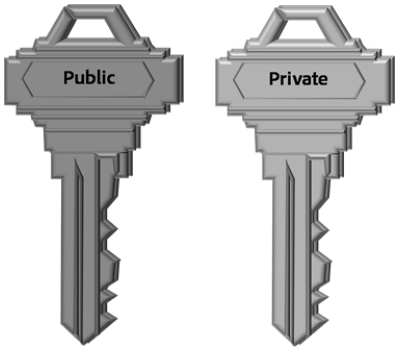
SEGURIDAD DE LA INFORMACIÓN

TEMA 2

TÉCNICAS CRIPTOGRÁFICAS BÁSICAS

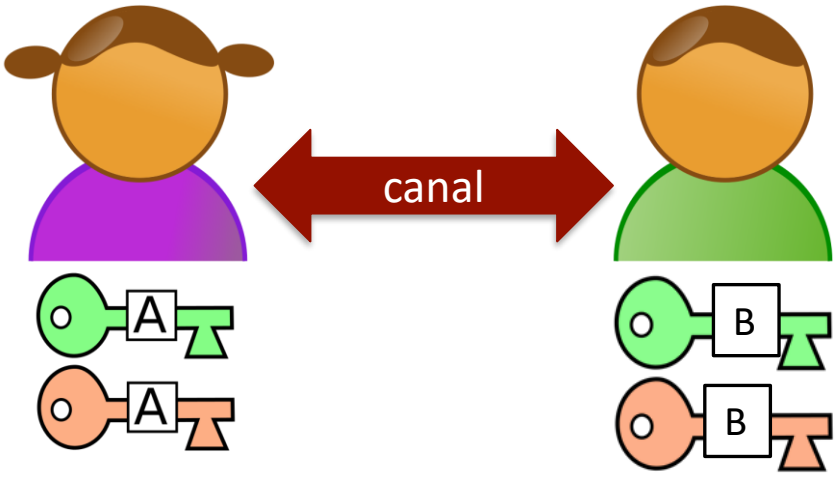
(Y SERVICIOS DE SEGURIDAD ASOCIADOS)

Algoritmos asimétricos (o de clave pública)



Introducción

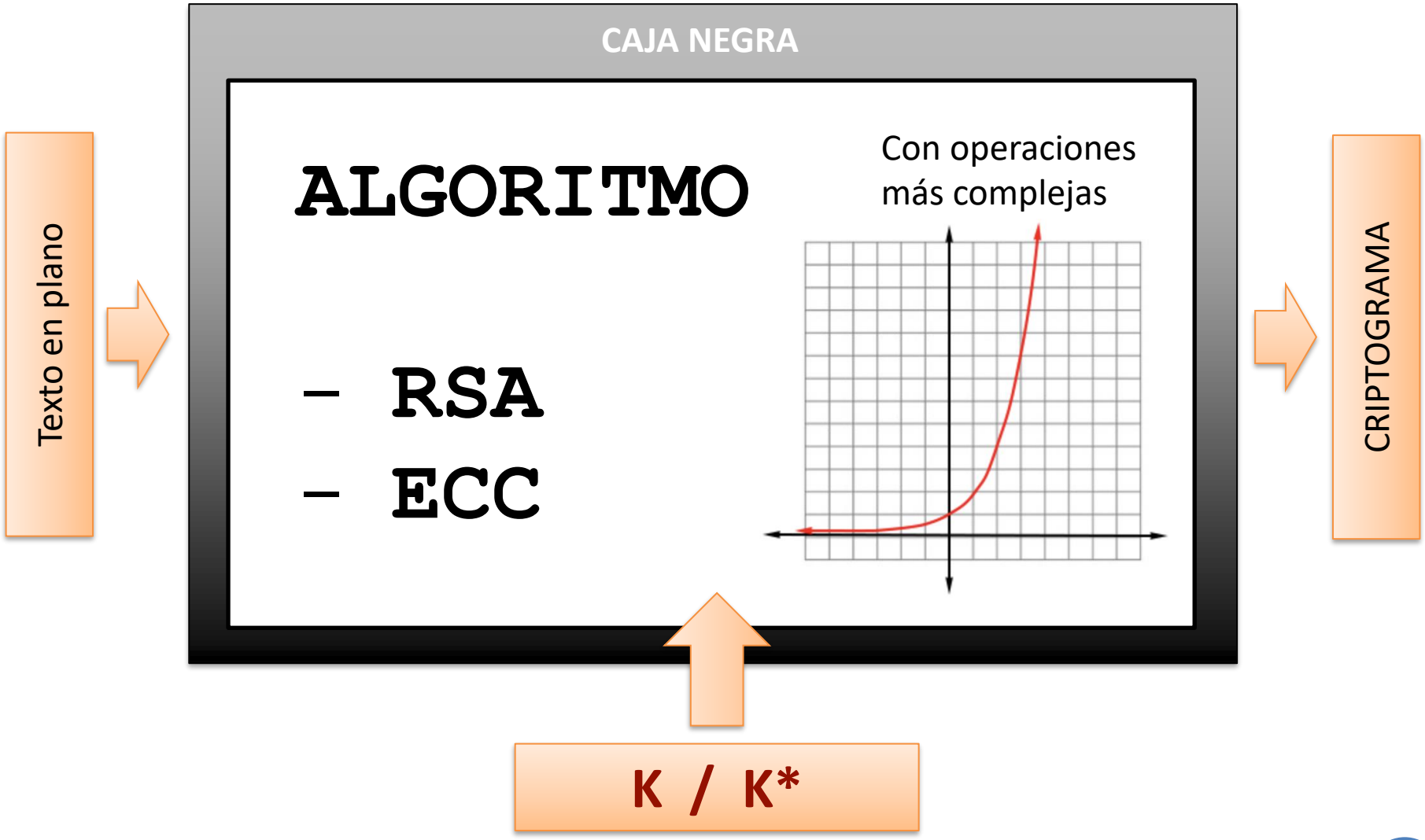
- El concepto de criptografía de clave pública (o asimétrica) fue inventado en 1976 por *Diffie* y *Hellman*, e independientemente por *Merkle*, para dar solución a algunos de los problemas de los criptosistemas simétricos



- 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

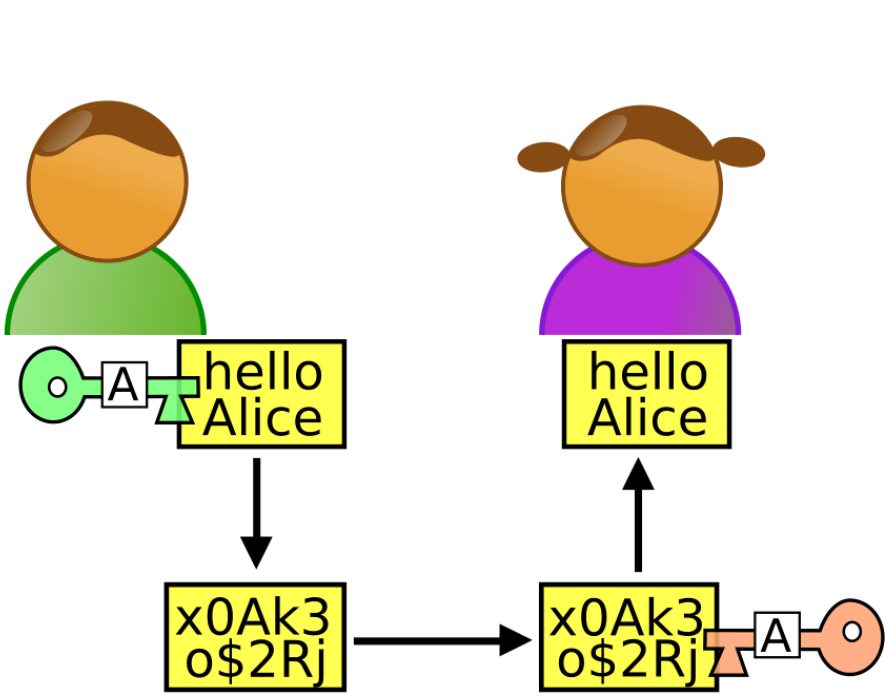
- **K / K***:
  - Las claves se utilizan por pares, de tal forma que cada usuario *U* posee dos claves:
    - Una **clave pública**, conocida por todos los usuarios
    - Una **clave privada**, conocida sólo por *U*

