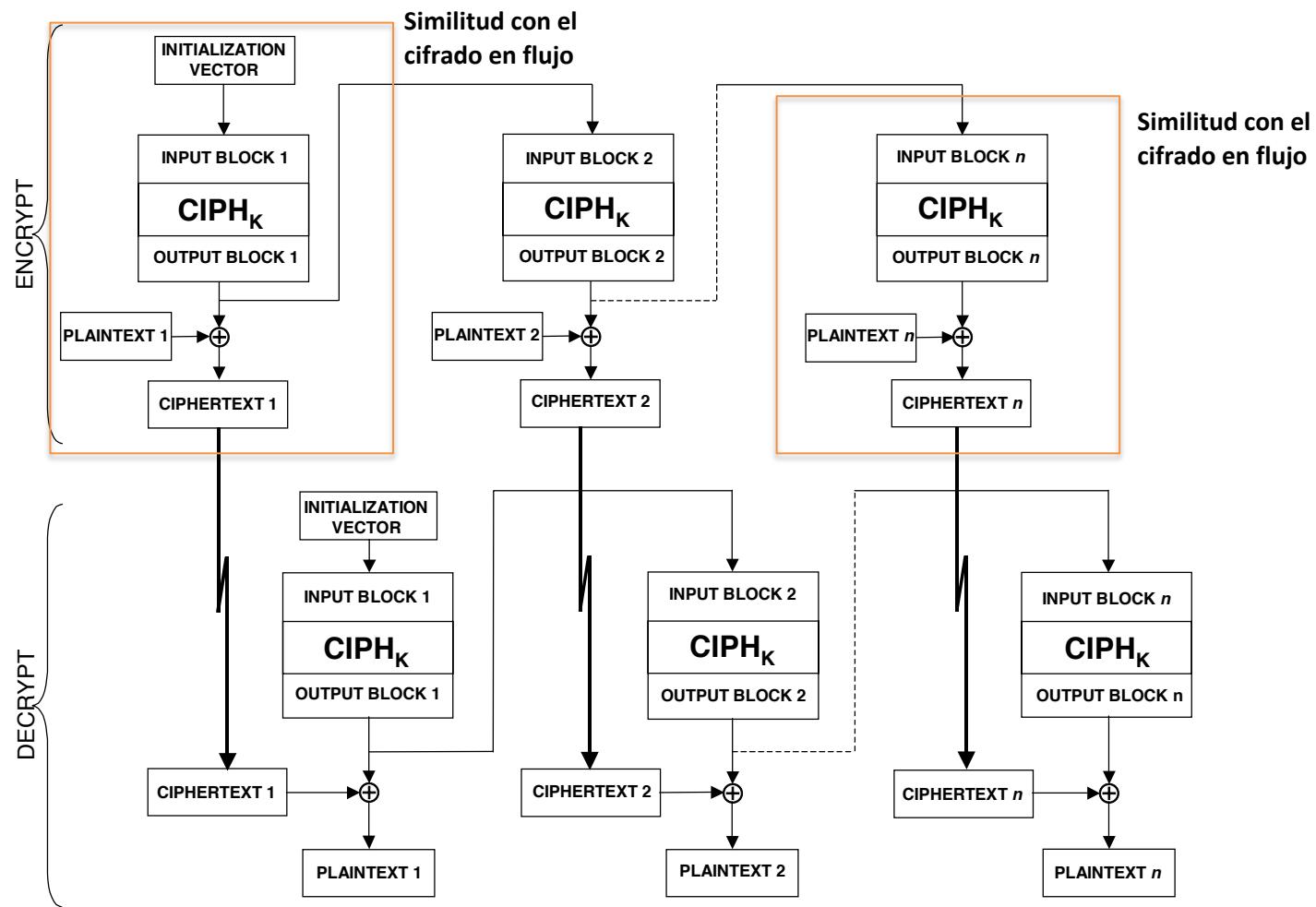


Modes other than ECB result in
pseudo-randomness

• Cipher Feedback (CFB)

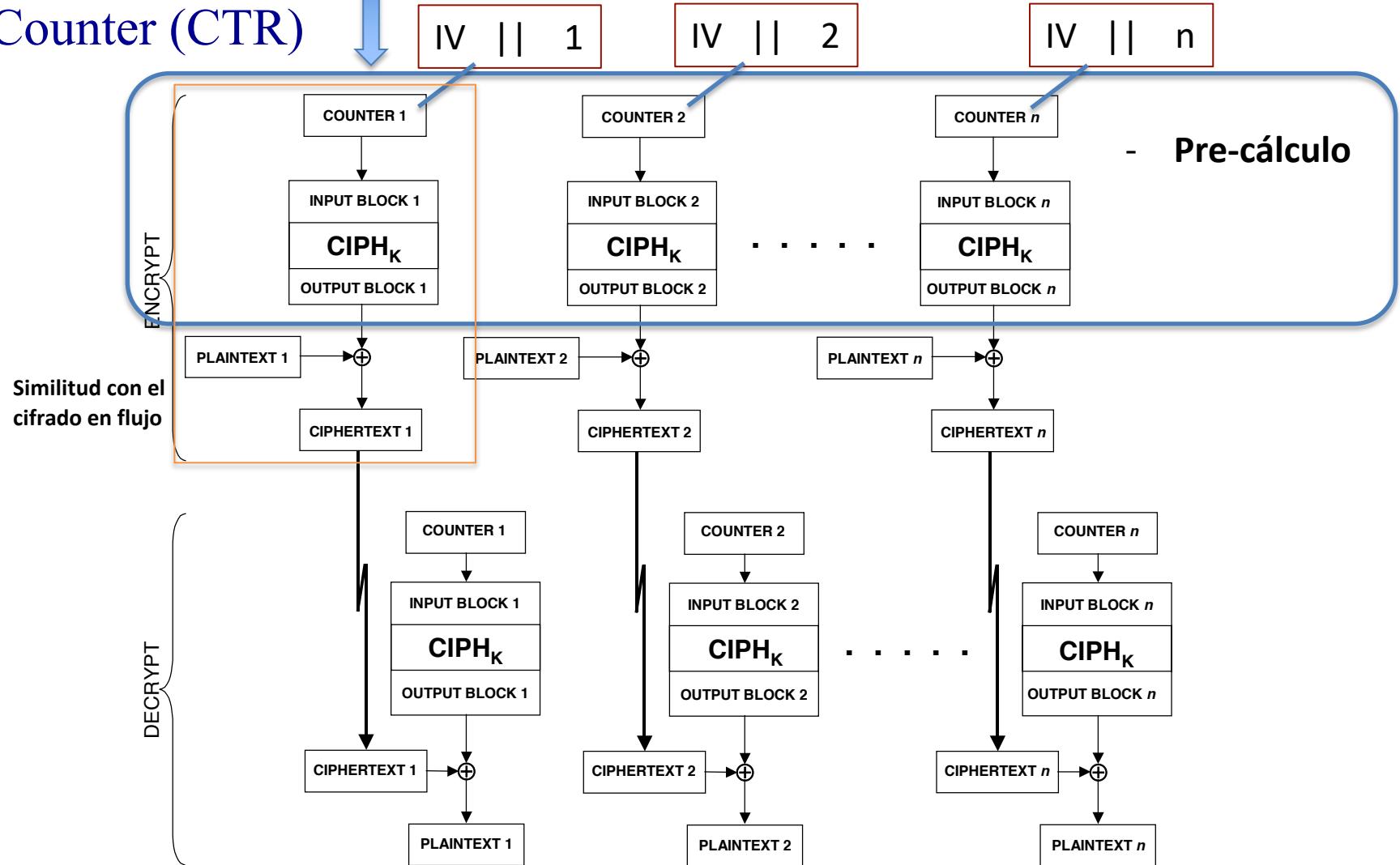


• Output Feedback (OFB)

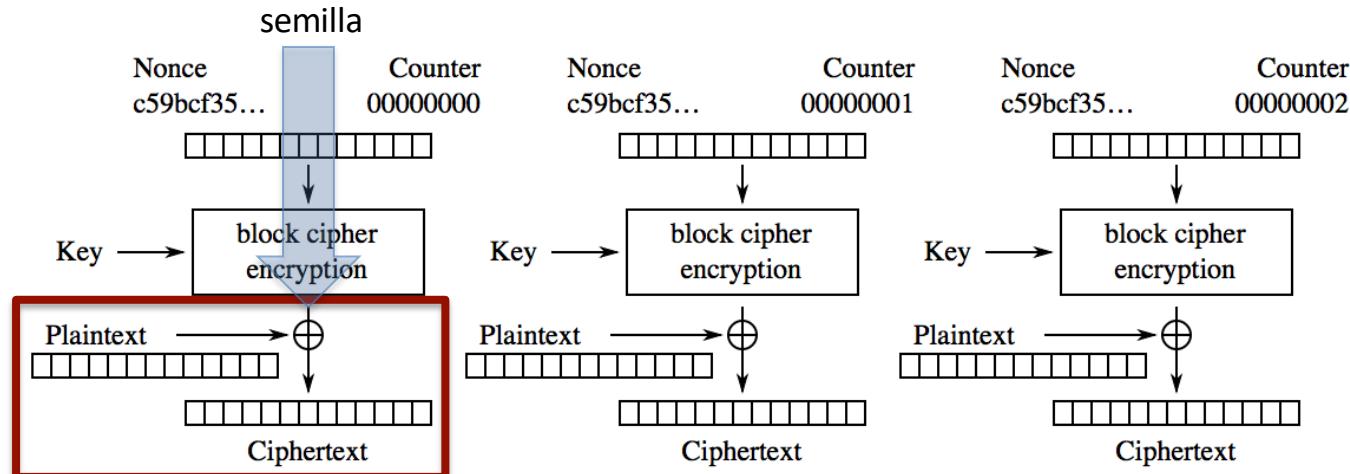


Normalmente se concatena el IV al contador

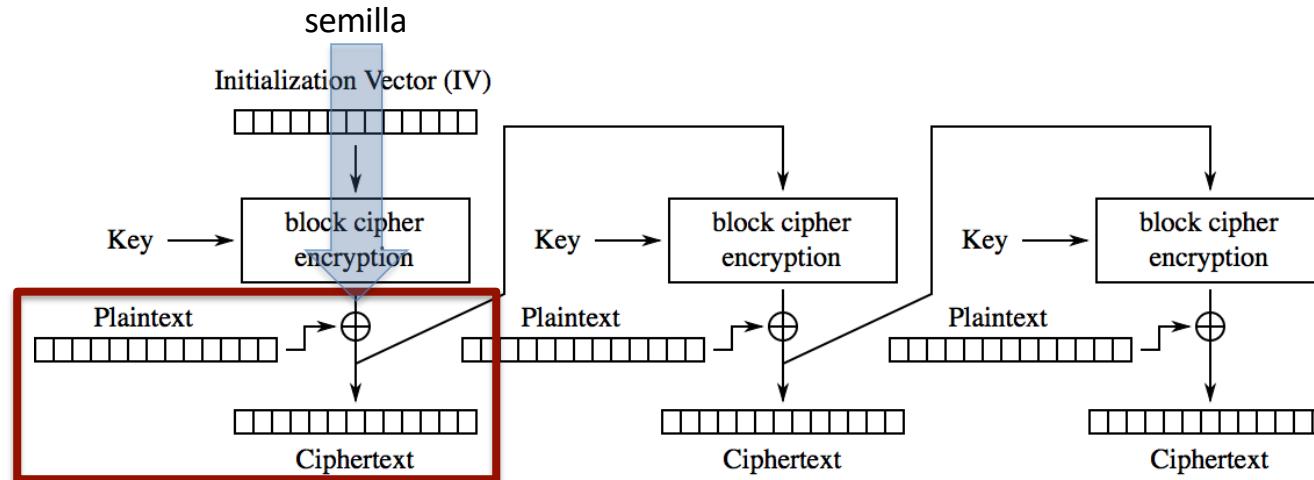
- Counter (CTR)



Otra forma de verlo - ¿ A qué se podría asemejar esto ?



Counter (CTR) mode encryption

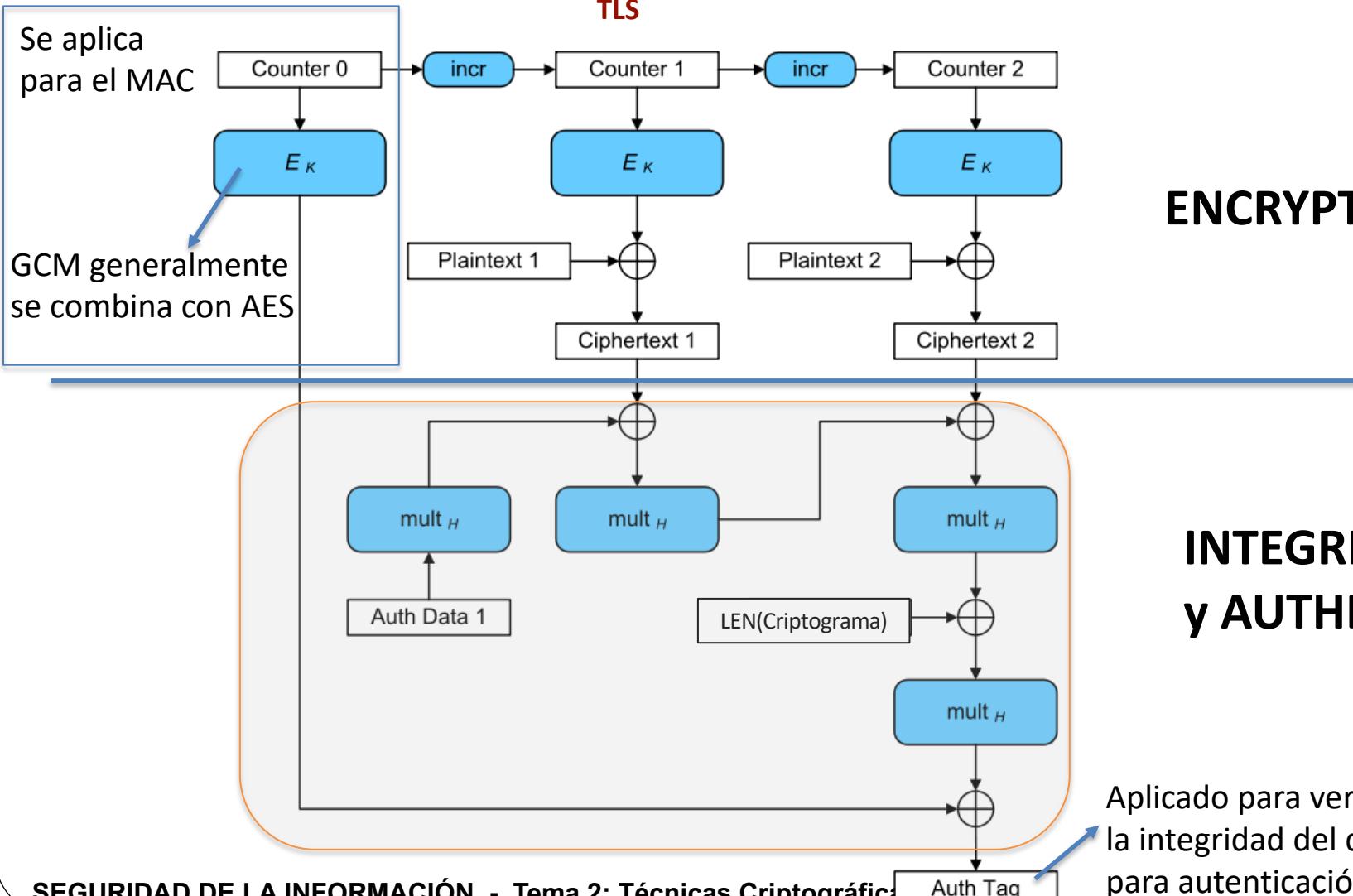


Cipher Feedback (CFB) mode encryption

Extra: Modos de operación con integridad

• Galois-CTR (GCM)

- Funciona de forma similar que el modo CTR pero usa Carter-Wegman MAC en un campo de Galois
- **Es rápido y eficiente, y está soportado por el suite de cifrado dado por TLS**



Ventajas y desventajas de los algoritmos simétricos

- **Ventajas:**
 - Los algoritmos simétricos se pueden diseñar para alcanzar un **alto rendimiento** (alto caudal de información cifrada)
 - En HW se pueden alcanzar del orden de cientos de Mbytes/sec
 - En SW se pueden alcanzar del orden Mbytes/sec
 - Se pueden **componer** para producir cifrados más fuertes
 - Recuerda: CIFRADO POR PRODUCTO
 - Se pueden utilizar como base para **construir otros mecanismos** criptográficos como:
 - **Funciones hash y**
 - **Generadores pseudo-aleatorios de números** (☺ i ya lo has visto !)
 - Los algoritmos simétricos necesitan:
 - **Claves K relativamente cortas**

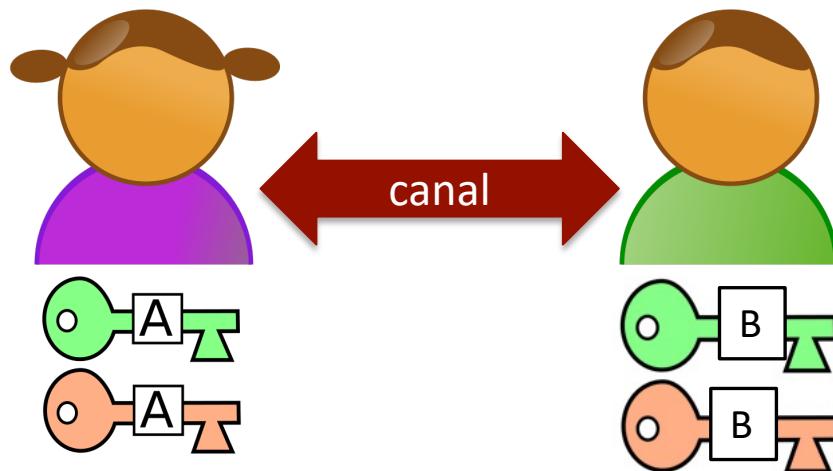
- **Desventajas:**
 - En una comunicación entre dos usuarios, estos han de **acordar, a priori, la clave K** con la que cifrarán/descifrarán sus comunicaciones
 - La clave ha de permanecer estrictamente en secreto, por lo que sólo la han conocer esos dos usuarios que se comunican
 - Si los dos usuarios están **físicamente lejanos** entre sí, acordar la clave puede convertirse en una tarea difícil
 - ¿Qué medio suficientemente seguro habrán de utilizar si no es posible una reunión presencial entre ambos?
 - Además, a efectos de seguridad, es recomendable que la clave K entre dos usuarios se cambie con cierta frecuencia, lo que complica el problema
 - En una red grande (de muchos usuarios) habrá demasiadas claves que administrar
 - Para una comunidad de n usuarios, el número de claves en el sistema será de:
 - $(n * (n-1)) / 2$; ej: 100 usuarios → 4950 claves
 - Probablemente, hará falta una **tercera parte confiable (ej. un administrador de claves)** para ayudar a los usuarios en las tareas de administración de claves

Algoritmos asimétricos (o de clave pública)



Introducción

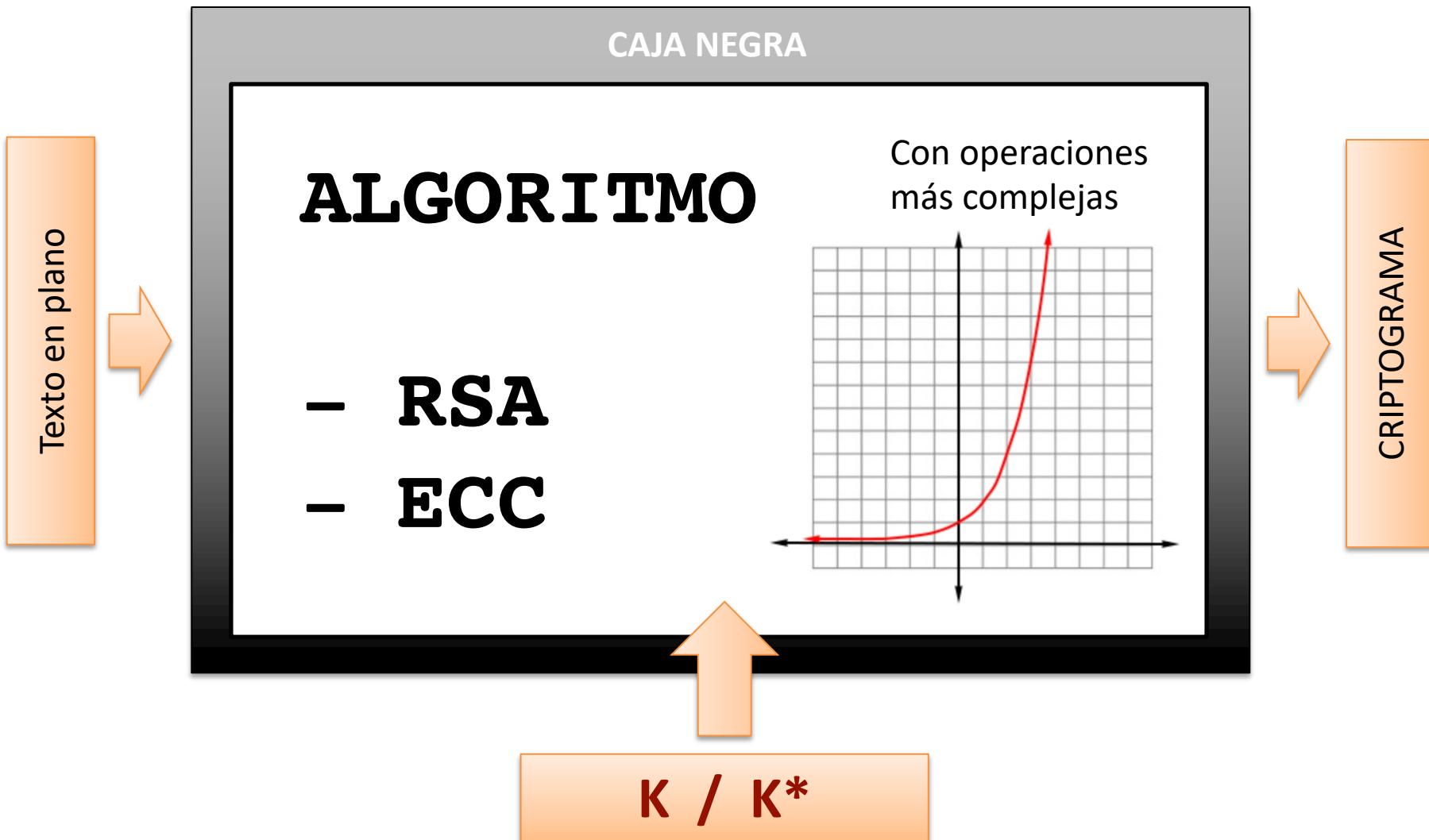
- El concepto de criptografía de clave pública (o asimétrica) fue inventado en 1976 por *Diffie y Hellman*, e independientemente por *Merkle*, para dar solución a algunos de los problemas de los criptosistemas simétricos



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

Introducción

Si lo vemos como cajas negras



Introducción

- **K / K*:**
 - 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
- Operaciones con K/K*:
 - Una clave se usa para cifrar, y la otra para descifrar
 - Es decir, 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 criptografía asimétrica



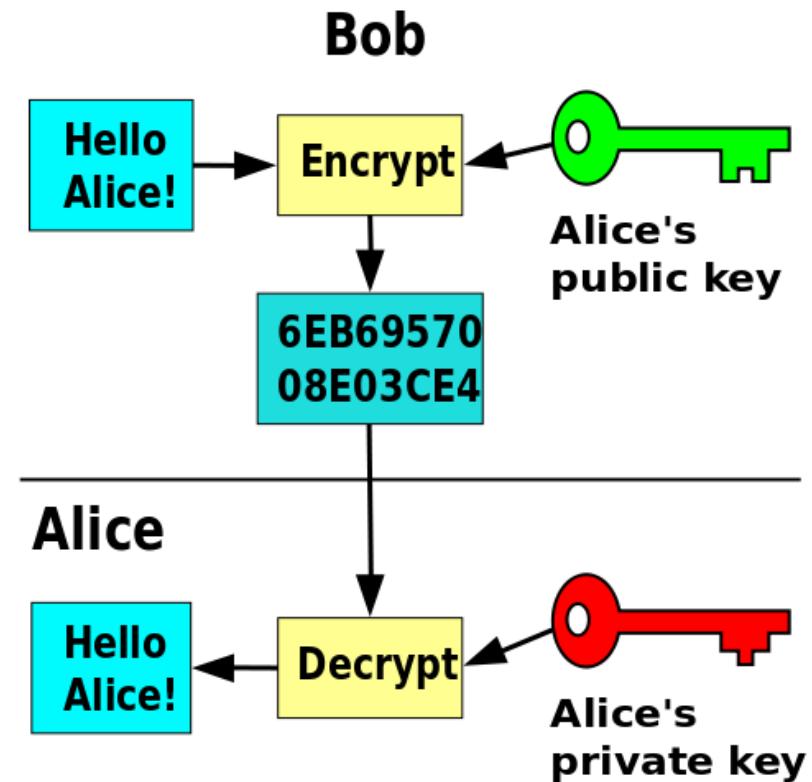
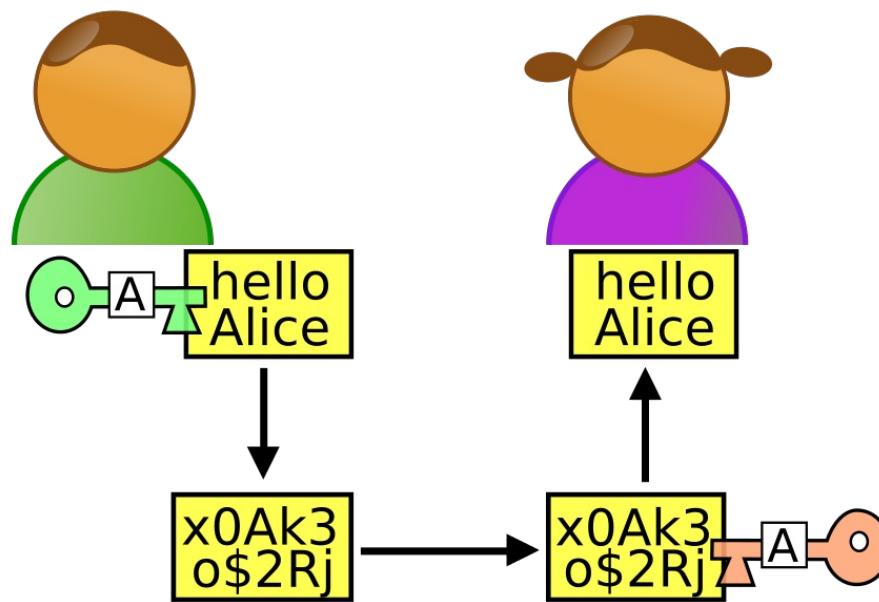
CIFRADO

FIRMA DIGITAL

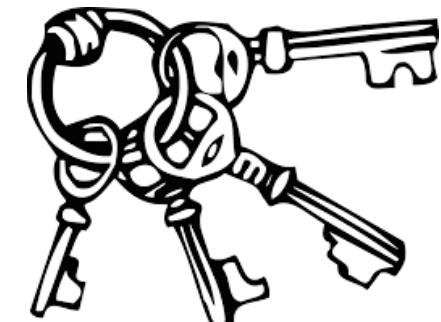
INTERCAMBIO DE
CLAVES

Cifrado/descifrado

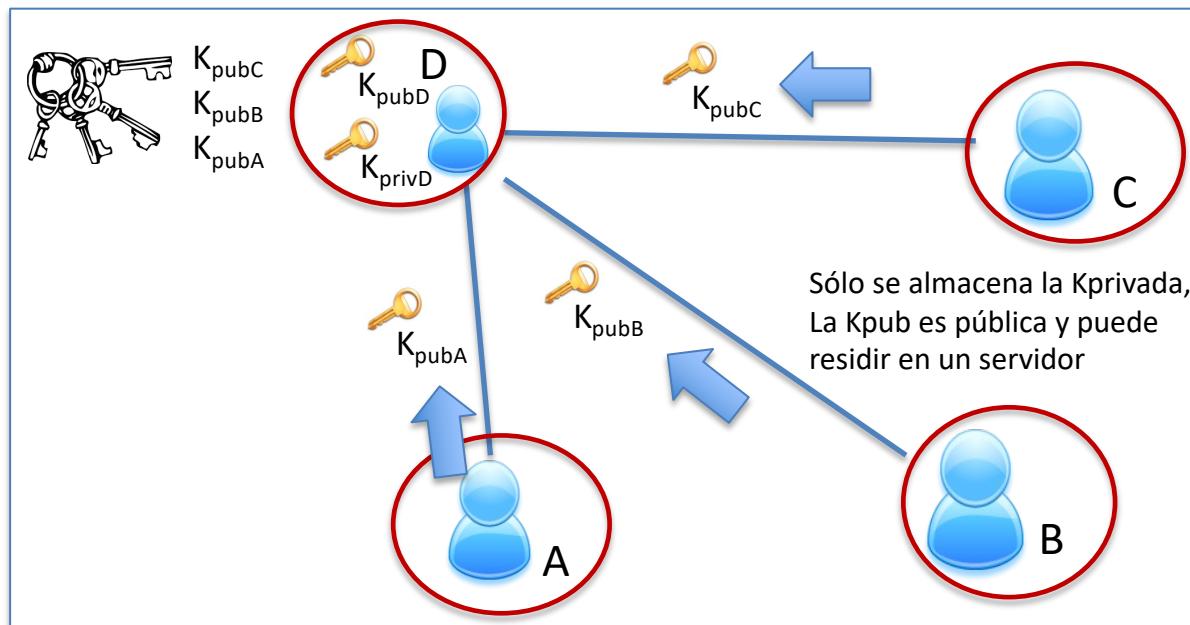
- En el caso de que *Bob* necesite del servicio de confidencialidad para su comunicación con *Alice*, el **cifrado/descifrado** se realiza de la siguiente forma:



- Del esquema anterior se desprende 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, existen varias soluciones para compartir la clave:
 - 1) Alice se lo proporciona mediante un conexión *peer-to-peer*
 - 2) Alice se lo proporciona por una de las vías más comunes: email, WhatsApp, sms, etc.
 - 3) Alice se lo deja en un repositorio común: foro, servidor de claves, etc.
- En cualquiera de los casos, *Bob* debe almacenar todas las claves públicas de todos los usuarios en un **key-ring** personal



- De lo anterior, también se deduce 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



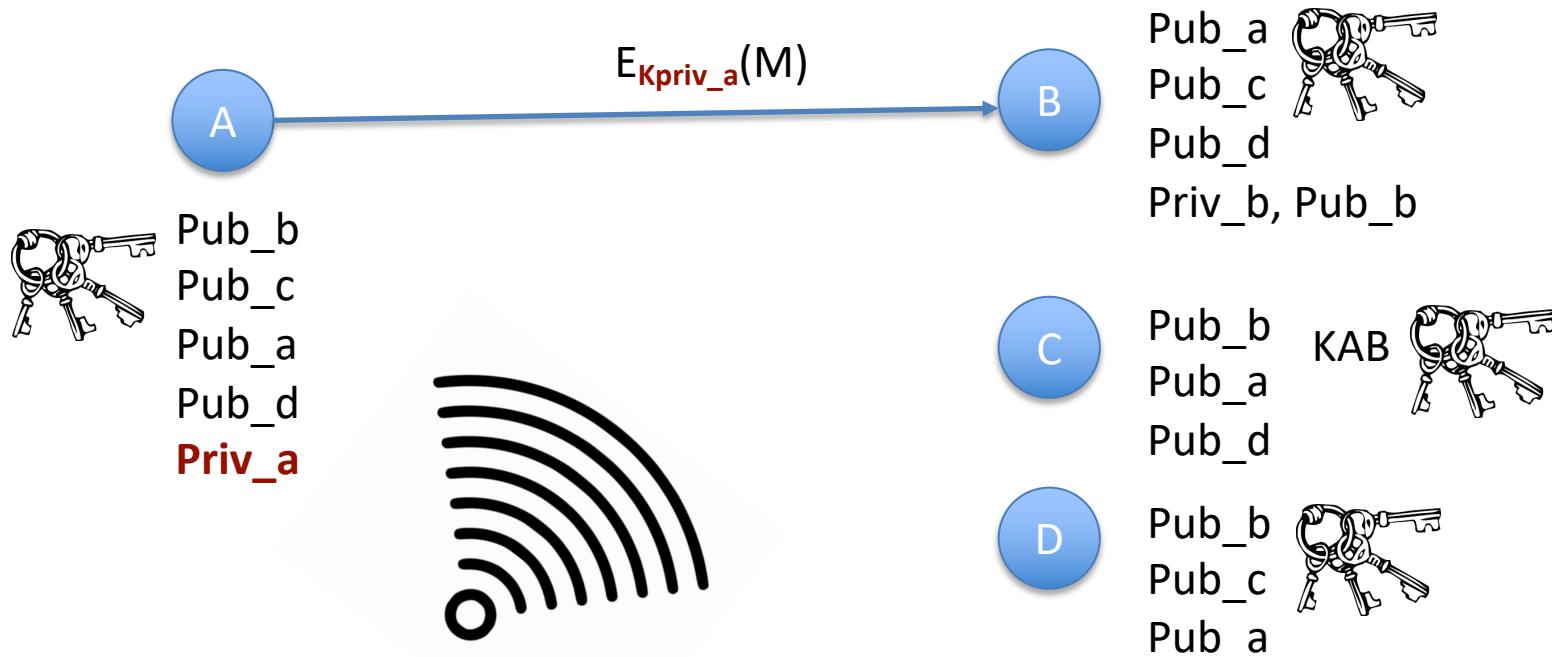
- Otras características relevantes que deberías saber:
 - Es **computacionalmente imposible deducir la clave privada** del usuario U a partir de su clave pública
 - Cualquier usuario con la **clave pública de U puede cifrar un mensaje hacia U** , pero no descifrarlo
 - Cifrar el mensaje con la clave 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



LO QUE UNA HACE, LA OTRA LA DESHACE

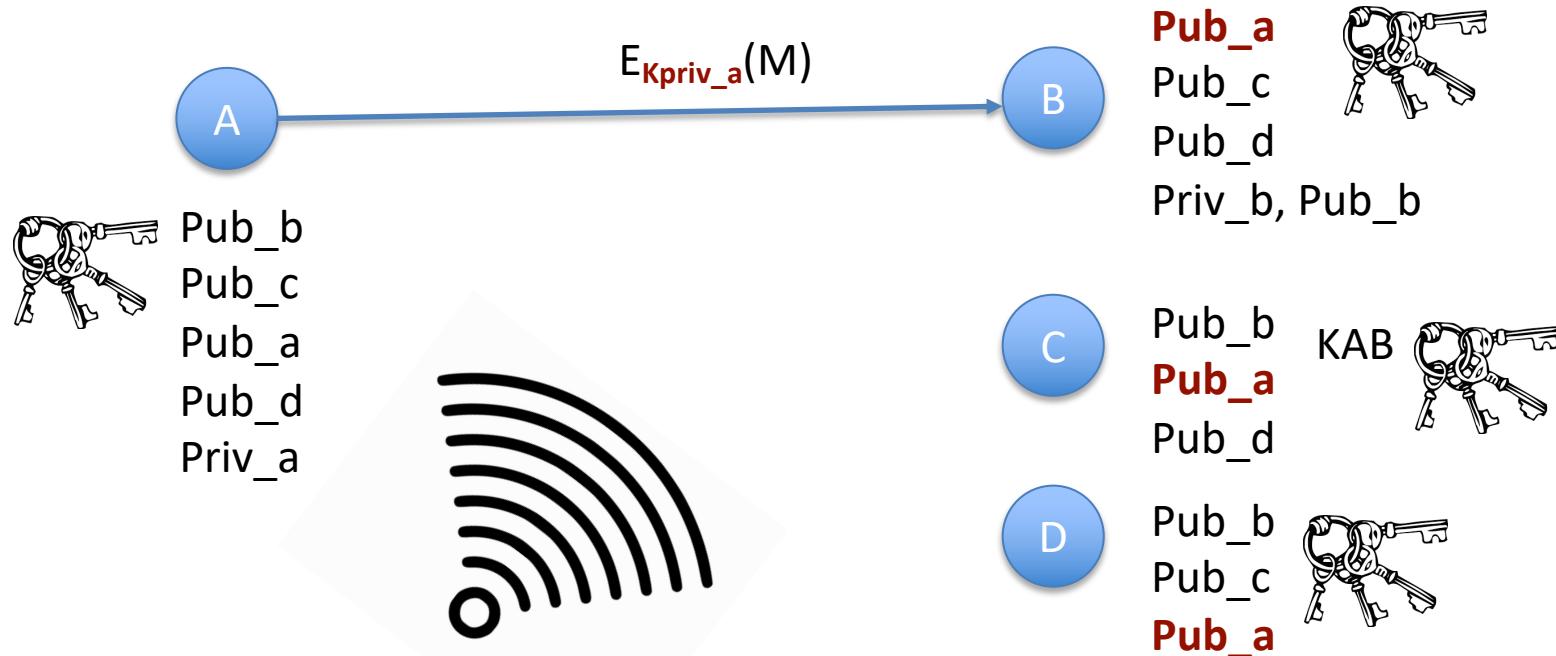
Algunos casos de ejemplos - Queremos confidencialidad