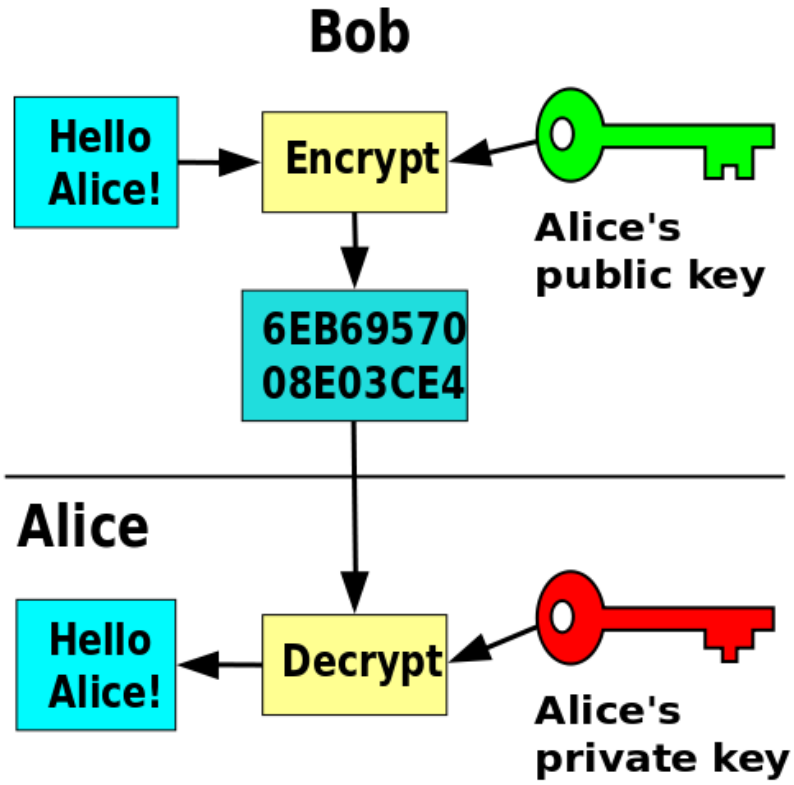
- Operaciones con K/K*:
  - Una clave se usa para cifrar, y la otra para descifrar
  - Es decir, si se cifra un mensaje con una de las claves, esa misma no servirá para descifrar, sino que necesariamente habrá que usar la otra
- Existen tres funcionalidades básicas con criptografía asimétrica

CIFRADO | **FIRMA DIGITAL** | **INTERCAMBIO DE CLAVES**

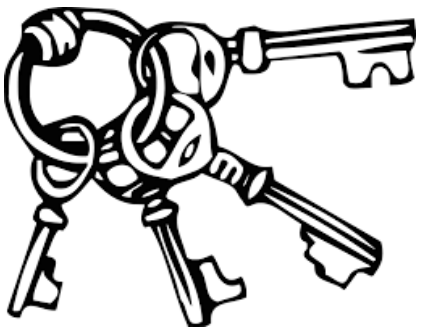
Cifrado/descifrado

- 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:

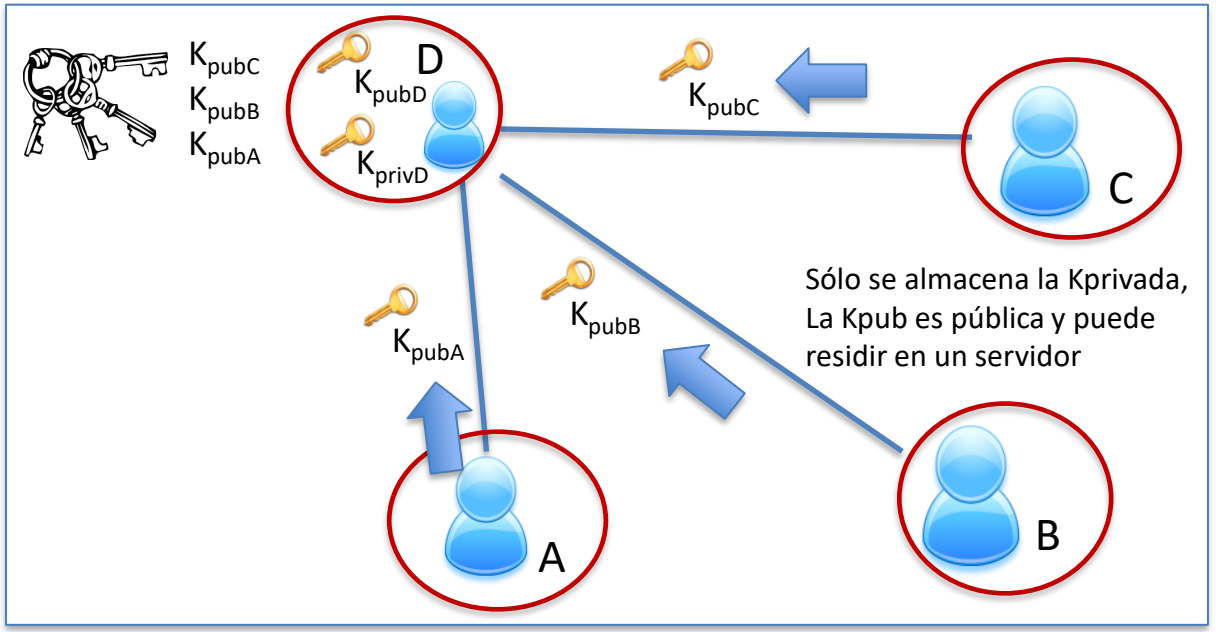




- Del esquema anterior se desprende que *Alice* y *Bob* no necesitan acordar a priori ninguna clave (a diferencia de los algoritmos simétricos)
- Pero *Bob* ha de conocer la clave pública de *Alice*
  - Y también la clave pública de cada uno de los usuarios con los que desee contactar
- Para ello, existen 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

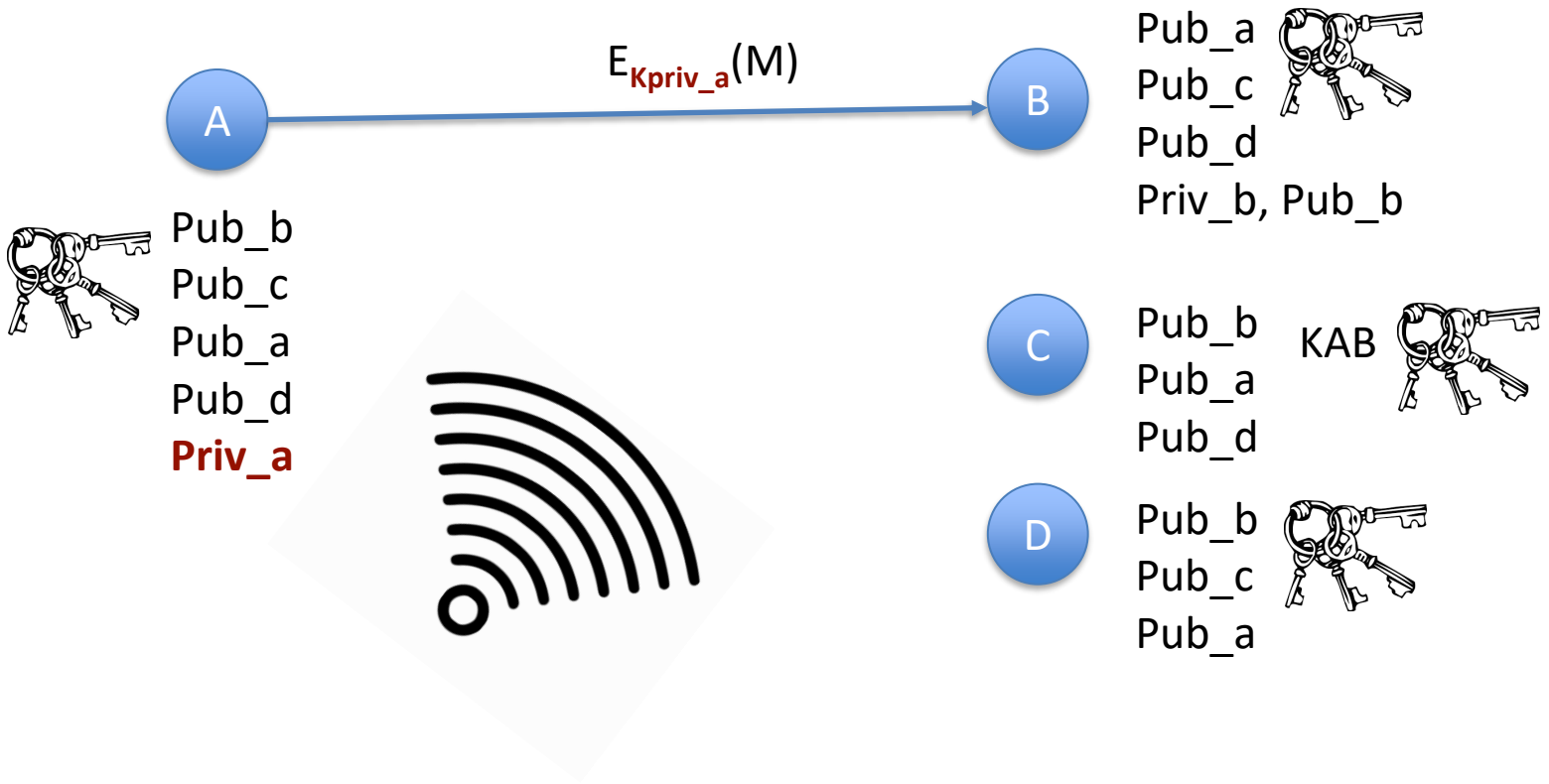


- Otras características relevantes que deberías saber:
  - Es **computacionalmente imposible deducir la clave privada** del usuario *U* a partir de su clave pública
  - Cualquier usuario con la **clave pública de *U* puede cifrar** un mensaje hacia *U*, pero no descifrarlo
    - Cifrar el mensaje con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

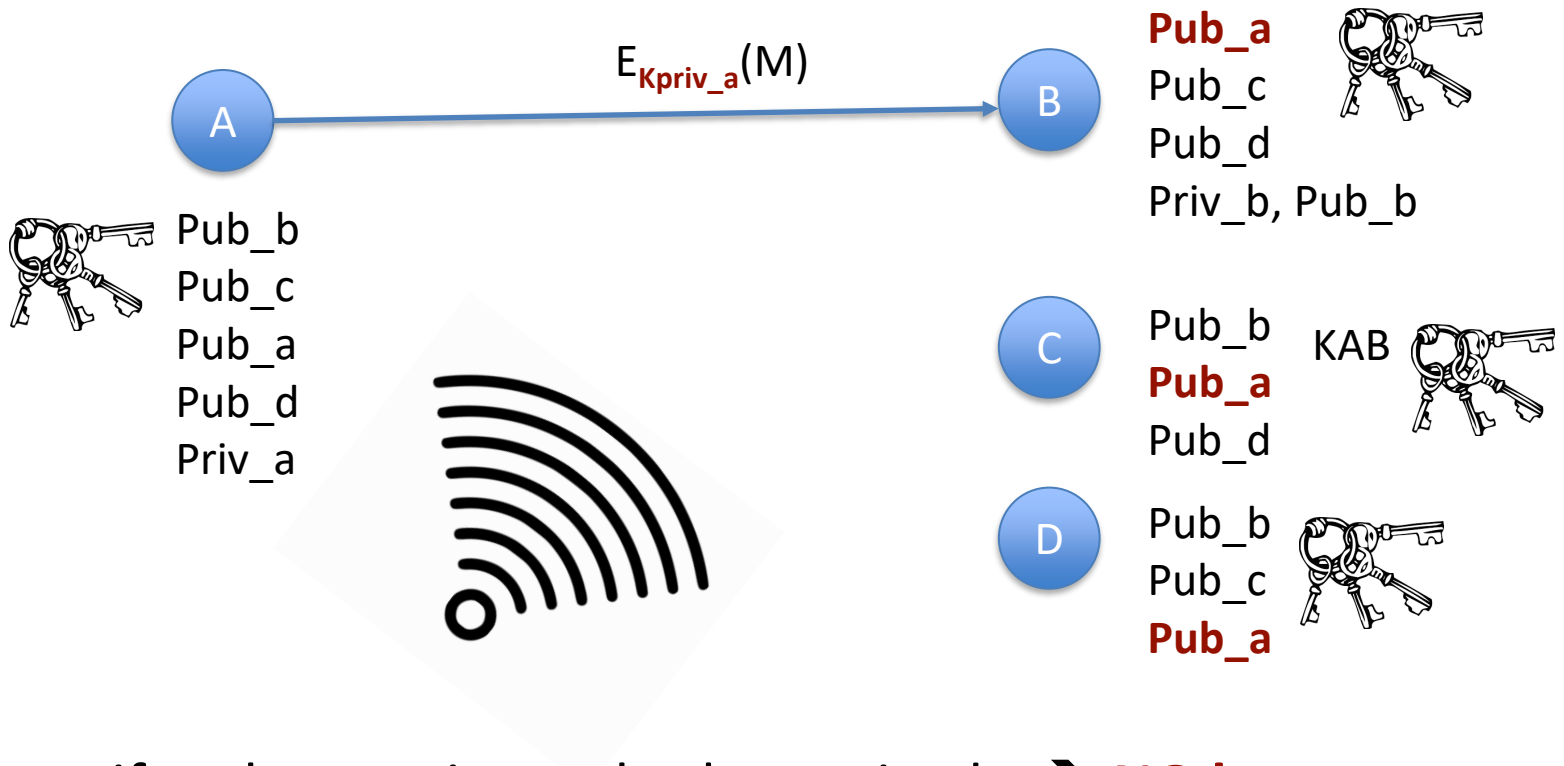
Algunos casos de ejemplos – Queremos confidencialidad

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



Algunos casos de ejemplos – Queremos confidencialidad

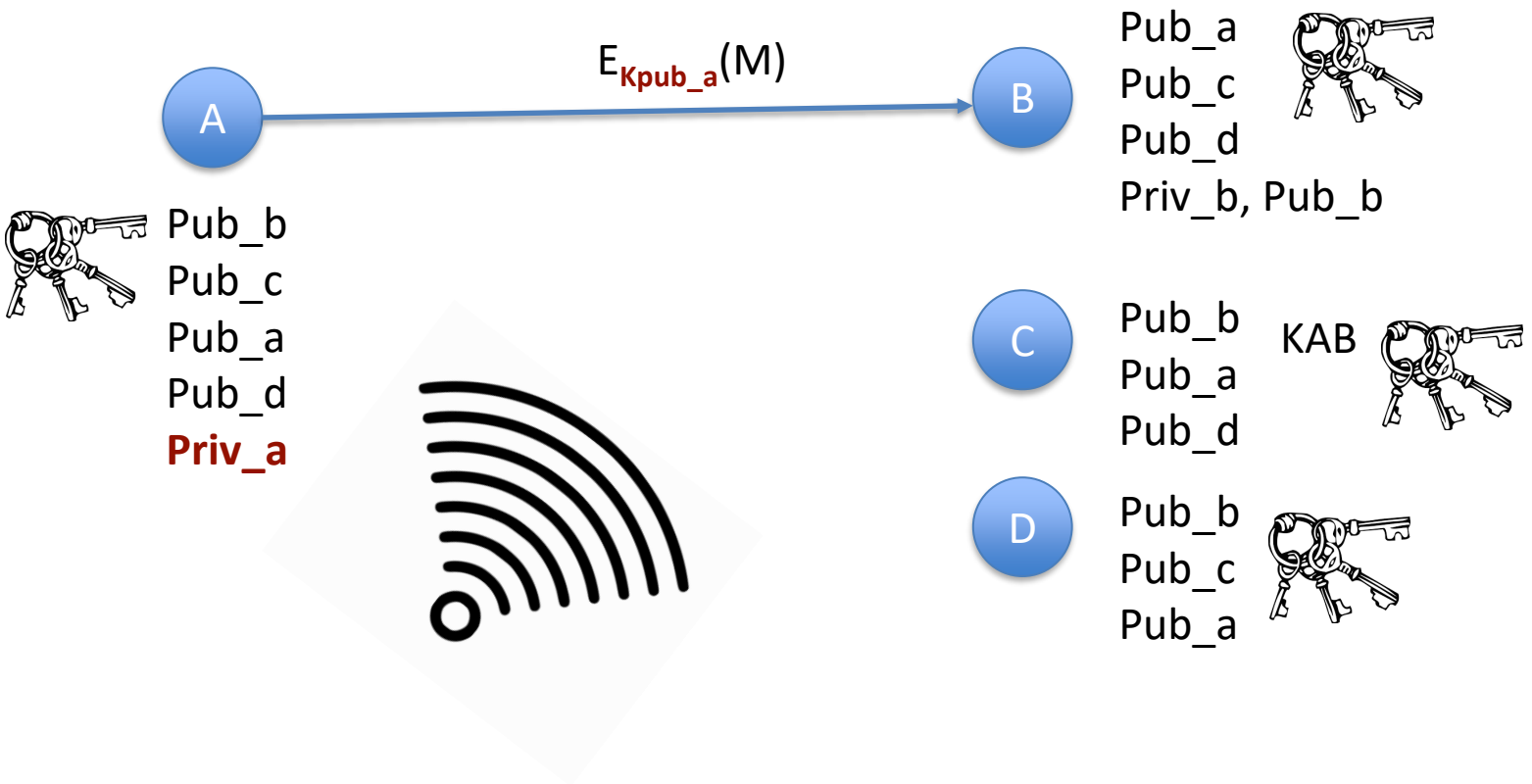
CASO A: A envía a B un mensaje con su clave privada - **¡¡ NO !!**