CASO A: A envía a B un mensaje con su clave privada



Algunos casos de ejemplos - Queremos confidencialidad

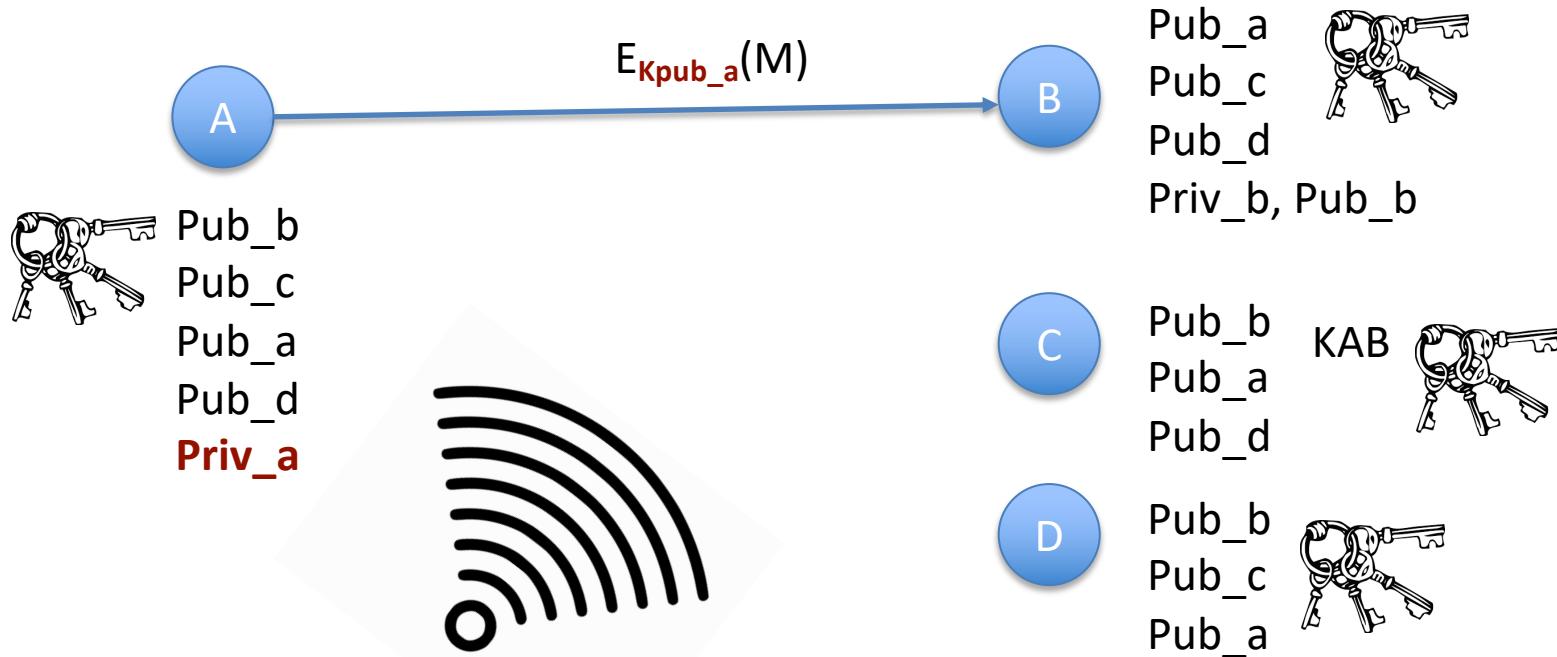
CASO A: A envía a B un mensaje con su clave privada - **¡¡ NO !!**



Si se cifra el mensaje con la clave privada → **NO hay confidencialidad** porque no sólo B tiene la $Kpub_a$, sino también C y D, por lo que ellos también podrían leer el mensaje

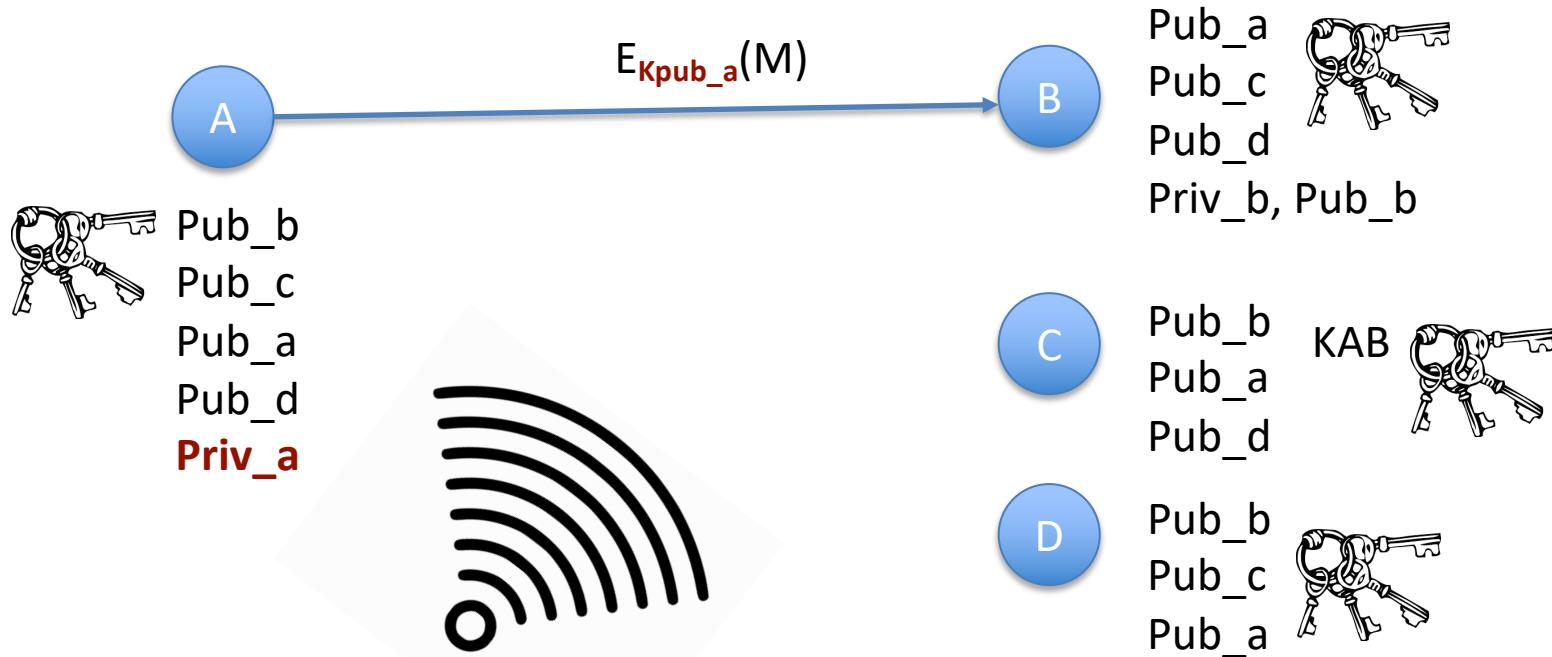
Algunos casos de ejemplos - Queremos confidencialidad

CASO B: A envía a B un mensaje con su clave pública



Algunos casos de ejemplos - Queremos confidencialidad

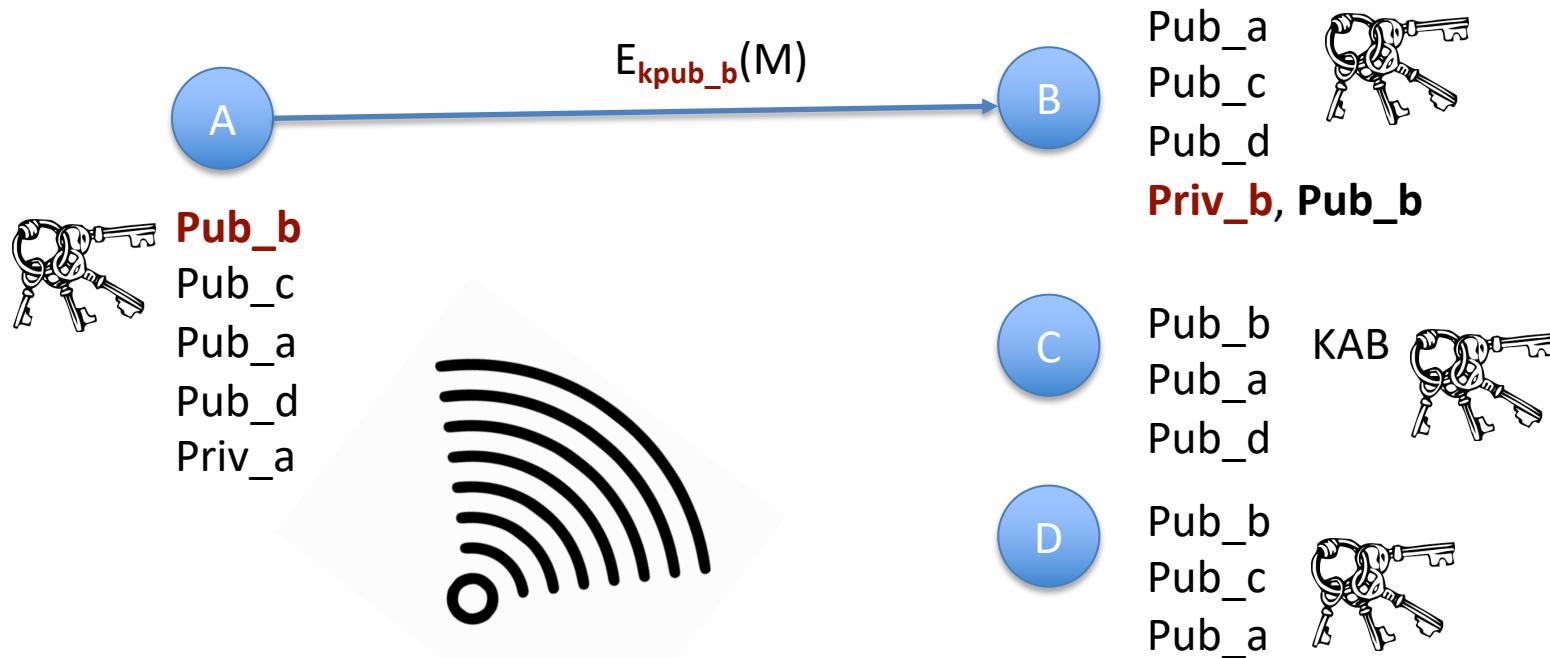
CASO B: A envía a B un mensaje con su clave pública - **¡¡ NO !!**



Si se cifra el mensaje con su propia clave pública → ¡¡ Ni la otra parte ni el resto de comunicación podrán leer el mensaje !! Porque el que tiene la clave privada es el que ha enviado el mensaje

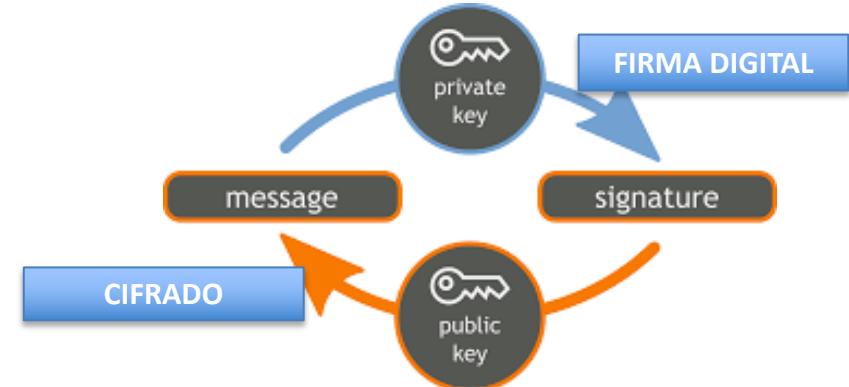
Algunos casos de ejemplos - Queremos confidencialidad

CASO C: A envía a B un mensaje con la clave pública de B - **¡¡ SÍ !!**



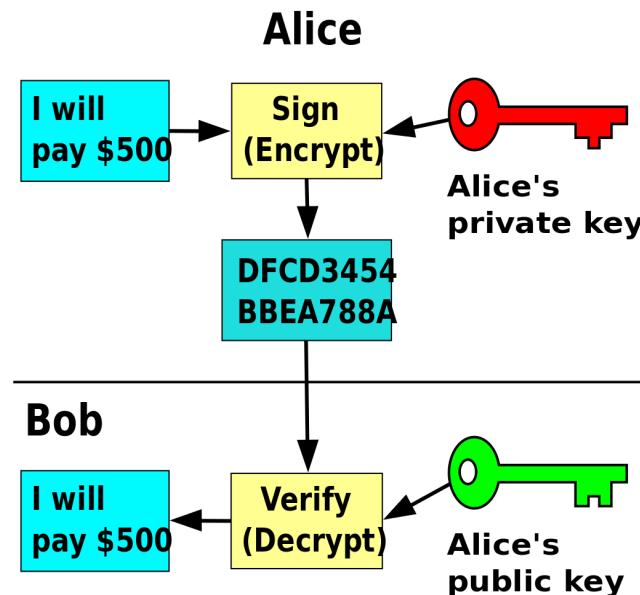
Si se cifra el mensaje con la clave pública de B → **HAY
confidencialidad** porque sólo B tiene la Kpriv_b asociada a Kpub_b, y C y D no podrán leer el mensaje

- 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



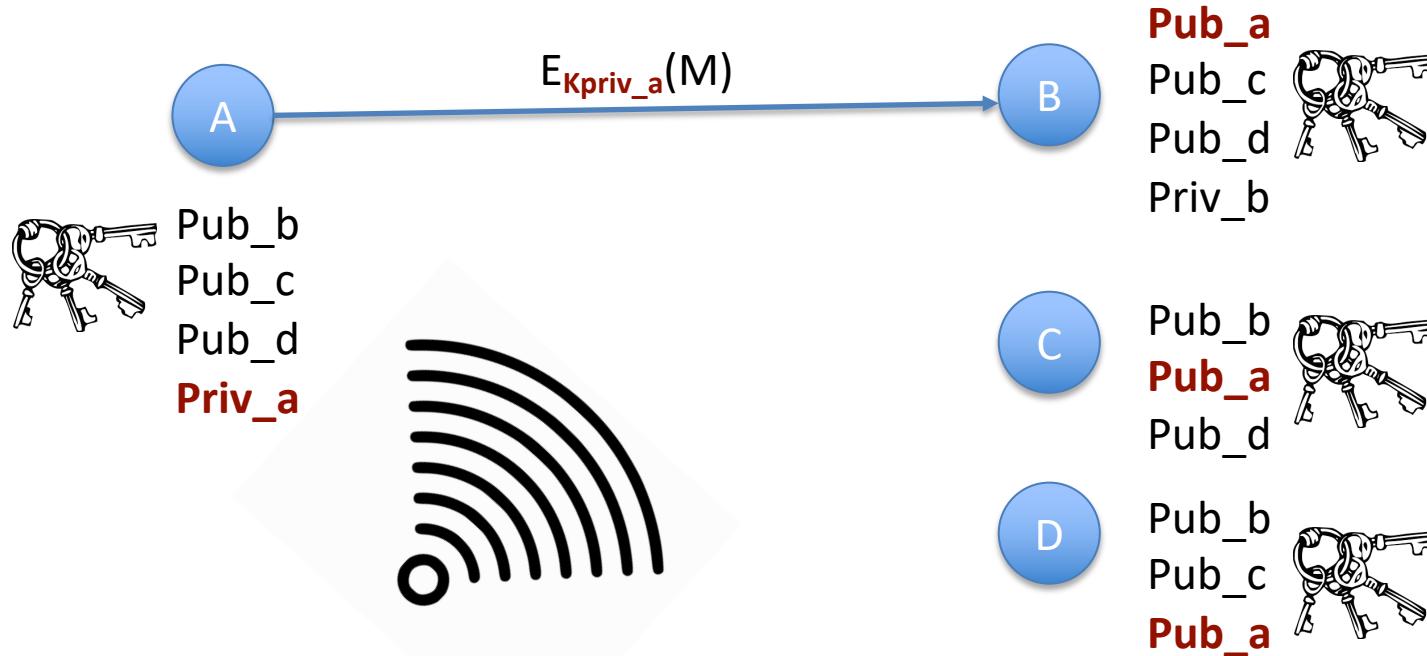
Firma digital

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



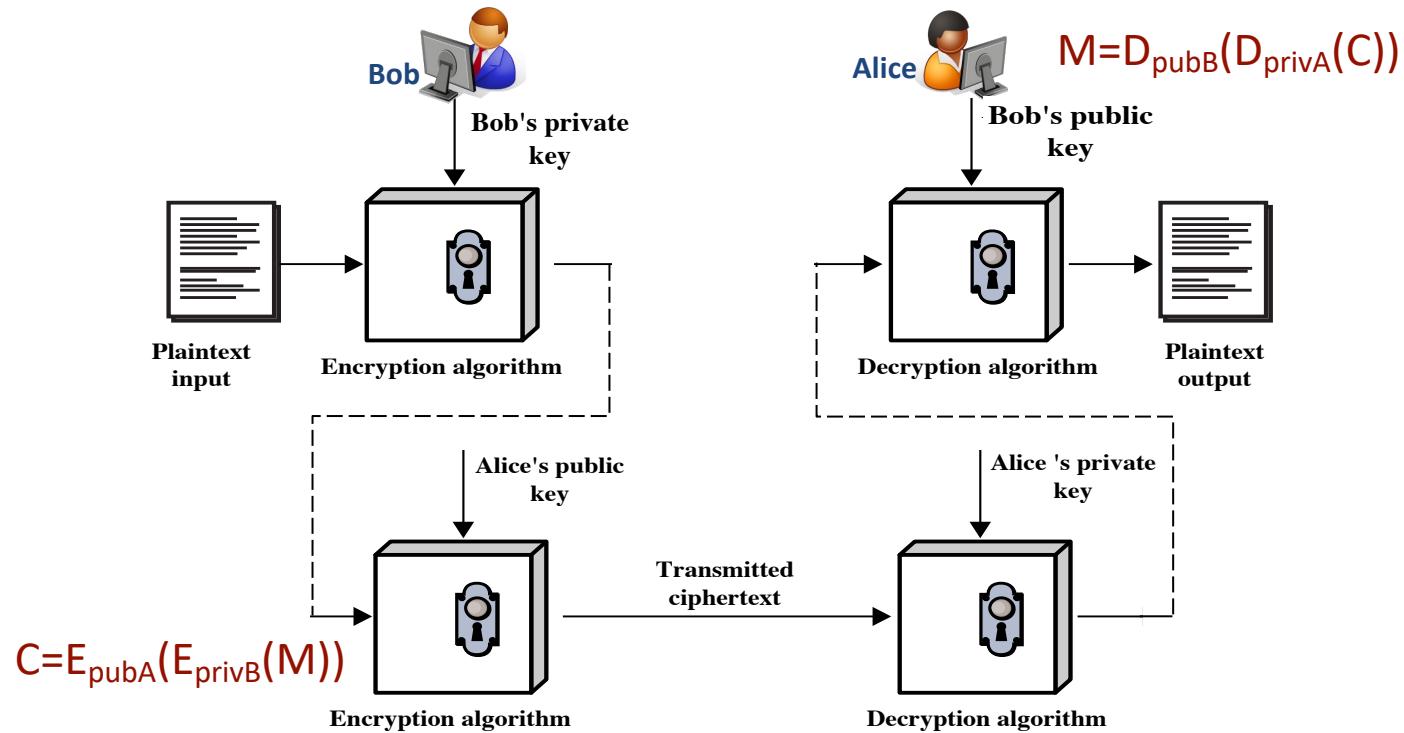
Algunos casos de ejemplos - Queremos autenticación