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

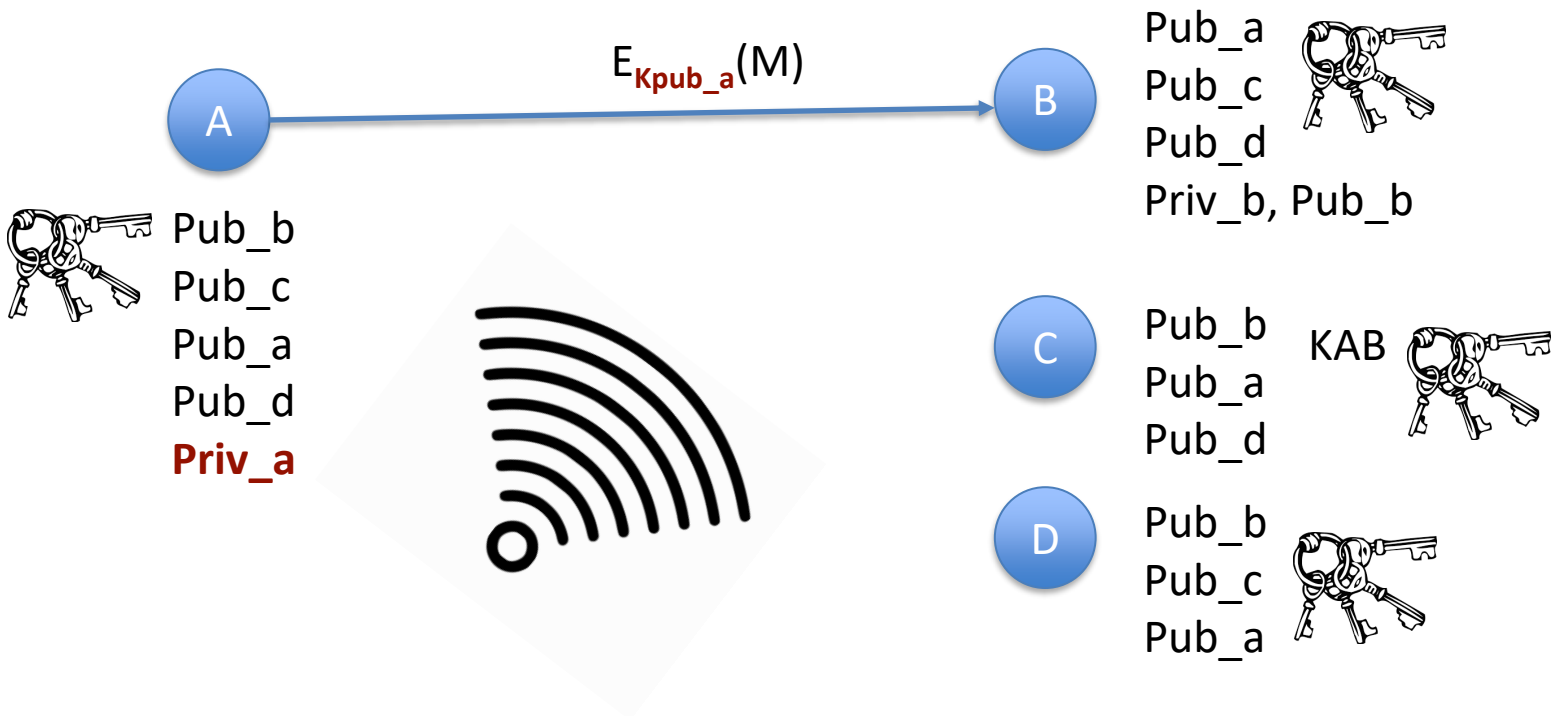
Algunos casos de ejemplos – Queremos confidencialidad

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



Algunos casos de ejemplos – Queremos confidencialidad

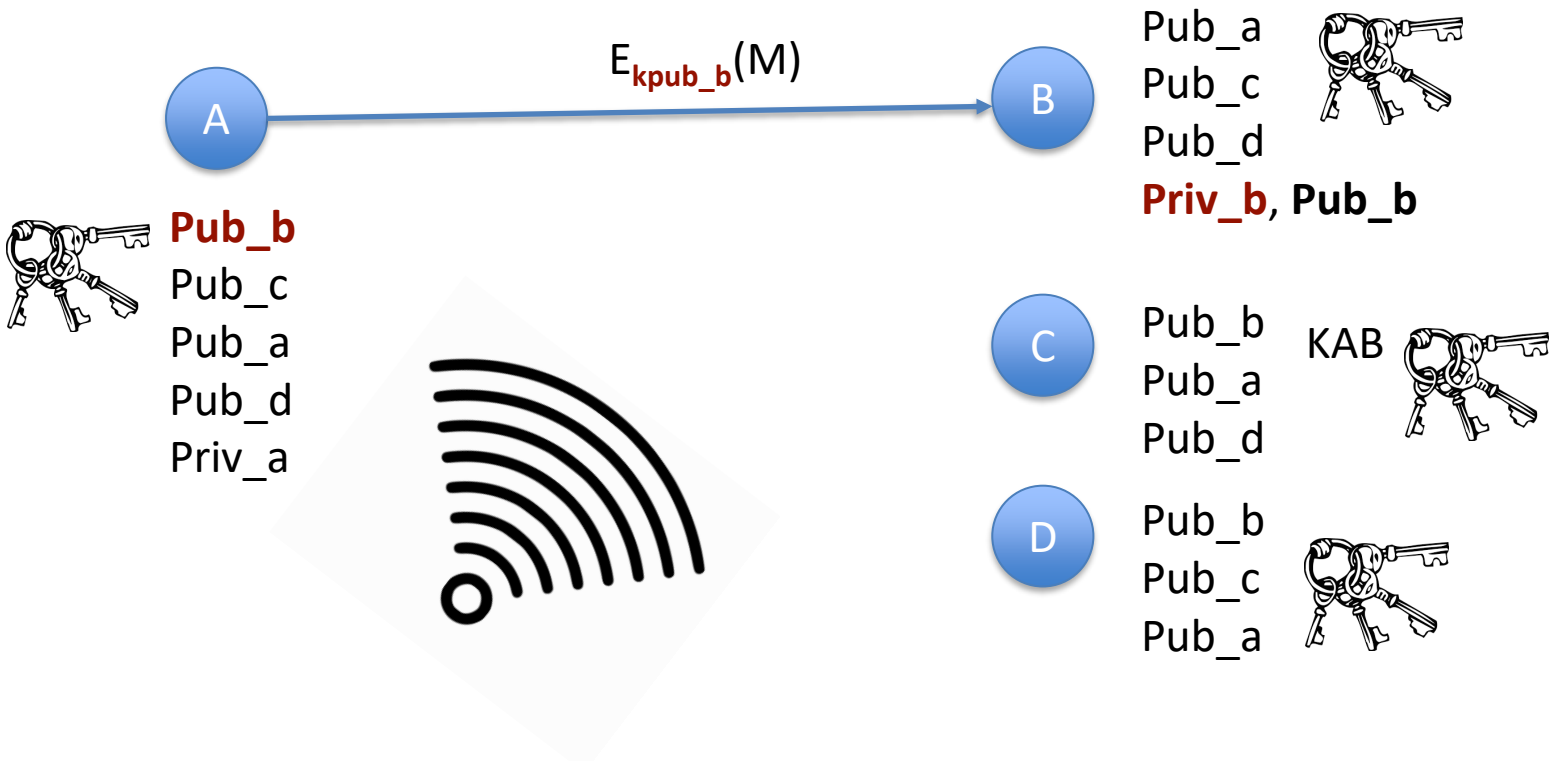
CASO B: A envía a B un mensaje con su clave pública - **¡¡ NO !!**



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

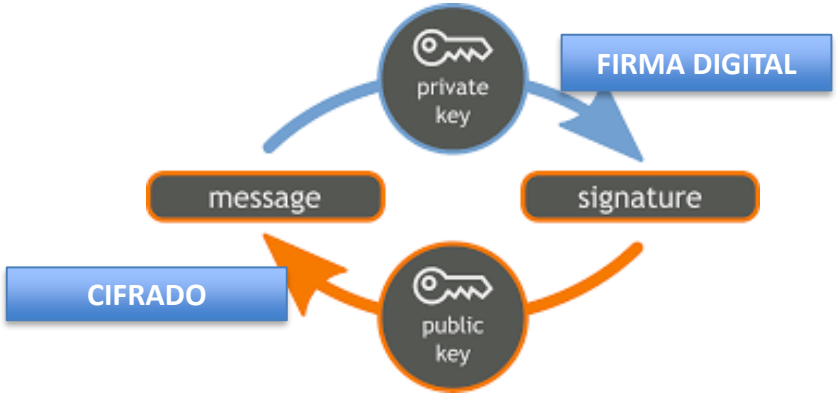
Algunos casos de ejemplos – Queremos confidencialidad

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



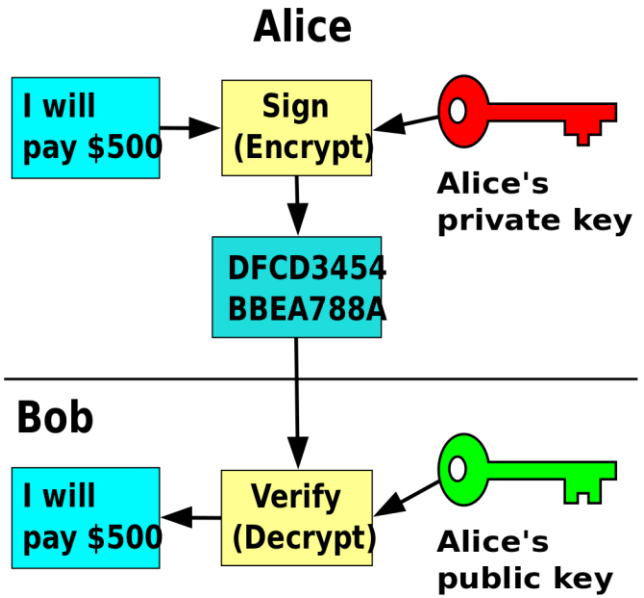
Si se cifra el mensaje con la clave pública de B ➔ **HAY confidencialidad** porque sólo B tiene la Kpriv_b asociada a Kpub_b, y C y D no podrán leer el mensaje

- 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



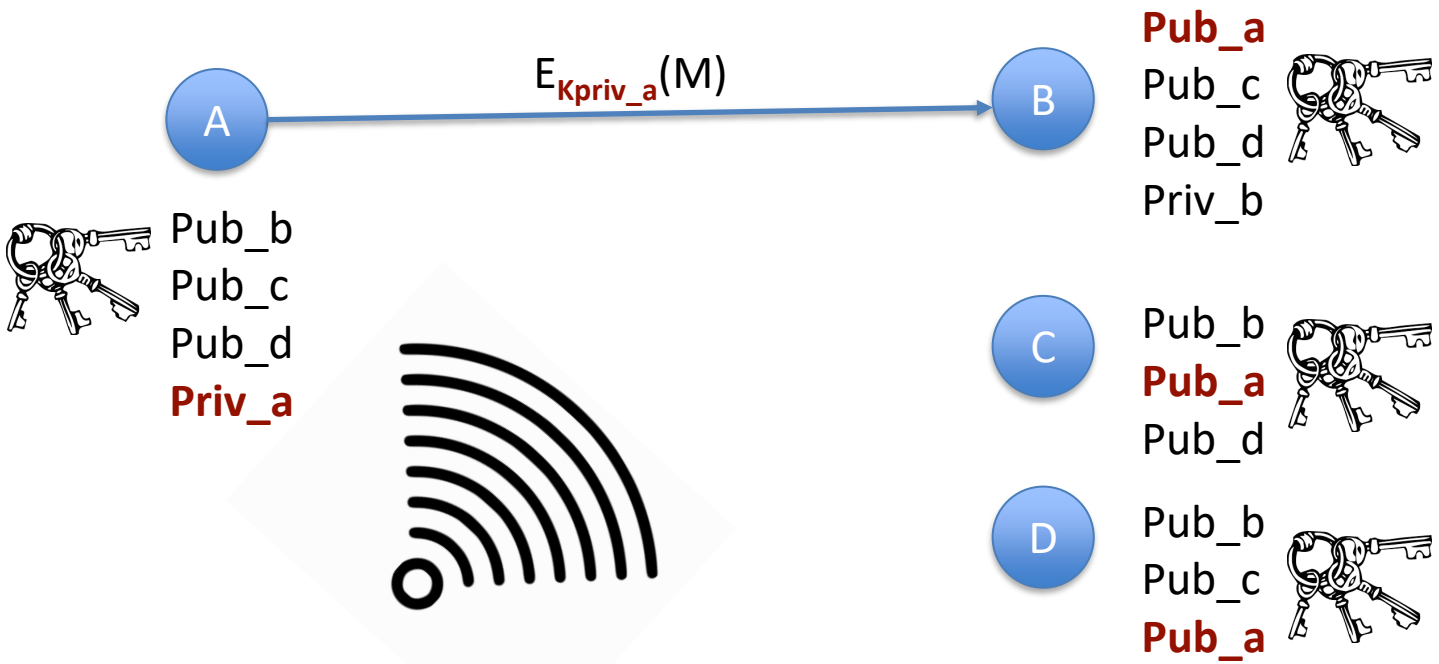
Firma digital

- *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)



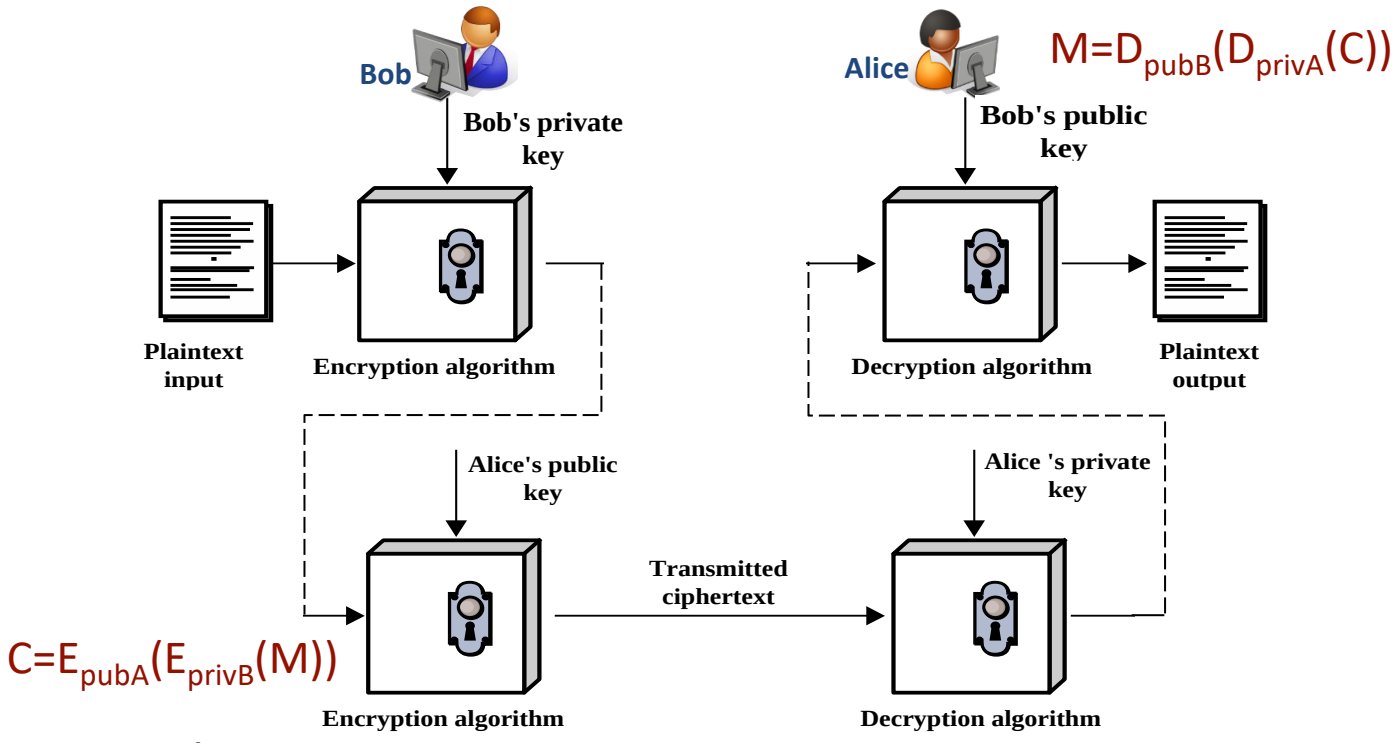
Algunos casos de ejemplos – Queremos autenticación

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
```

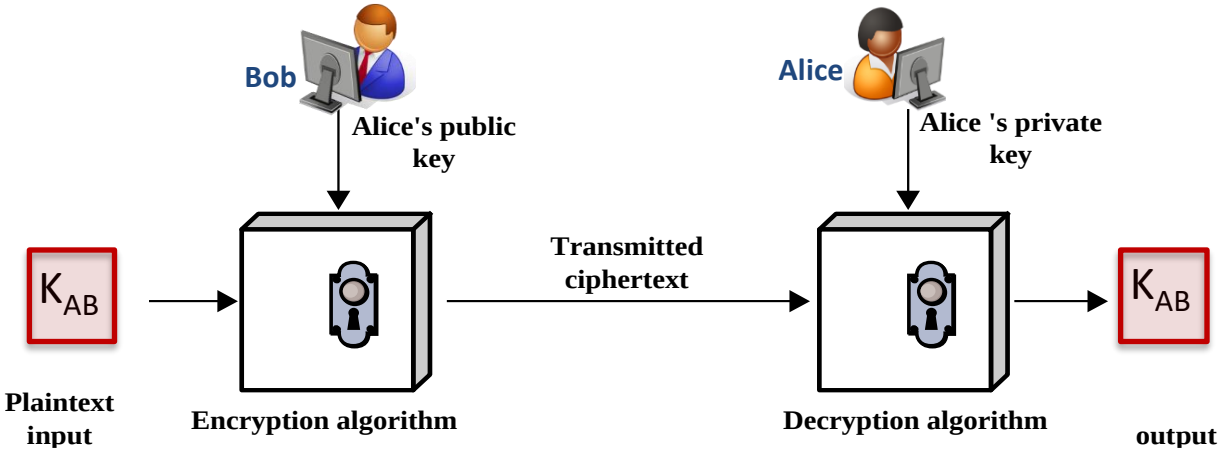
```
$ openssl speed aes
To get the most accurate results, try to run this
program when this computer is idle.
Doing aes-128 cbc for 3s on 16 size blocks: 30108378 aes-128 cbc's in 2.97s
Doing aes-128 cbc for 3s on 64 size blocks: 7712443 aes-128 cbc's in 2.96s
Doing aes-128 cbc for 3s on 256 size blocks: 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