CASO A: A envía a B un mensaje firmado con su clave privada - **¡¡ SÍ !!**



Firma digital + Cifrado

- 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, por ejemplo, aplican **claves de gran tamaño**
 - 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 las convencionales sustituciones, permutaciones y sumas de los criptosistemas 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
```

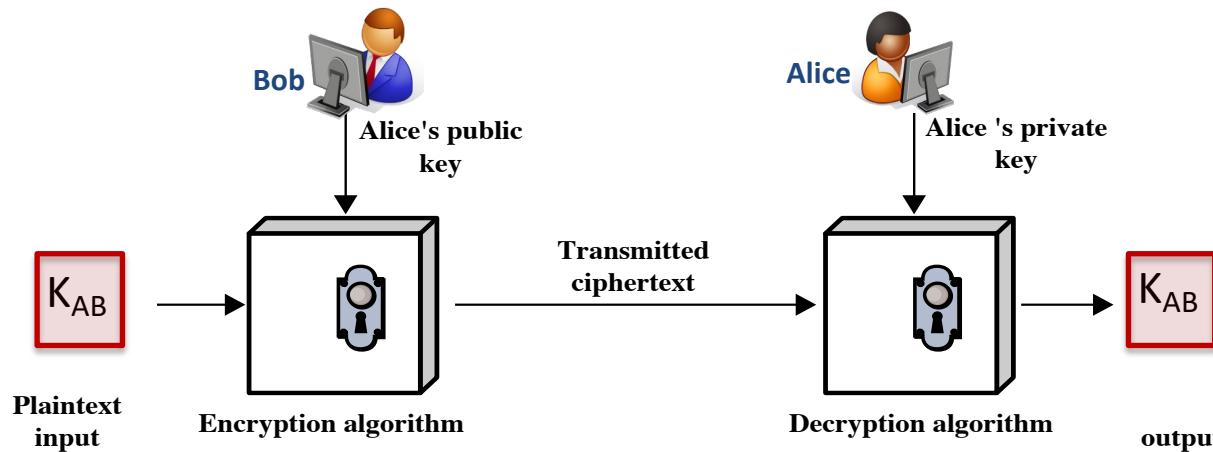
To get the most accurate results, try to run this
program when this computer is idle.
Doing 512 bit private rsa's for 10s: 79651 512 bit private RSA's in 9.98s
Doing 512 bit public rsa's for 10s: 1079143 512 bit public RSA's in 9.95s
Doing 1024 bit private rsa's for 10s: 22746 1024 bit private RSA's 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 → Criptografía híbrida

- 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 un criptosistema de clave pública + un criptosistema simétrico

Intercambio de claves → Criptografía híbrida

C. Simétrica

R+:

- los algoritmos son más simples y demandan menos recursos

S-:

- las claves son pequeñas y se pueden derivar por fuerza bruta
- Las claves son inseguras y requieren frecuentes procesos de rekeying

C. Asimétrica

S+:

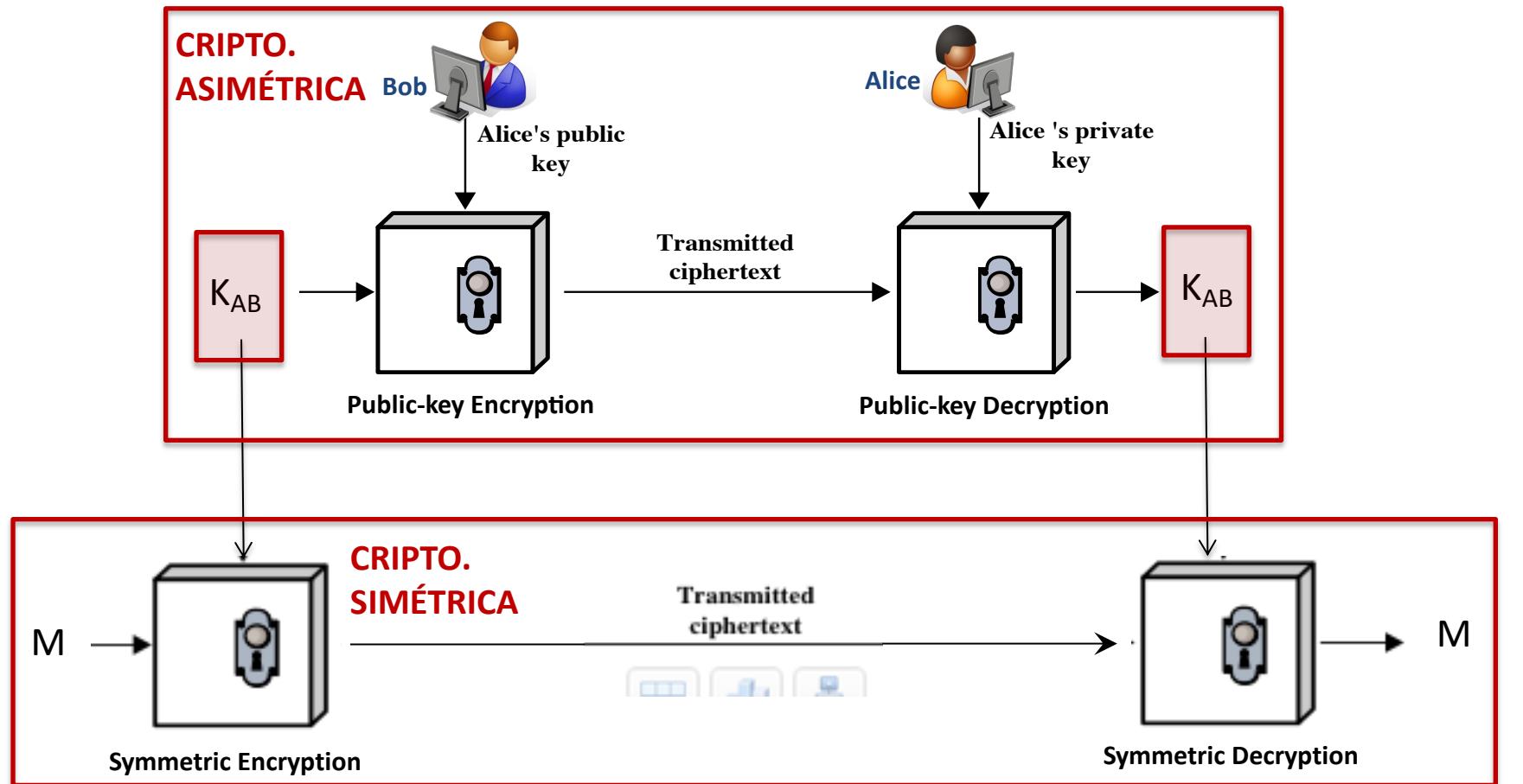
- las claves son de tamaño mayor, y la Kpriv no se puede (computalmente y matemáticamente hablando) derivar de la Kpub
- Los algoritmos son más robustos frente ataques y permite enviar datos seguros por cualquier medio

R-:

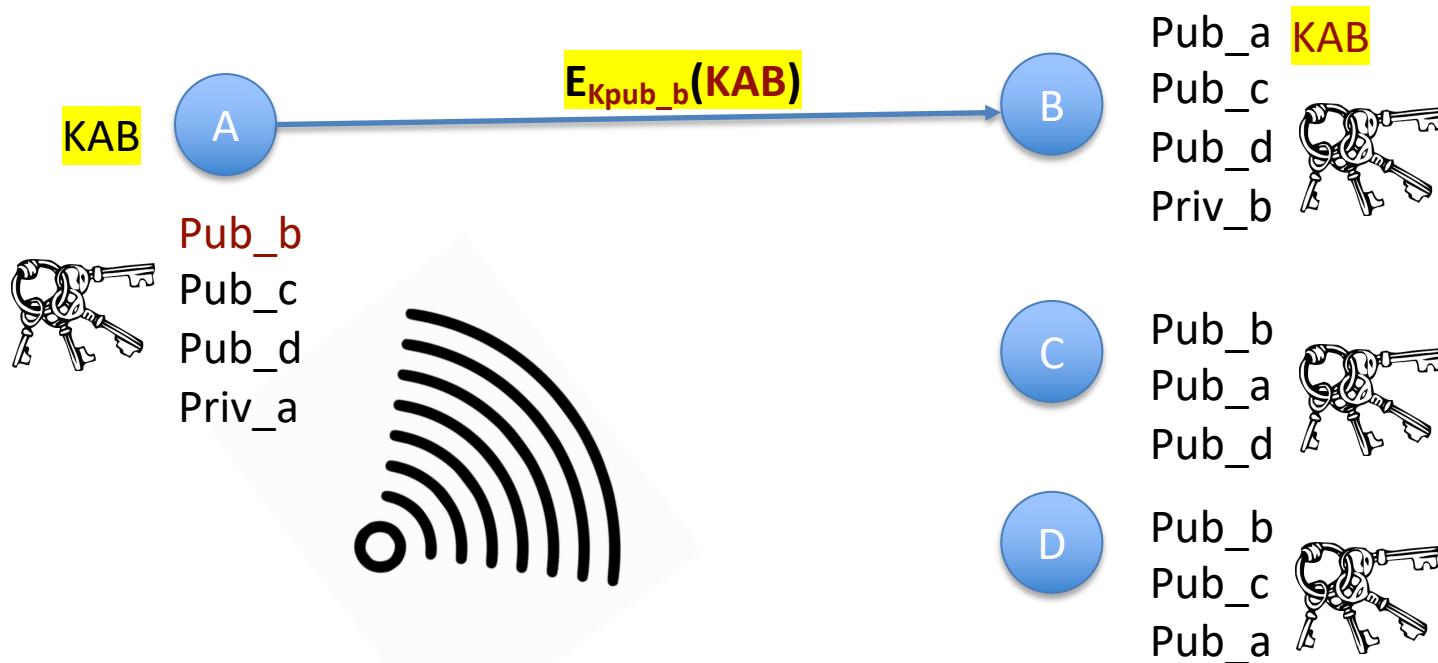
- los algoritmos son complejos y las claves son grandes, lo que demanda recursos para computar el crifrado

Intercambio de claves → Criptografía híbrida

- Una evolución del planteamiento anterior es el siguiente:
 - Enviar simultáneamente la clave de sesión y el mensaje cifrado con esa misma clave para **aprovechar los canales de comunicación**

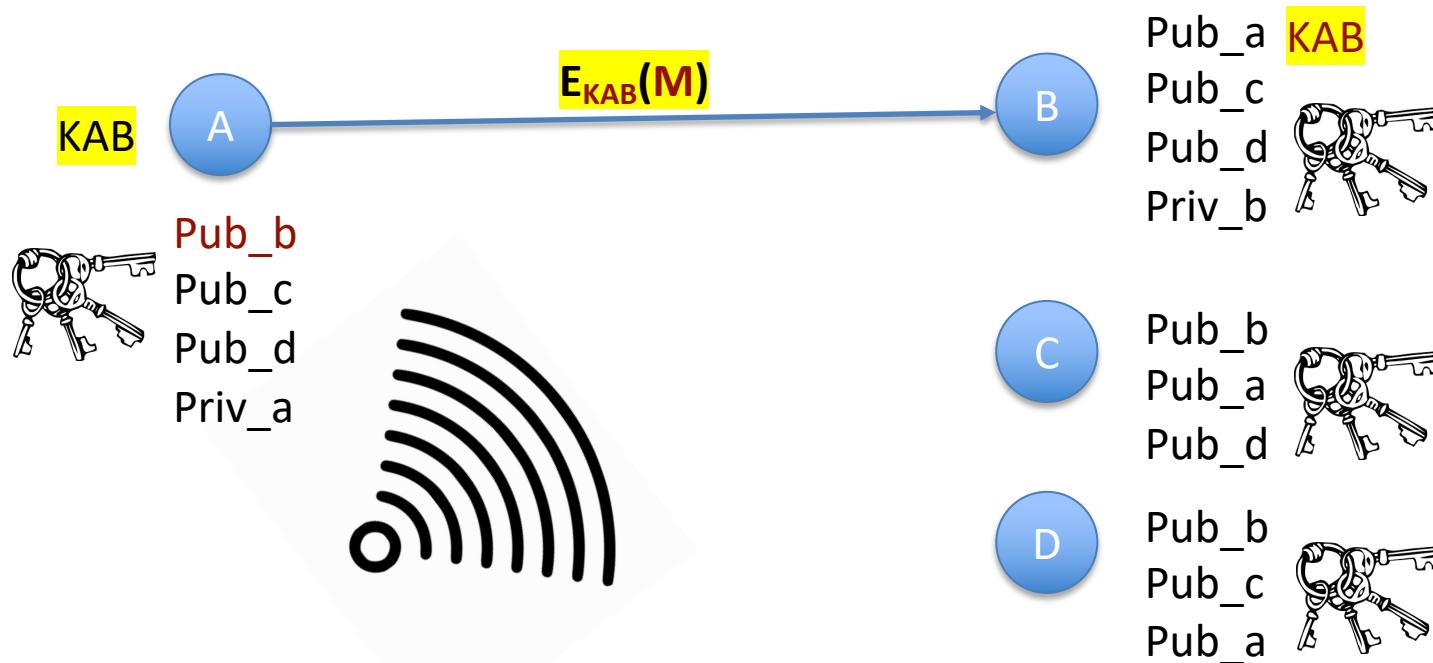


Algunos casos de ejemplos - Queremos criptografía híbrida



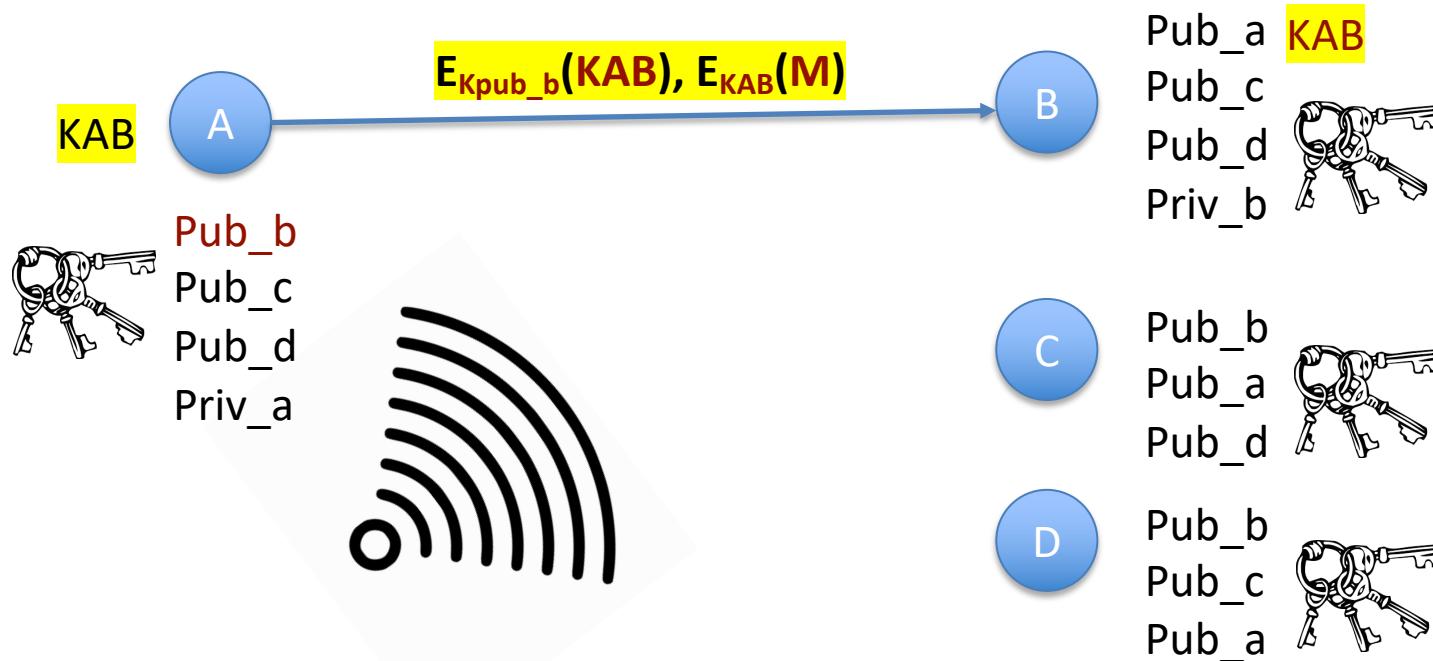
Fase 1: PROTEGER la KAB

Algunos casos de ejemplos - Queremos criptografía híbrida



Fase 2: PROTEGER el mensaje M

Algunos casos de ejemplos - Queremos criptografía híbrida



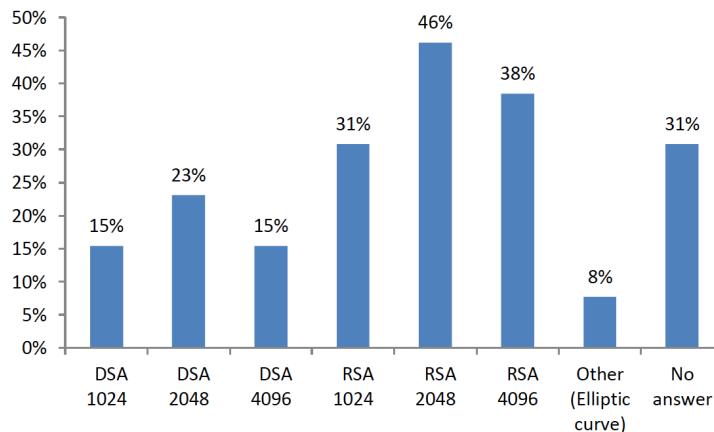
Fase 1 + 2: para proteger el
canal de comunicaciones

Funcionalidades de la criptografía de clave pública

- Como se muestra en la siguiente tabla, no todos los algoritmos de clave pública son capaces de realizar las tres funcionalidades mencionadas:

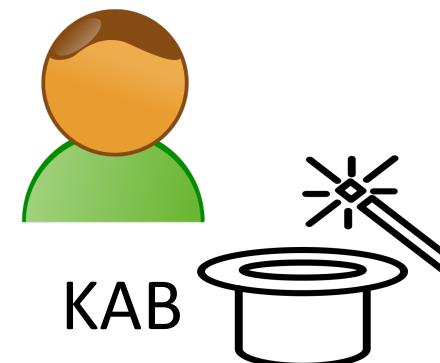
Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

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



Algoritmo Diffie-Hellman

- Diseñado en 1976 y se conoce como el “**algoritmo Diffie-Hellman (DH) 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
 - Criptografía híbrida ☺
- Básicamente, DH permite que dos entidades (A y B) puedan generar una clave KAB de forma simultánea, y sin enviarla por el canal de comunicaciones



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

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

1) Para ello, A y B necesitan establecer y **compartir valores comunes**, como un valor q primo y una raíz primitiva α de q

2) Tanto A como B generan sus claves privadas ($X_{A/B}$) y públicas ($Y_{A/B}$) teniendo en cuenta que: $Y_A \equiv \alpha^{X_A} \pmod{q}$, donde $0 \leq X_A \leq (q-1)$, X_A es el **logaritmo discreto de Y_A** , y se representa $dlog_{\alpha,q}(Y_A)$

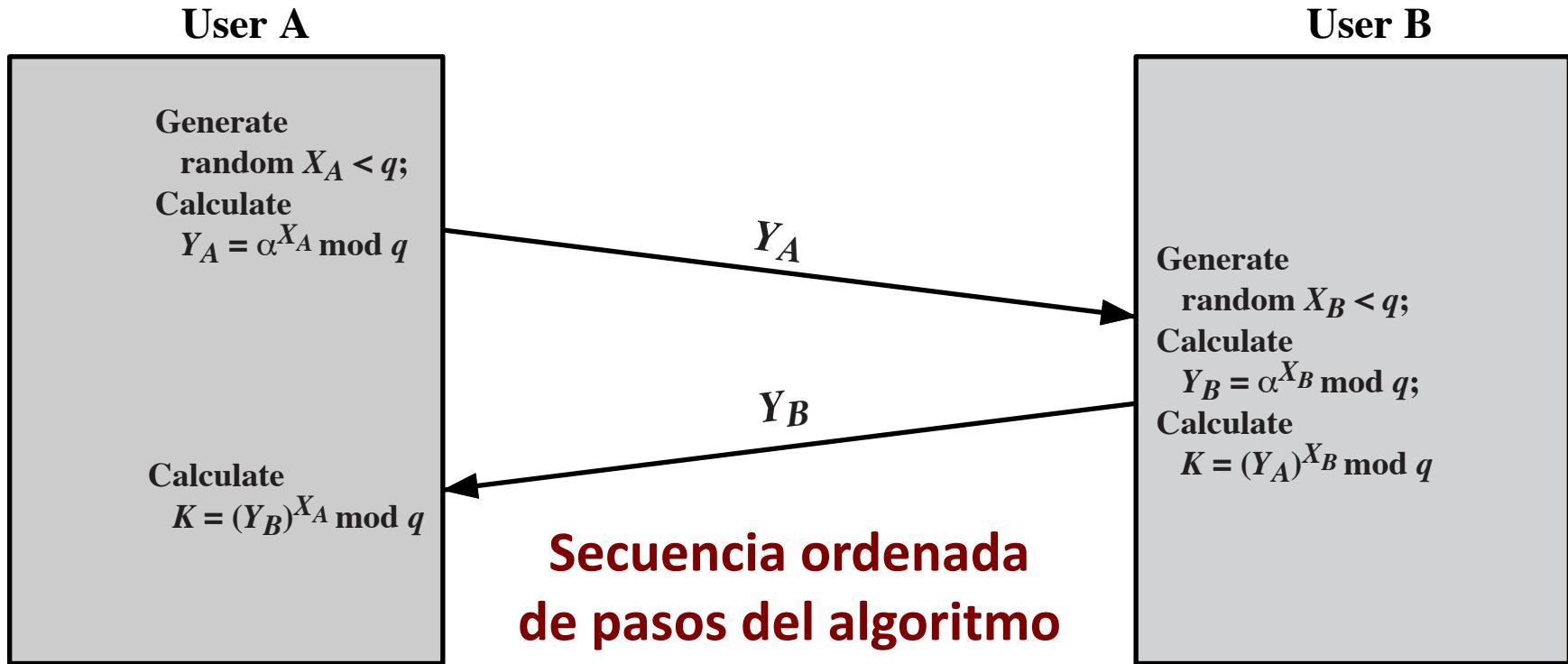


Luego, la efectividad del algoritmo depende de la **dificultad de computar logaritmos discretos**

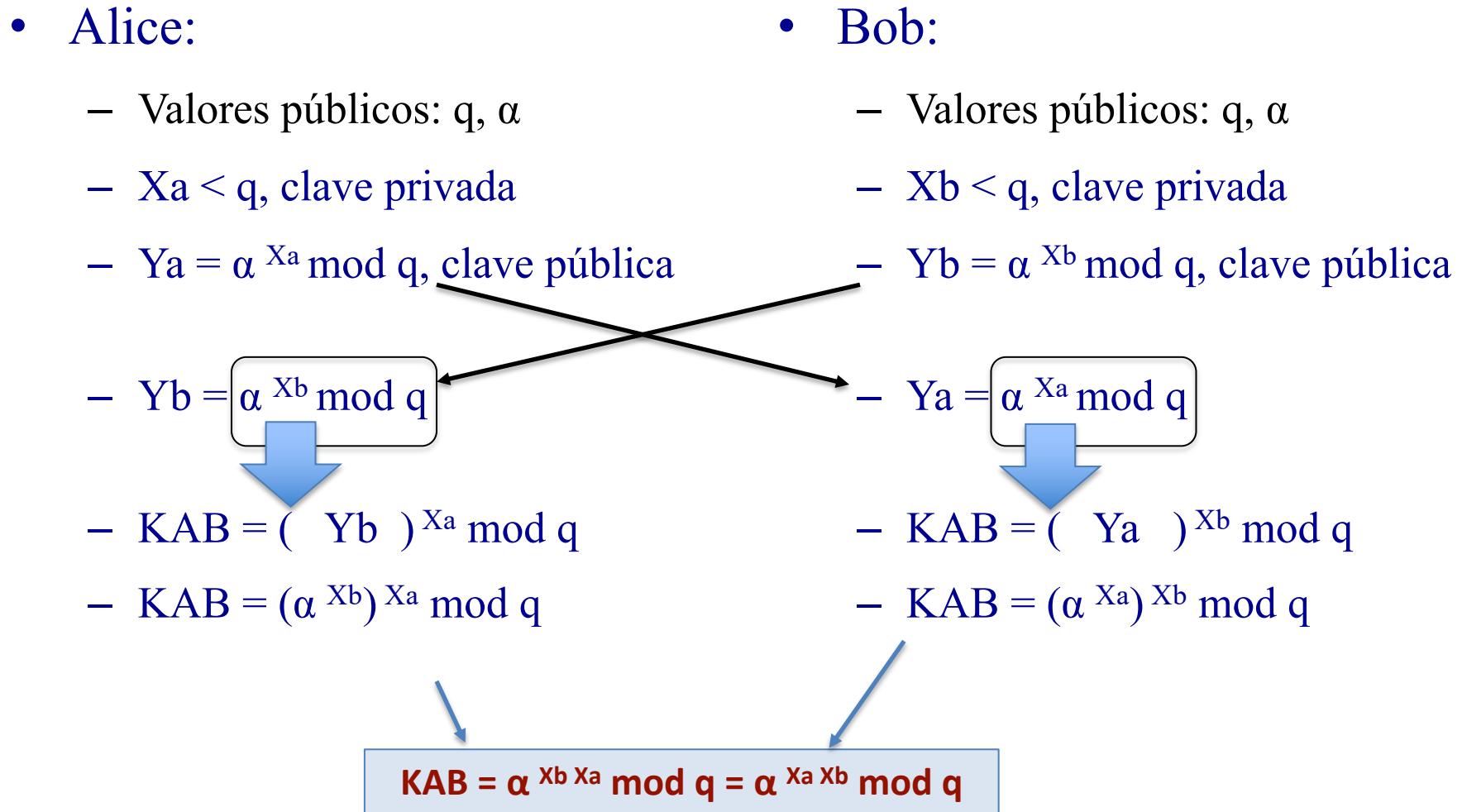
3) Tanto A como B comparten sus respectivas claves públicas (Y_A / Y_B)

4) Tanto A como B generan de forma "mágica" la clave de sesión teniendo en cuenta: **$K_{AB} = (Y_B)^{X_A} \text{ mod } q$**

Otra forma de verlo ...

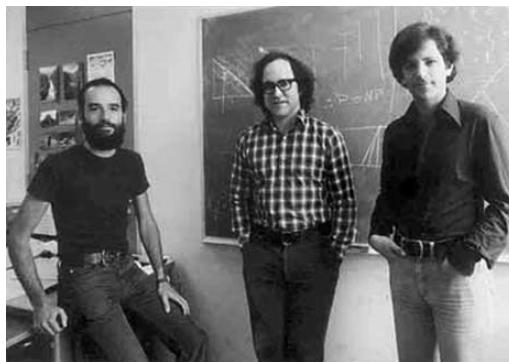


Otra forma de verlo ...



Algoritmo RSA

- Actualmente, es el criptosistema asimétrico o de clave pública más usado en la vida real
 - Su nombre procede de sus inventores: *Rivest, Shamir y Adleman* del Instituto Tecnológico de Massachusetts (MIT), y fue definido en 1977
- El criptosistema se basa de 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 $\phi(n)$, de forma que

$$\phi(n) = (p - 1) * (q - 1)$$



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

- se elige aleatoriamente un número grande e tal que

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

(o sea, e y $\phi(n)$ son primos relativos o coprimos)

- Se determina el número d que cumple

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

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

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 : $d \cdot e \equiv 1 \pmod{160}$

$$d = 23 \text{ dado que } 23 \times 7 = 161 \pmod{160} = 1$$

⑥ Clave pública = 7 y módulo = 187

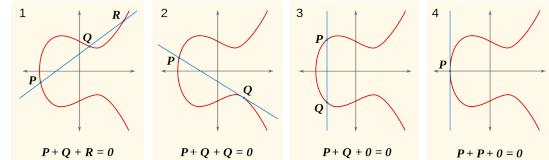
⑦ Clave privada = 23

- 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

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

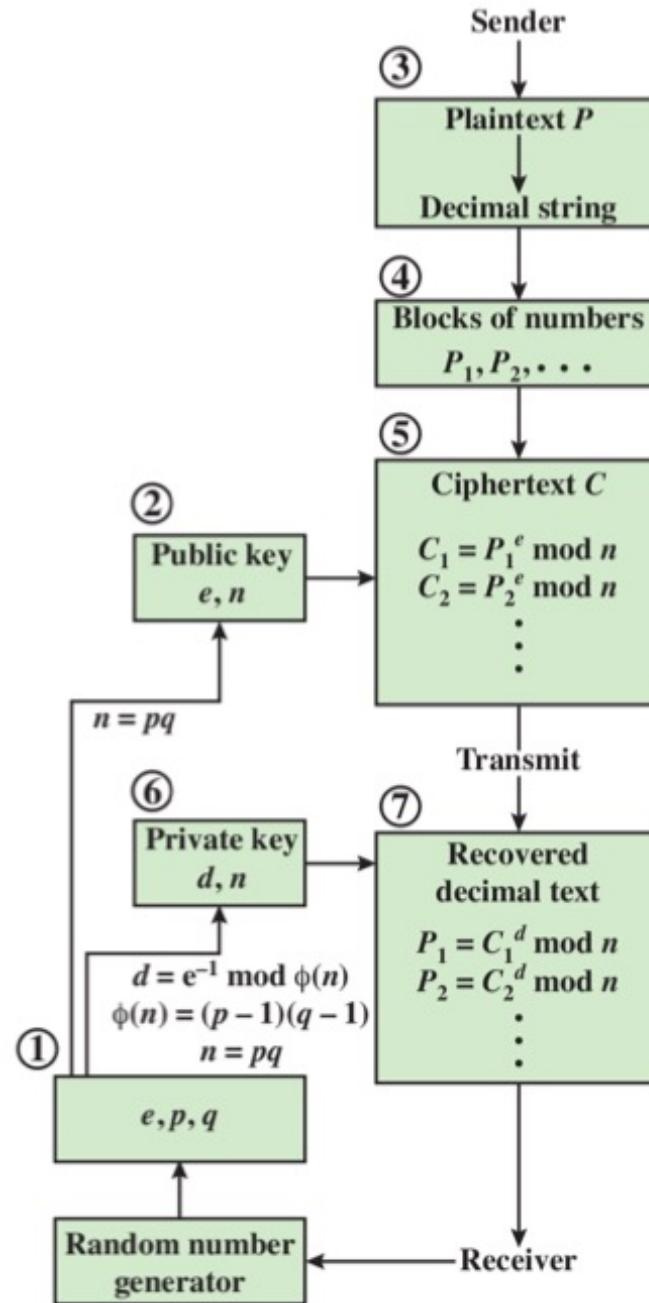
ALGO MUY
PARECIDO A LO QUE
SE HACE CON EL
CIFRADO EN BLOQUE

- Cosas que hay que tener en cuenta:
 - 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
 - También hay que tener cuidado al elegir el tamaño
 - **Cuanto mayor sea, más seguro, pero también más lento** será el sistema en sus cálculos
 - Para seleccionar p ó q debe testearse si es primo o no con cualquiera de los tests existentes (se recomienda *Miller-Rabin*)
 - 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: ECC

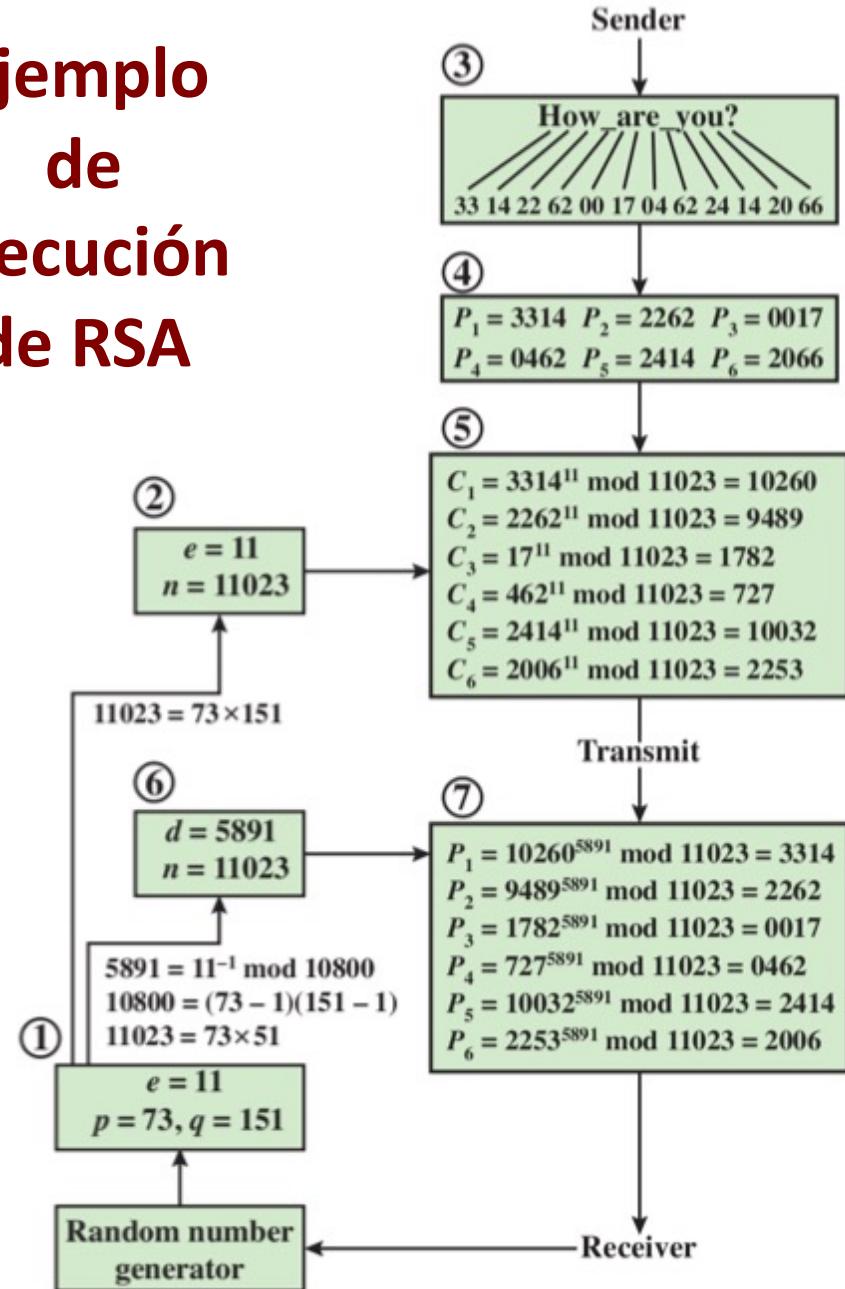


Tamaño de clave simétrica (bits)	Tamaño de clave RSA y DH (bits)	Tamaño de clave de ECC (bits)
128	1024	160
192	2048	224
256	3072	256
	7680	384
	15360	521

Recomendación dada por el NIST para el tamaño de claves



Ejemplo de ejecución de RSA



Padding

- Los cifrados en bloque están diseñados para trabajar con mensajes compuestos de bloques de un tamaño específico