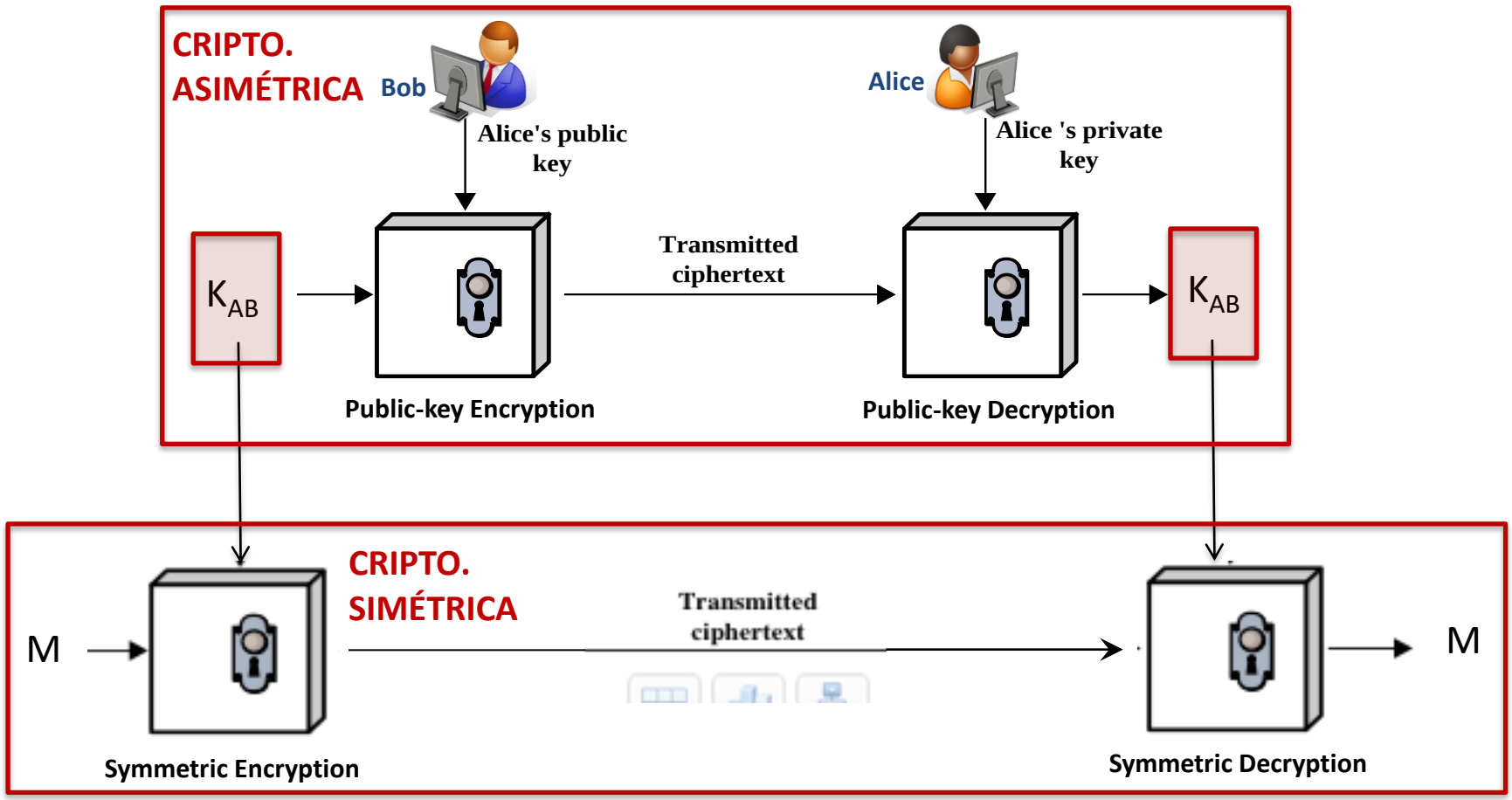
- El resultado final es un **criptosistema híbrido**
  - Uso de **un criptosistema de clave pública + un criptosistema simétrico**

Intercambio de claves → Criptografía híbrida

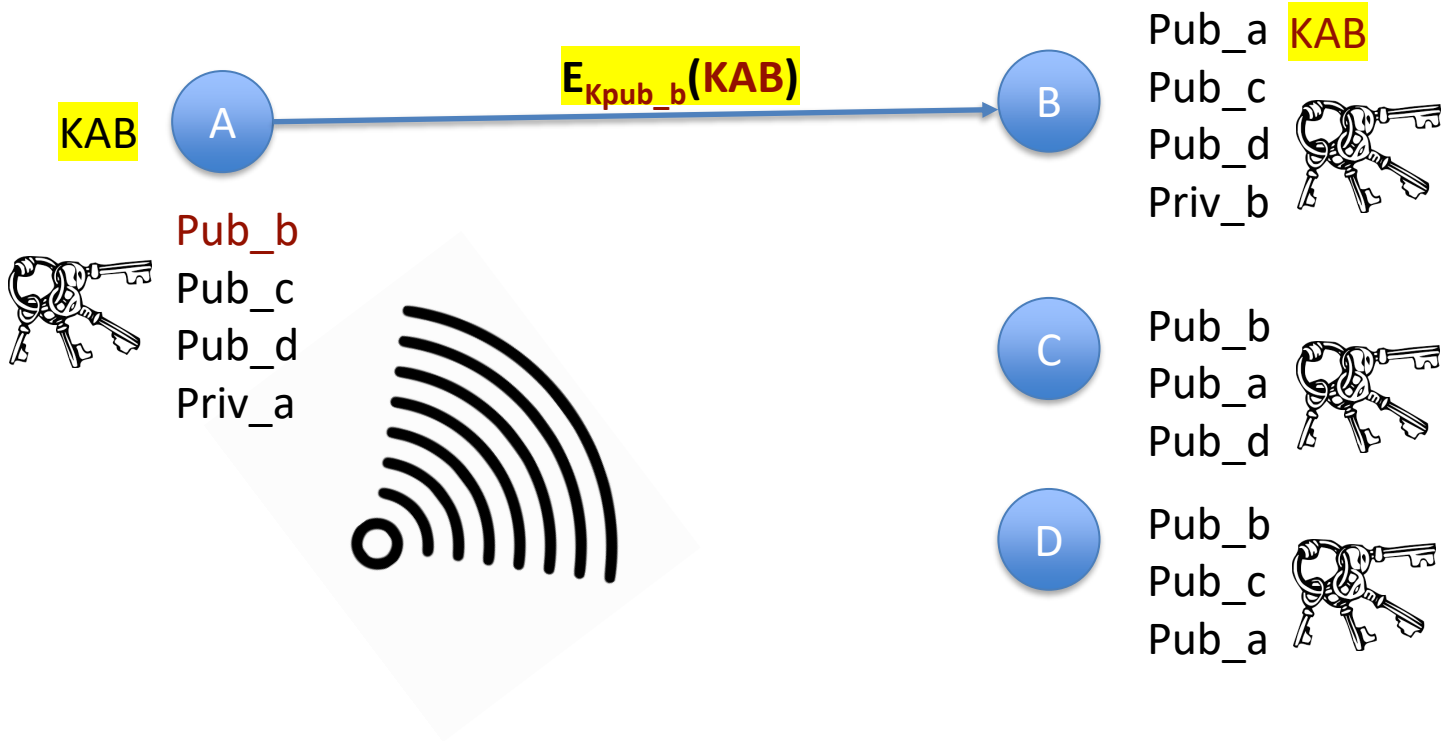
C. Simétrica	C. Asimétrica
R+: • los algoritmos son más simples y demandan menos recursos	S+: • las claves son de tamaño mayor, y la Kpriv no se puede (computalmente y matemáticamente hablando) derivar de la Kpub • Los algoritmos son más robustos frente ataques y permite enviar datos seguros por cualquier medio
S-: • las claves son pequeñas y se pueden derivar por fuerza bruta • Las claves son inseguras y requieren frecuentes procesos de rekeying	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**