 - Ejemplo: AES-128 trabaja con bloques de 128 bits (16 bytes)
 - Problema: Supongamos que usamos AES-128 (16 bytes), ¿Qué ocurre cuando queremos cifrar un mensaje que ocupa, por ejemplo, 20 bytes?
 - Tendremos un primer bloque de 16 bytes, y un segundo bloque de 4 bytes
 - El primer bloque lo podemos cifrar sin problemas
 - Al segundo bloque tenemos que añadirle algo al final (“**padding**”)

Padding

- Esquemas de Padding:
 - ISO 10126: añadir bytes aleatorios, excepto el último, que indicará la longitud del padding
 - |12 63 12 65 E7 82 A7 C1|B7 02 9E 29 4E 8C 7B 05|
 - ISO/IEC 7816-4: añadir ceros, excepto el primero, que siempre tendrá el valor 80:
 - |12 63 12 65 e7 82 a7 c1|b7 02 9e 80 00 00 00 00|
 - Zero Padding: simplemente añadir ceros
 - |12 63 12 65 e7 82 a7 c1|b7 02 9e 00 00 00 00 00|
 - Problema: si el mensaje original acaba en alguna secuencia de ceros, no es posible determinar dónde empieza el padding
 - PKCS#5, PKCS#7: Si necesitamos N bytes de padding, usamos N veces el valor N
 - |12 63 12 65 e7 82 a7 c1|b7 02 9e 05 05 05 05 05|

Otras primitivas criptográficas

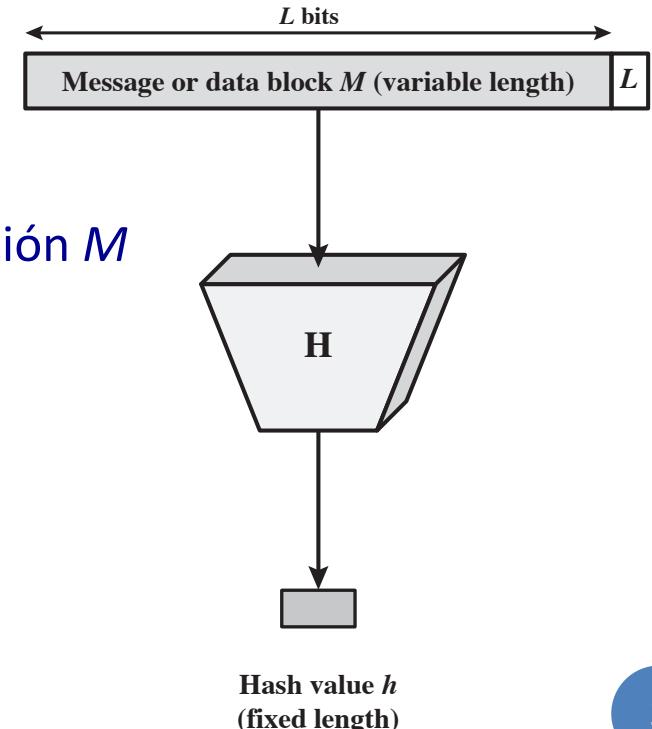


Funciones Hash

- 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 para el cifrado
 - porque un mensaje cifrado mediante una función unidireccional no se podría descifrar

- Una función hash es una función que:

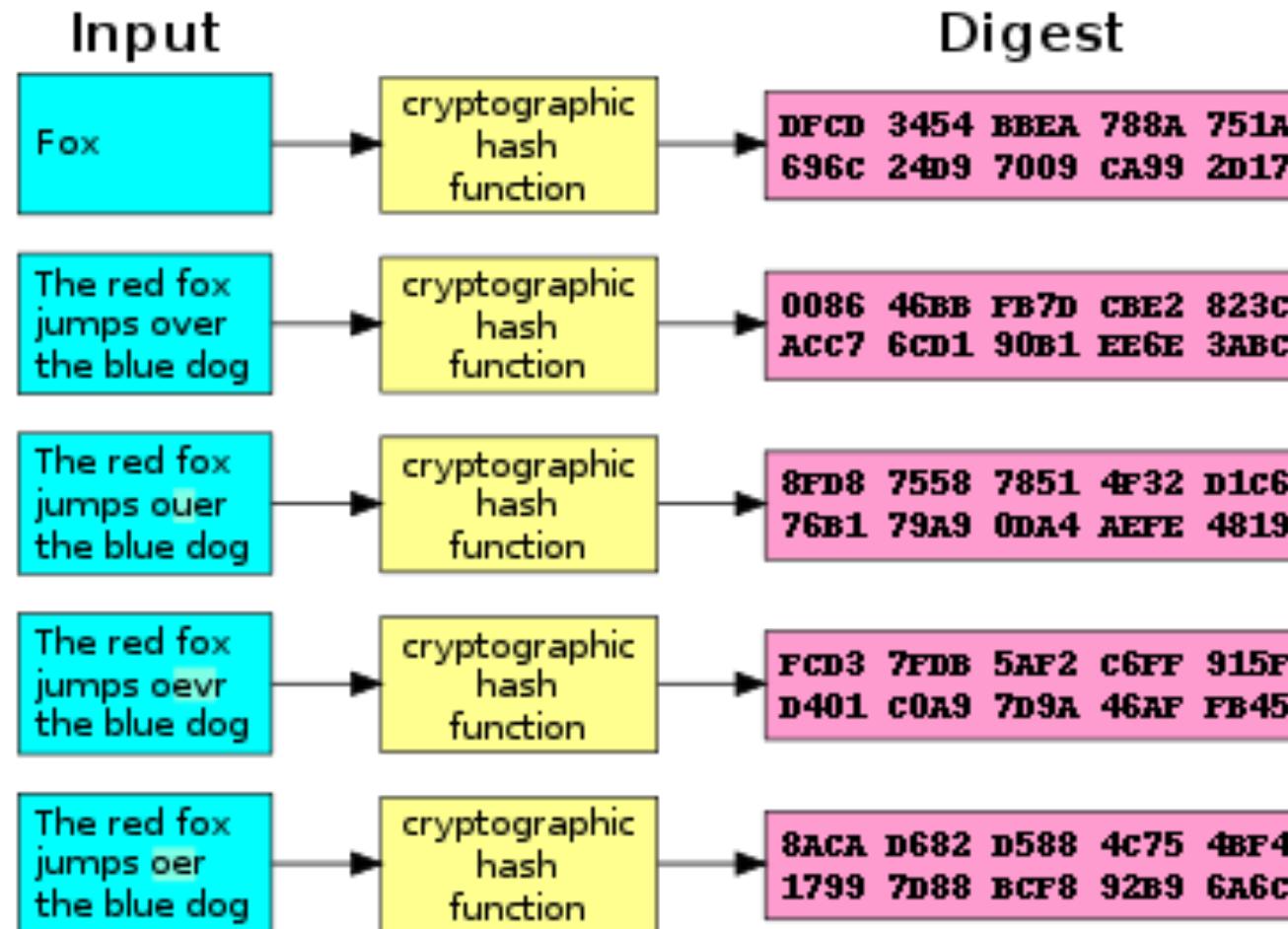
- acepta como entrada un bloque de información M (preimagen) de longitud variable
- produce un bloque de salida h (**valor hash**) de longitud fija
 - Ese valor se denomina **huella digital** de M



- 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**), y
 2. encontrar dos bloques M y M' que produzcan el mismo valor hash h (propiedad: **libre de colisiones**), tal que $h = h'$
- De lo anterior, se puede determinar que dos entradas M (ej. “*hola*”) y M' (ej. “*hora*”) a una función hash criptográfica, producen dos salidas h y h' completamente diferentes
 - De hecho, se asegura que un cambio en un único bit de la preimagen M da lugar a *un cambio de, como media, la mitad de los bits de salida tal que $h \neq h'$*



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



Ahora lo pruebas tú ...

- Accede a <https://cryptii.com> y prueba con varios textos de diferentes tamaños

The screenshot shows two separate instances of the cryptii.com interface side-by-side.

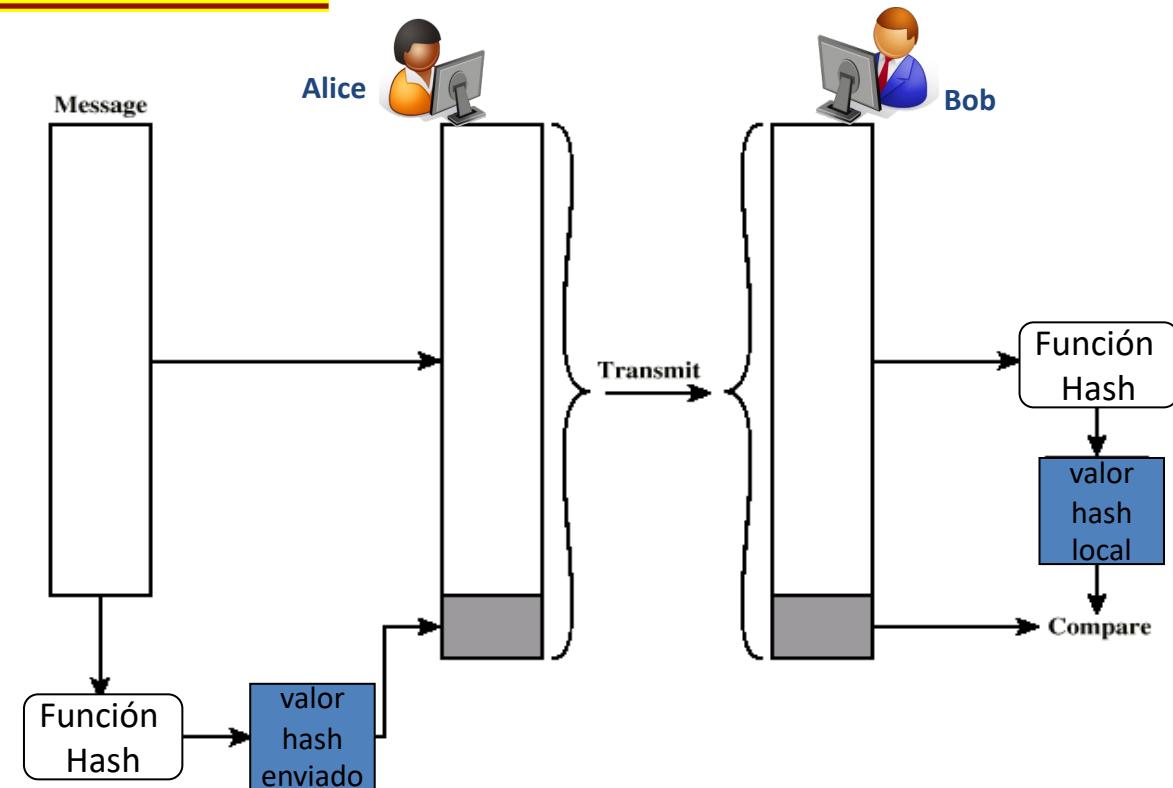
Top Instance:

- Text Input:** Hola, NO te toca a ti !
- Algorithm Selection:** SHA-256 (selected)
- Output:** Hexadecimal bytes: 1b 29 b9 aa ce 7b ac 2b f1 dc 77
7a 75 ce b5 f5 44 e3 bb 73 a7 b2
8c b4 bd 66 75 7d c1 9b 82 61

Bottom Instance:

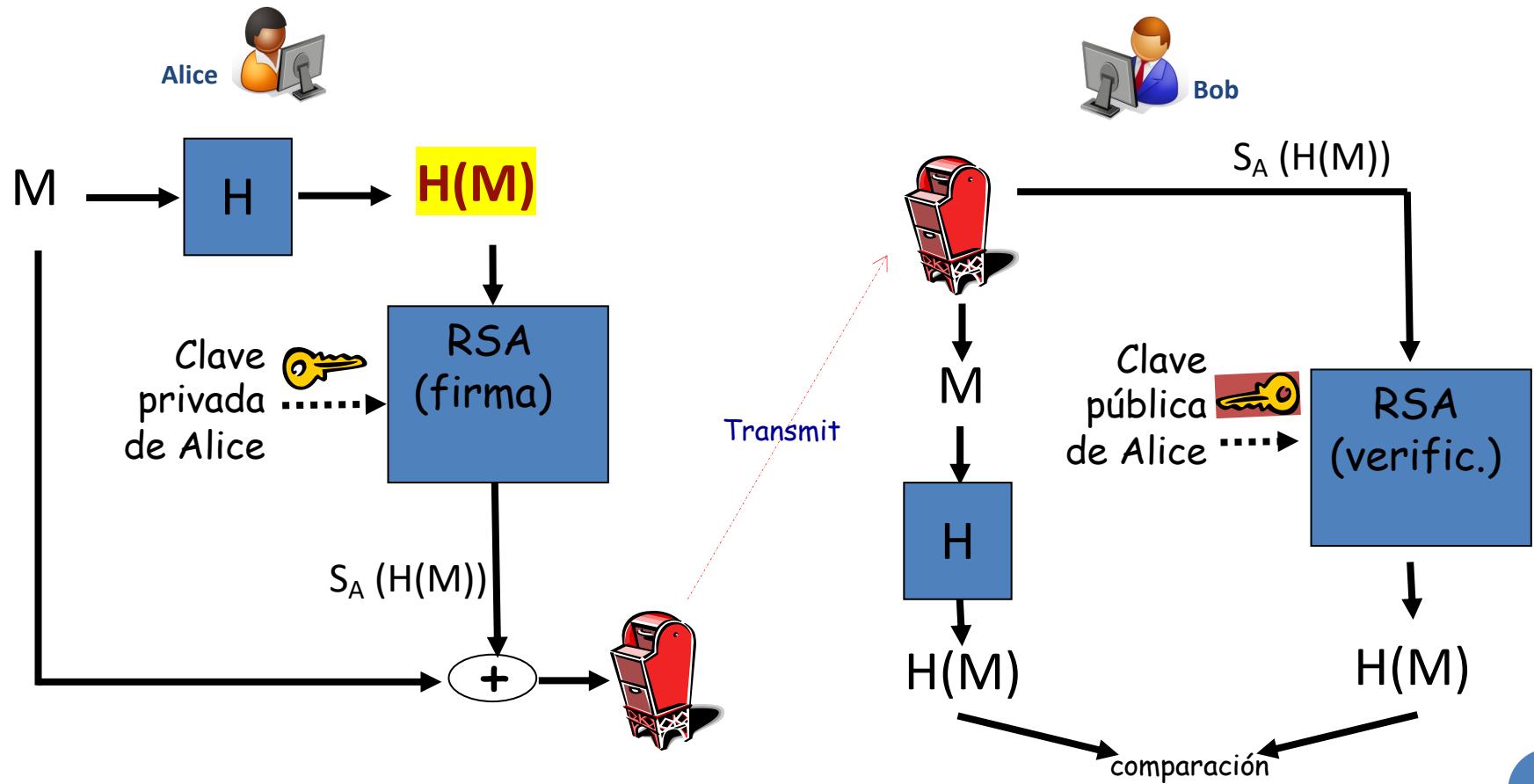
- Text Input:** Hola, te toca a ti !
- Algorithm Selection:** SHA-256 (selected)
- Output:** Hexadecimal bytes: 37 d0 ae bd eb 78 ab 7a 90 1b b2
9a ea aa b3 a5 1f b6 3c 33 bb 02
90 15 71 bd 8d 2b ce c0 48 f5

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

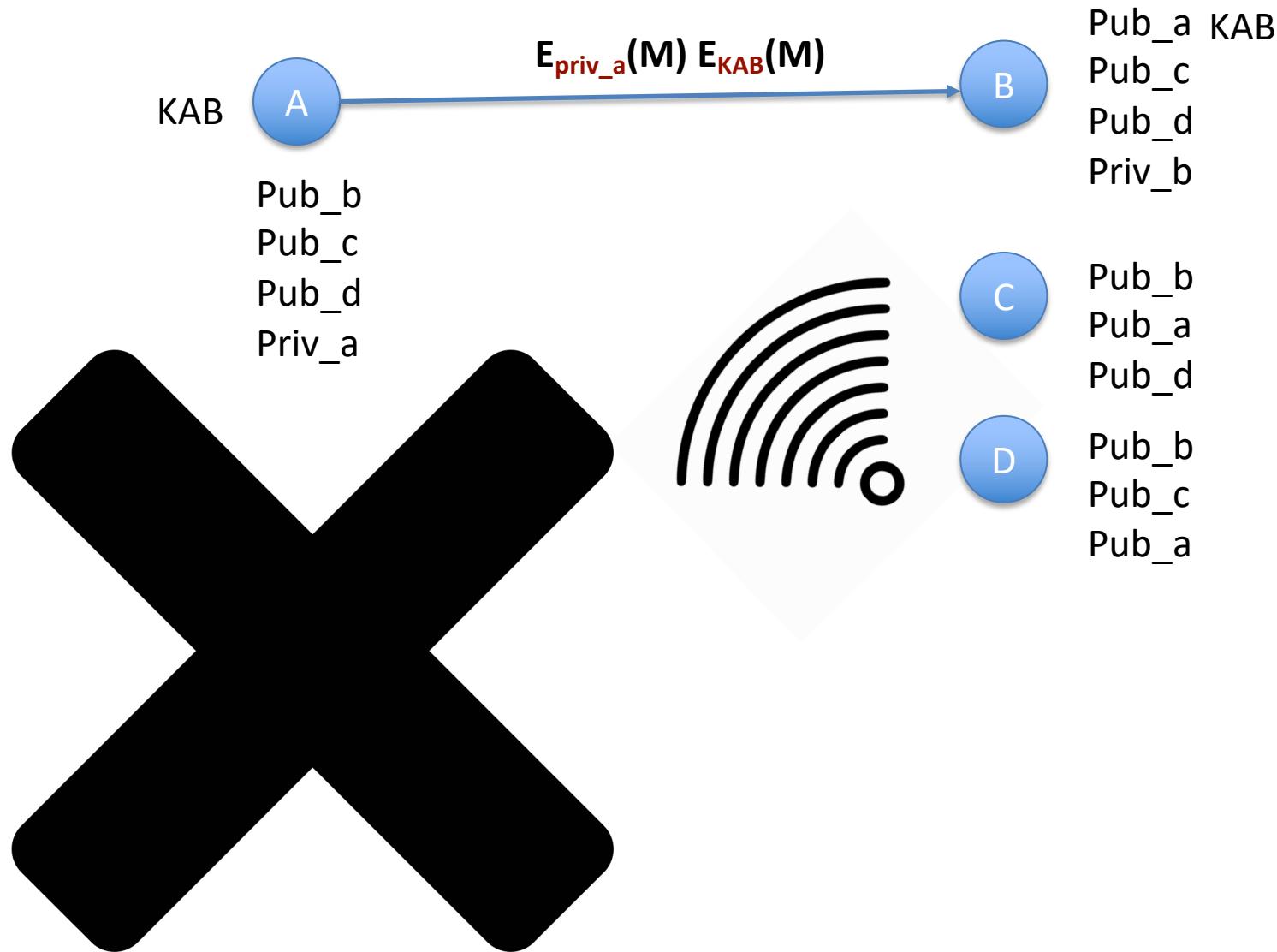


- Sin embargo, se observa que ***no hay autenticidad***
 - *Bob* no tiene garantías de que *Alice* sea el origen

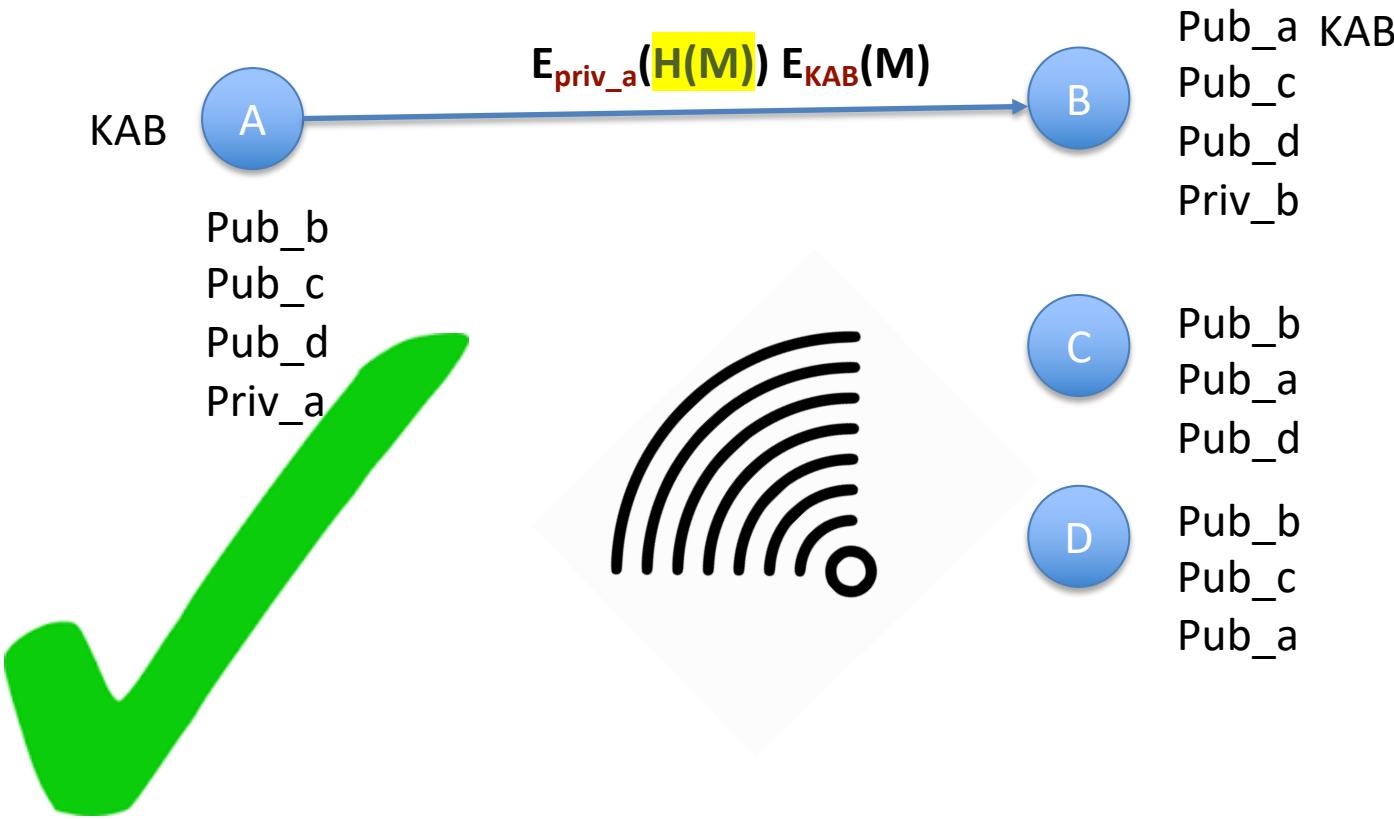
- El uso combinado de las funciones hash con la criptografía de clave pública mejora sustancialmente la eficiencia de uso de esta última (sobre todo en el procedimiento específico de firma digital)



Inciso



Inciso

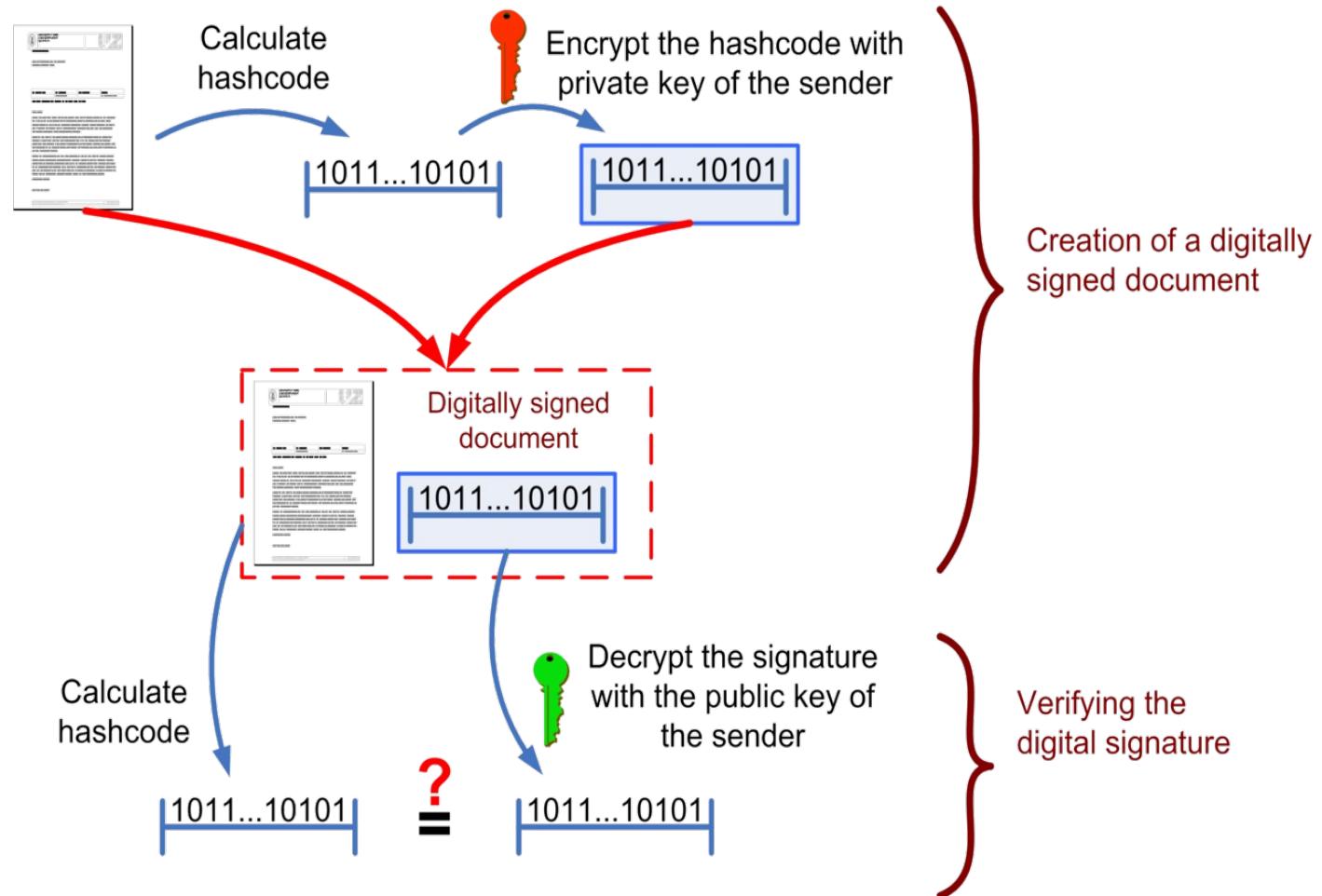


NO SE DEBE USAR C. ASIMÉTRICA PARA CIFRAR MENSAJES.
REALMENTE SE APLICA EN ESTOS CASOS:

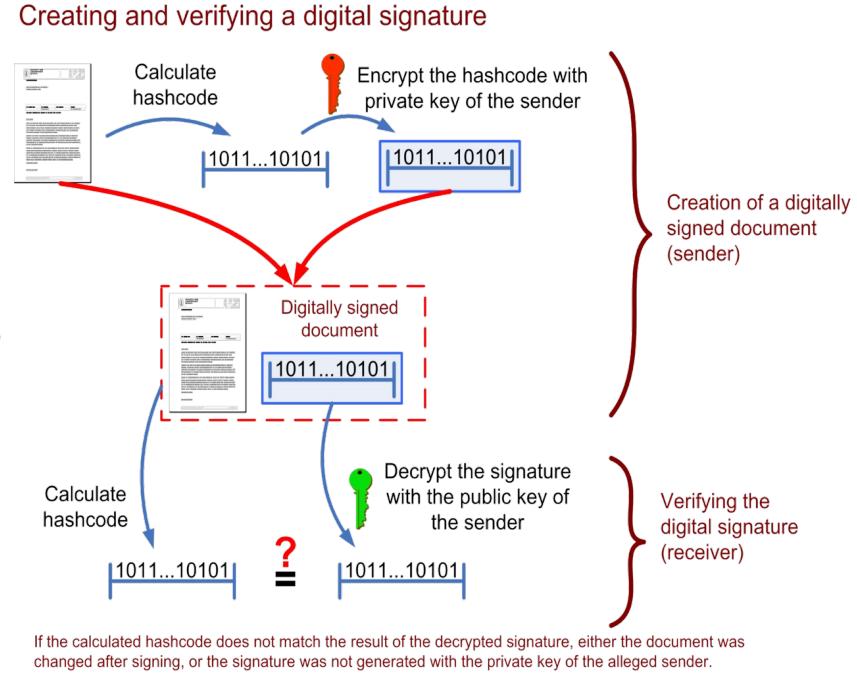
- CRIPT. HÍBRIDA → PARA PROTEGER LA CLAVE DE SESIÓN
- FIRMA DIGITAL → PERO APLICADO A EL HASH DEL MENSAJE

→ Debido al TAMAÑO
de los mensajes

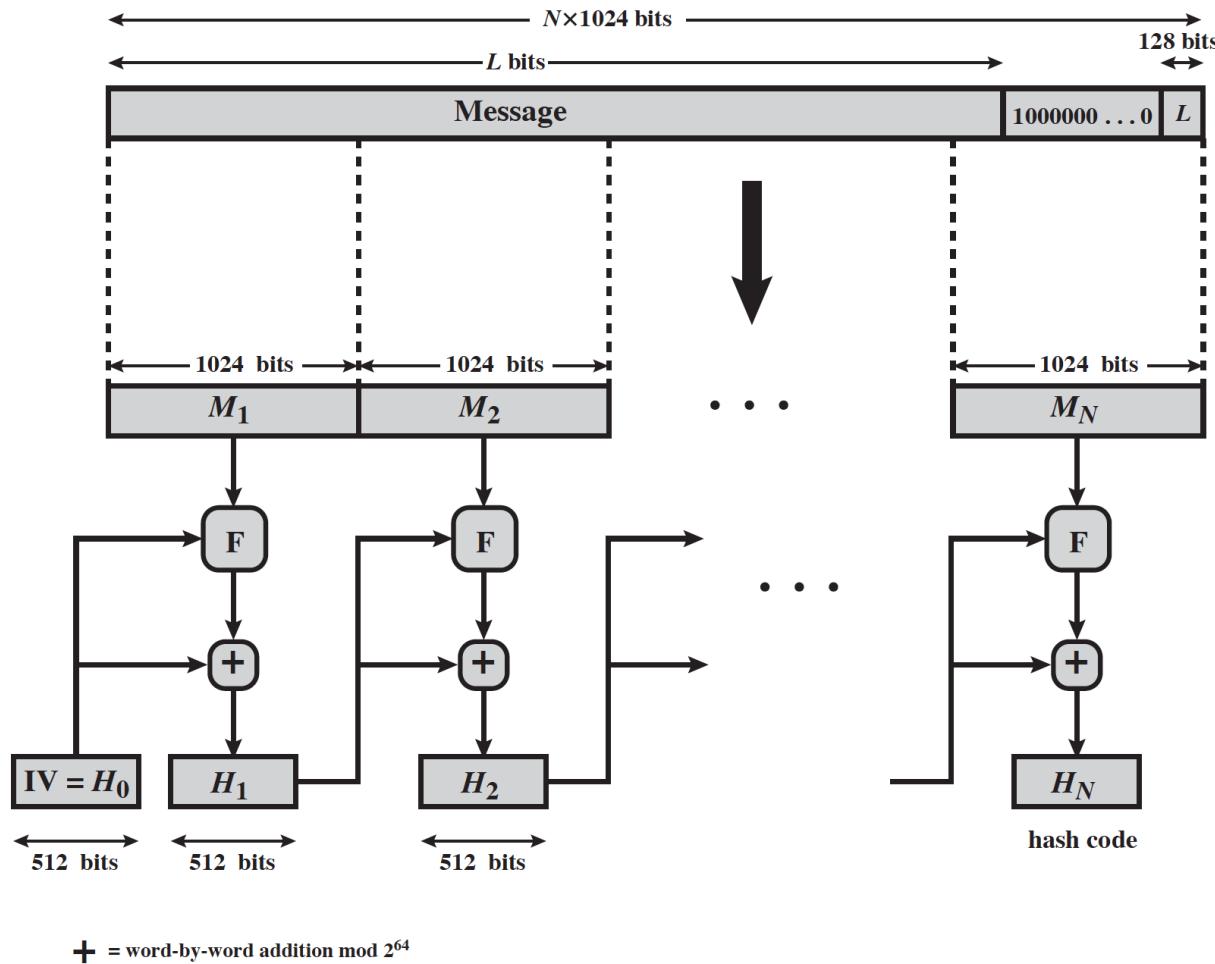
Creating and verifying a digital signature



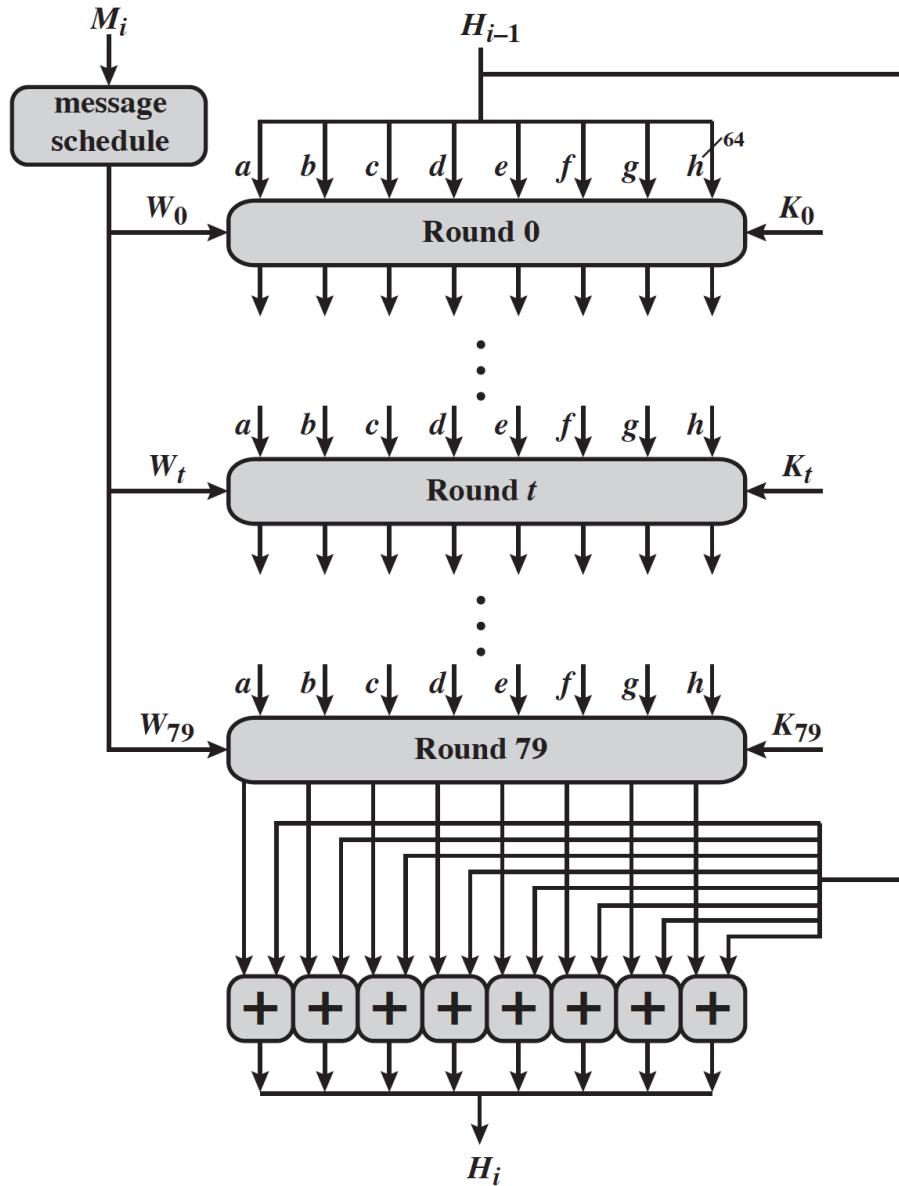
- Los beneficios del esquema:
 - la firma se puede mantener separada del documento
 - los requisitos de almacenamiento y firma son mucho menores
- Un sistema de archivos puede usar este tipo de esquema para verificar la existencia de documentos sin tener que almacenar sus contenidos



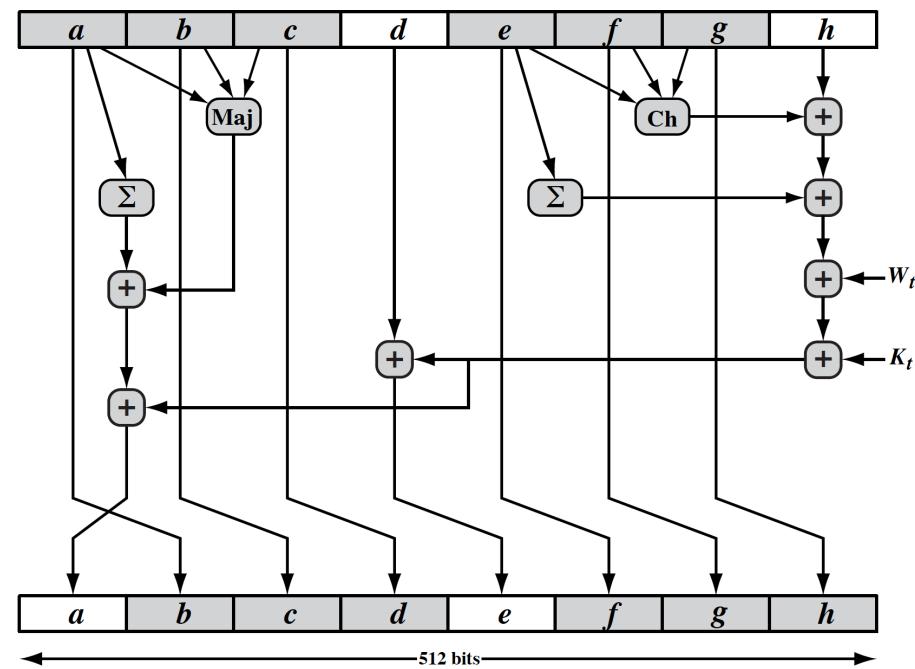
- Ejemplo de función hash: *SHA-512 (SHA-2)*



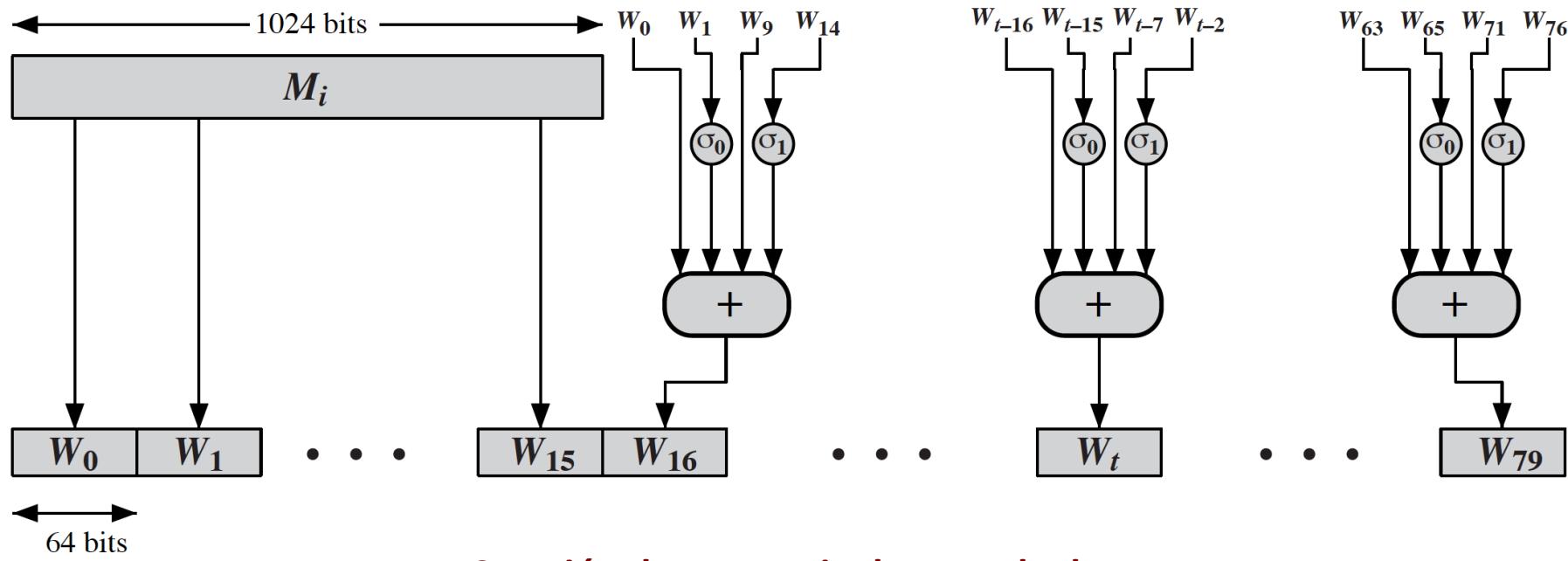
**Esquema
general del
algoritmo
SHA-512**



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	240calcc77ac9c65
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	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	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** (dado 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

theregister.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.

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

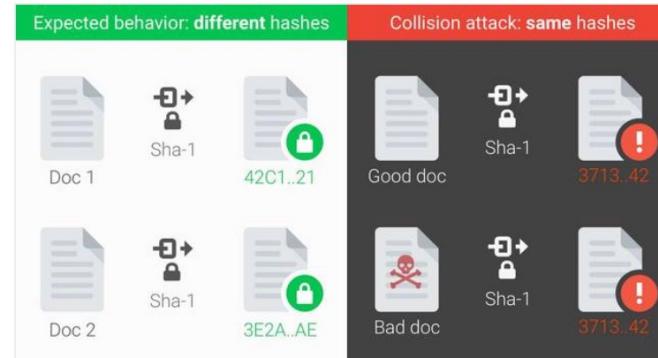


Google researchers and academics have today demonstrated it is possible – following years of number crunching – to produce two different documents that have the same SHA-1 hash signature.

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.

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.



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

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)

Complaints to [@steike](#).

<https://alf.nu/SHA1>

Hola a tod@s

Hola a tod@s



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

SHATTERED

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

Good doc	Sha-1	3713.42

Bad doc	Sha-1	3713.42

Attack proof

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

SHAttered
The first ever collision attack against SHA-1

CWI **Google**

SHAttered
The first ever collision attack against SHA-1

CWI **Google**

Drag some files here
Or, if you prefer...
Choose a file to upload

PDF 1 | PDF 2

<https://shattered.io>

SHATTERED

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

Good doc	Sha-1	3713.42

Bad doc	Sha-1	3713.42

Attack proof

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