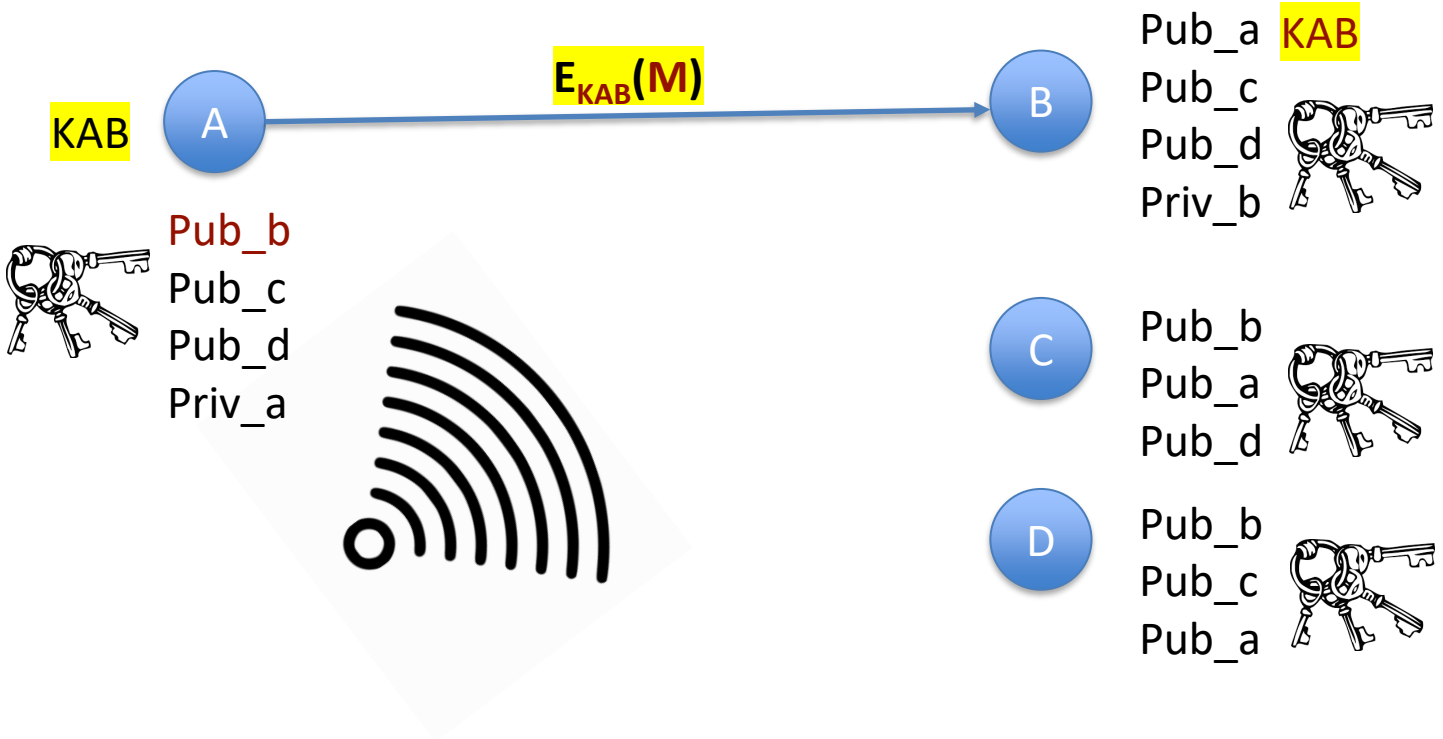


Algunos casos de ejemplos – Queremos criptografía híbrida



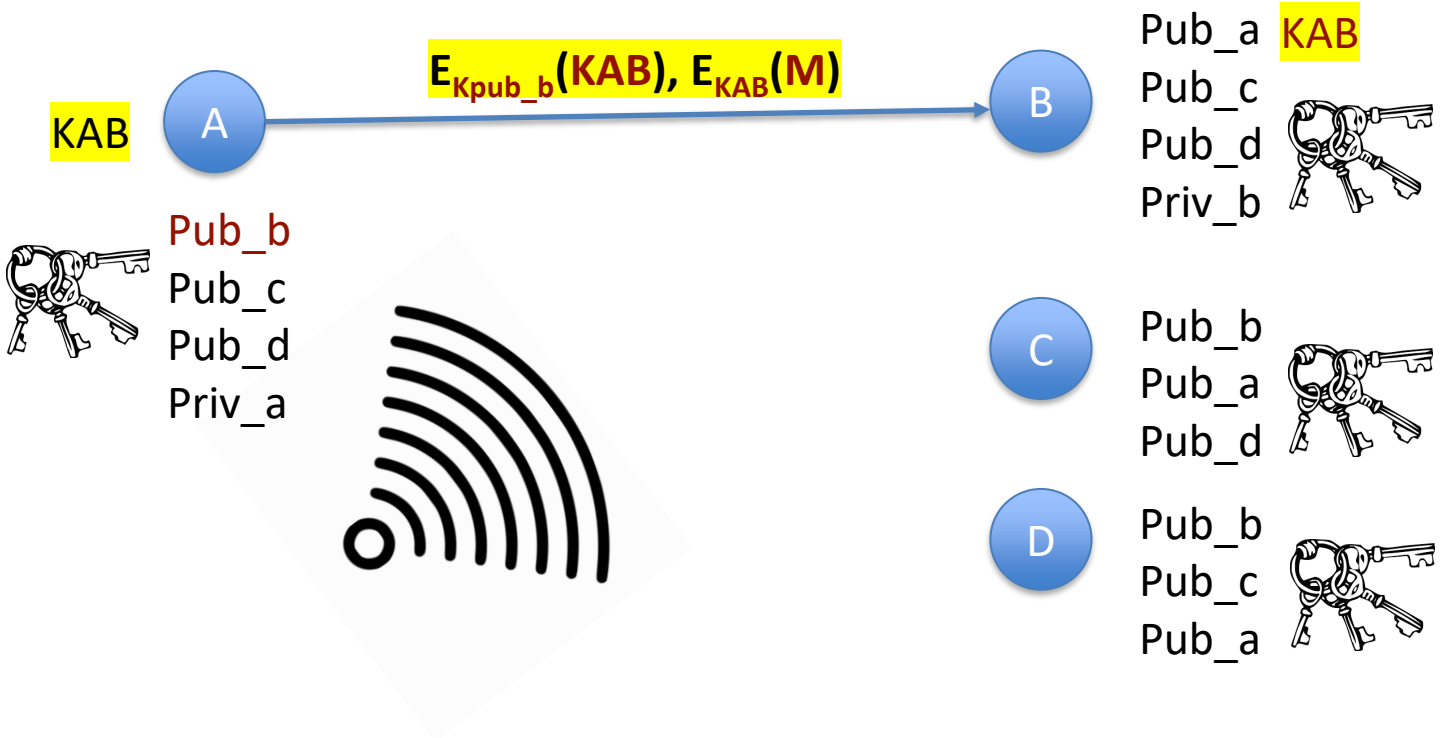
Fase 1: PROTEGER la KAB

Algunos casos de ejemplos – Queremos criptografía híbrida



Fase 2: PROTEGER el mensaje M

Algunos casos de ejemplos – Queremos criptografía híbrida



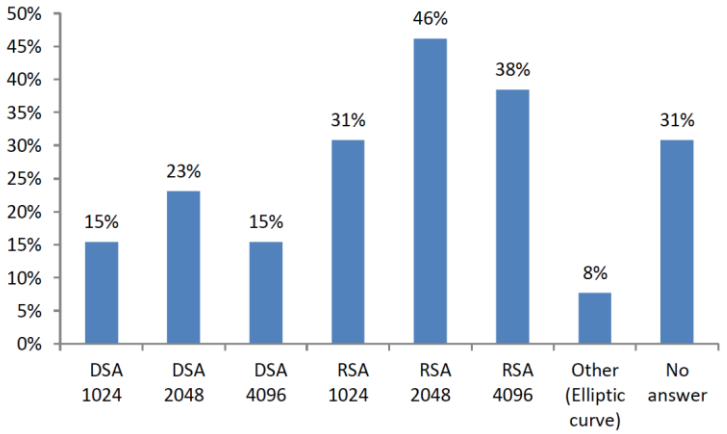
Fase 1 + 2: para proteger el canal de comunicaciones

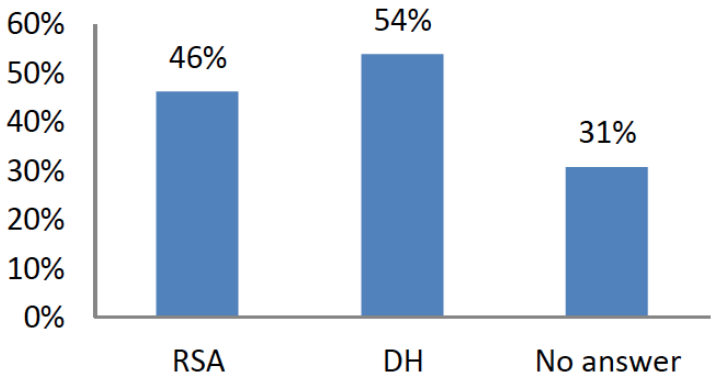
Funcionalidades de la criptografía de clave pública