SHAttered
The first ever collision attack against SHA-1

CWI **Google**

SHAttered
The first ever collision attack against SHA-1

CWI **Google**

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

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
Reset

File tester

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

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

- En la actualidad hay una familia de cuatro algoritmos hash:
 - *SHA-224, SHA-256, SHA-384* y *SHA-512*
- 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*

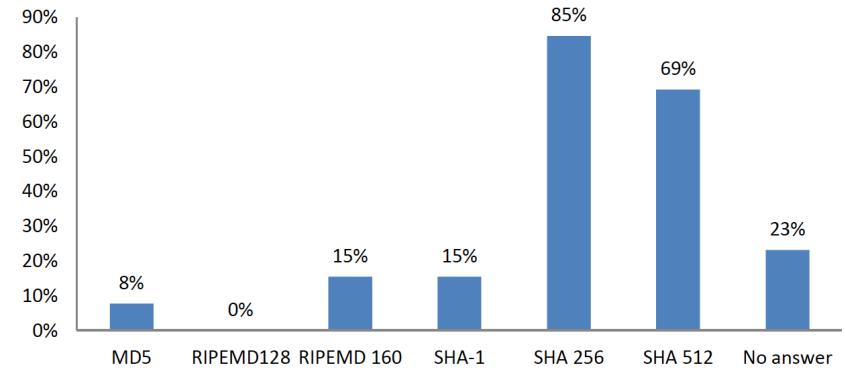
– Whirlpool

- 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:

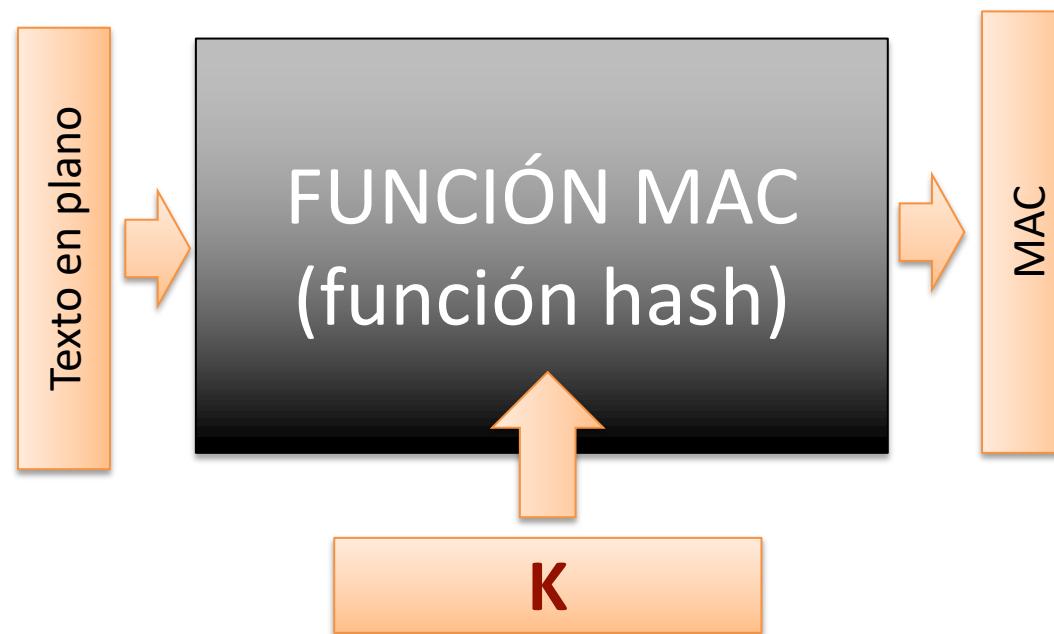
- El desarrollo de la funciones hash es, probablemente, el área de la Criptografía que ha atraído más atención durante la pasada década
- La mayoría de las funciones hash existentes derivan mucho de los criterios de diseño de la función hash *MD-4* (familia *MD-X*)
 - Esta familia incluye *MD-4*, *MD-5*, *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 Lenfth	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	✗	✗

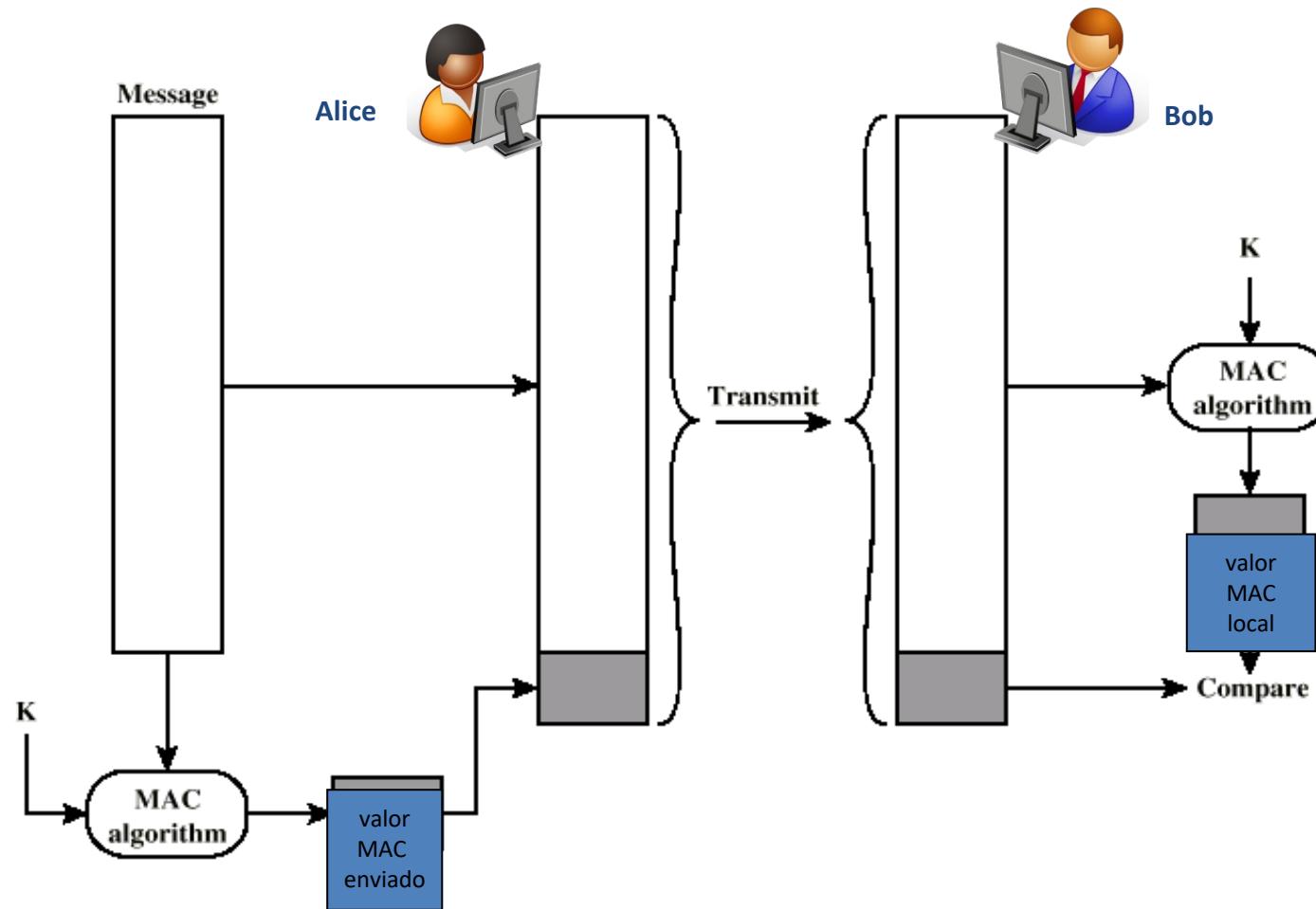


Funciones MAC

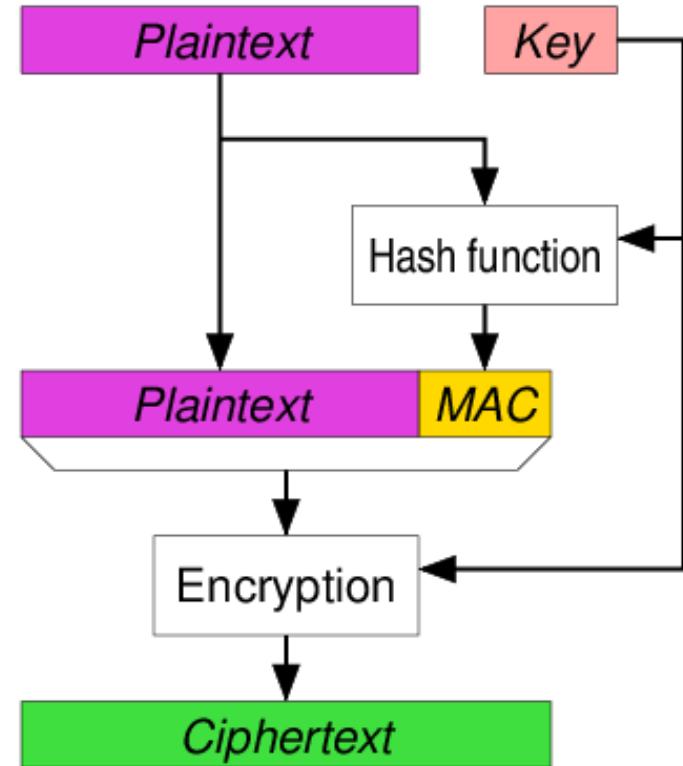
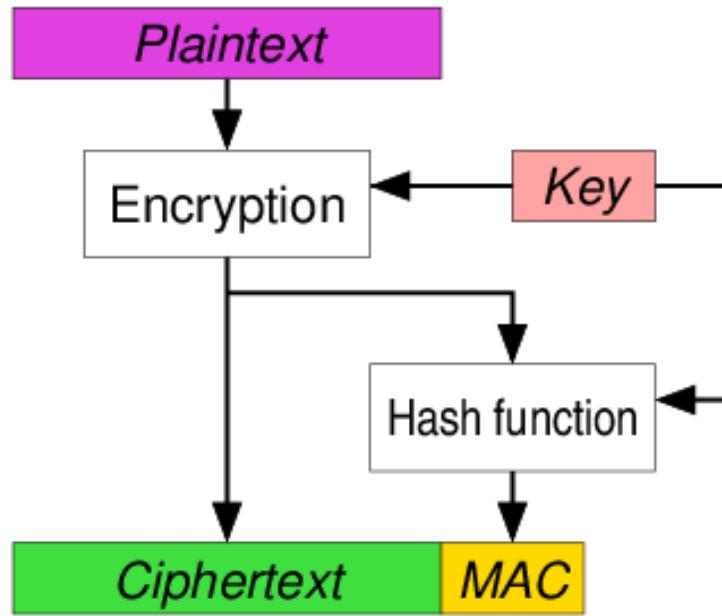
- Un código de autenticación de mensaje, o función MAC, es un concepto que evoluciona a partir del concepto de función hash
- Concretamente, una función MAC toma como entrada un mensaje M y una clave K , y produce un valor hash (que en este caso se denomina **valor MAC**)



- Ejemplo de uso:



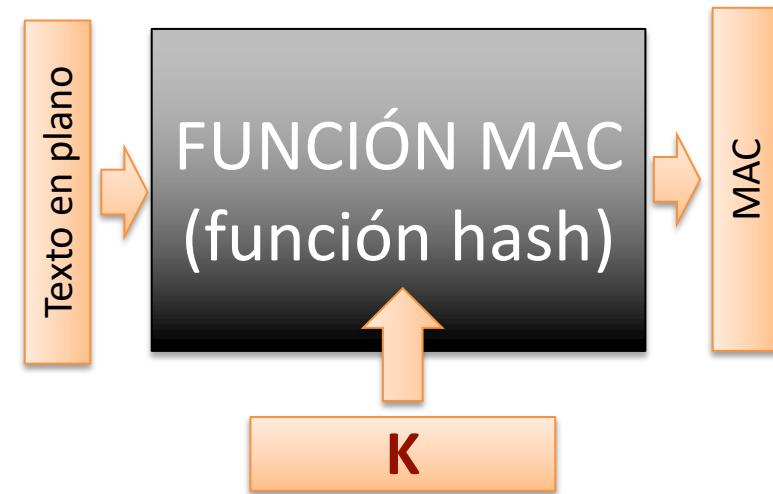
Dicho de otro modo...



- 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***
- Luego se garantiza:

AUTENTICACIÓN



- 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*)

Ahora lo pruebas tú ...

- Accede a <https://cryptii.com> y prueba con varios textos de diferentes tamaños

The screenshot shows three examples of using the HMAC feature on the Cryptii website:

- Example 1:** Input text: "Hola, te toca a ti !". HMAC interface: Key: 62 72 79 70 74 69 69. Algorithm: SHA-256. Bytes output: 0b a4 1c 9a b6 c9 05 bf 34 ff ea 3a a8 06 62 a2 6c 27 90 e7 19 18 74 b2 ad 97 88 0e a1 6e 14 0e.
- Example 2:** Input text: "Hola, te toca a ti !". HMAC interface: Key: 63 72 79 70 74 69 69. Algorithm: SHA-256. Bytes output: 39 73 b7 da 85 62 a8 54 07 e4 3b fe 0f b7 4f fd 53 13 00 fd a2 b7 7e db 62 d3 c0 ef 79 b8 95 8e.
- Example 3:** Input text: "Hola, NO te toca a ti !". HMAC interface: Key: 62 72 79 70 74 69 69. Algorithm: SHA-256. Bytes output: 10 3a e2 e7 2d 2c 5b c6 cd 0d fe 34 77 67 f8 97 16 72 00 4e 73 96 d8 4c b7 cf b9 07 0b ee 32 96. Note: Encoded 32 bytes.

- 1) Cambia el texto
- 2) Cambia el valor de la clave

Inciso - recordatorio

• Galois-CTR (GCM)

- Funciona de forma similar que el modo CRT pero usa Carter-Wegman MAC en un campo de Galois
- Es rápido y eficiente, y está soportado por el suite de cifrado dado por TLS 1.3