- 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

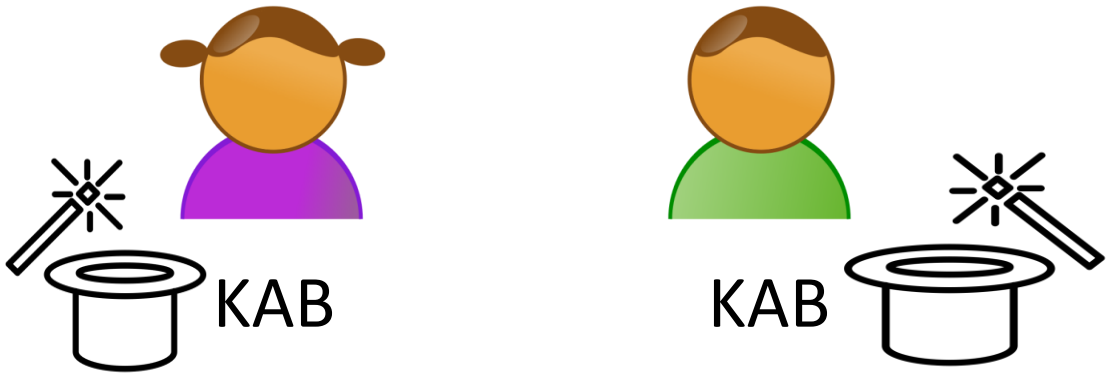
DSS: Digital Signature Standard, e incorpora
DSA (Digital Signature Algorithm)





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
a	a < q and a a primitive root of q

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = a^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = a^{X_B} \bmod q$

Calculation of Secret Key by User A
$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B
$K = (Y_A)^{X_B} \bmod q$

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

q = 3, α = 2 ; 2^1 % 3 = 1, 2^2 % 3 = 2

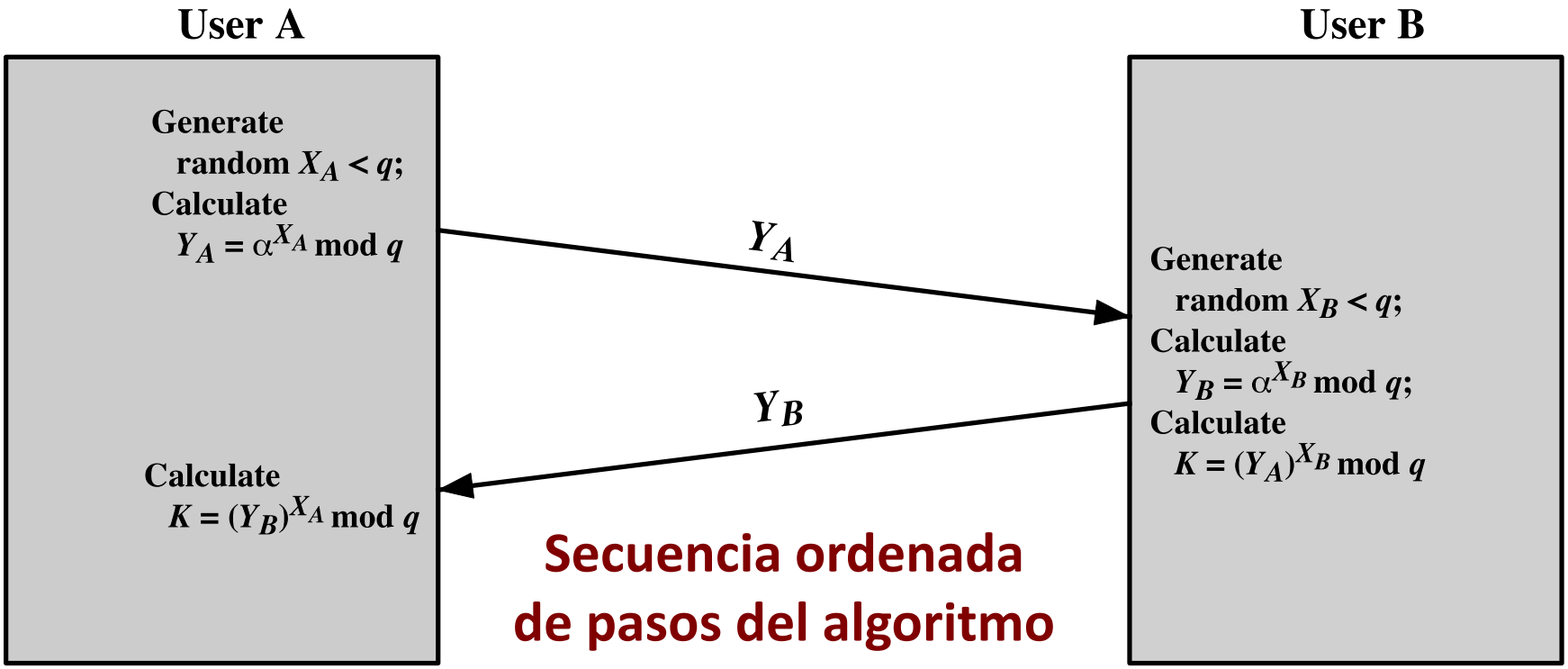
2) Tanto A como B generan sus claves privadas ($X_{A/B}$) y públicas ($Y_{A/B}$) teniendo en cuenta que: $Y_A \equiv \alpha^{XA} \pmod q$, donde $0 \leq X_A \leq (q-1)$, X_A **es el** ***logaritmo discreto de*** Y_A, y se representa $dlog_{\alpha,q}(Y_A)$

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: **KAB = $(Y_B)^{XA}$ mod q**

Otra forma de verlo …



Secuencia ordenada de pasos del algoritmo

Otra forma de verlo …

- **Alice:**
  - Valores públicos: q, α
  - Xa < q, clave privada
  - Ya = $\alpha^{Xa} \bmod q$, clave pública
  - Yb = $\alpha^{Xb} \bmod q$
  - KAB = $(Yb)^{Xa} \bmod q$
  - KAB = $(\alpha^{Xb})^{Xa} \bmod q$

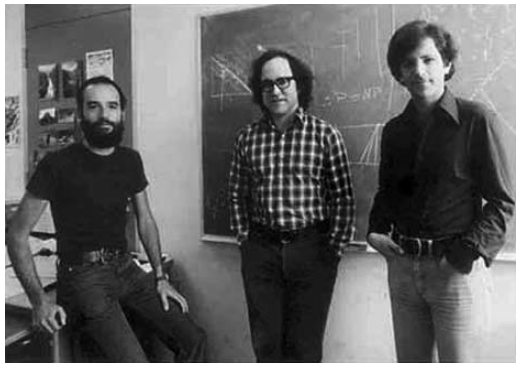
- **Bob:**
  - Valores públicos: q, α
  - Xb < q, clave privada
  - Yb = $\alpha^{Xb} \bmod q$, clave pública
  - Ya = $\alpha^{Xa} \bmod q$
  - KAB = $(Ya)^{Xb} \bmod q$
  - KAB = $(\alpha^{Xa})^{Xb} \bmod q$

KAB = $\alpha^{Xb\,Xa} \bmod q = \alpha^{Xa\,Xb} \bmod q$

Ejemplo para casa:
q = 353; α = 3 (raíz primitiva q); Xa = 97; Xb = 233

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

    $n = p * q \quad ; \quad p \neq q$

  - se calcula $\varphi(n)$, de forma que

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

    Para extraer los primos, donde ϕ(n) (función indicatriz de Euler) 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

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

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

  - Se determina el número *d* que cumple

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

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

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*: $M = C^d \pmod n$

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

- RSA se trata, por lo tanto, de un algoritmo exponencial

- 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: mcd(e,160)=1 y $e < 160$
se selecciona $e = 7$

⑤ Determinar d: $d.e \equiv 1 \bmod 160$
d = 23 dado que $23 \times 7 = 161 \bmod 160 = 1$

⑥ Clave pública = 7 y módulo = 187

⑦ Clave privada = 23

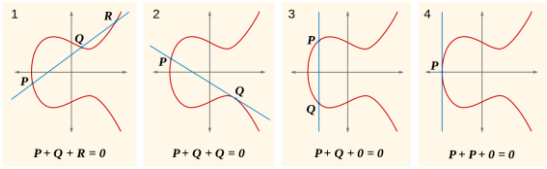
- El texto original, *M* (y también el cifrado, *C*) debe tomarse como un **número decimal**
  - De esta forma, si se trabaja con el código ASCII:

    M = “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

    $m_0 = 72111$, $m_1 = 10809$ y $m_2 = 7$

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

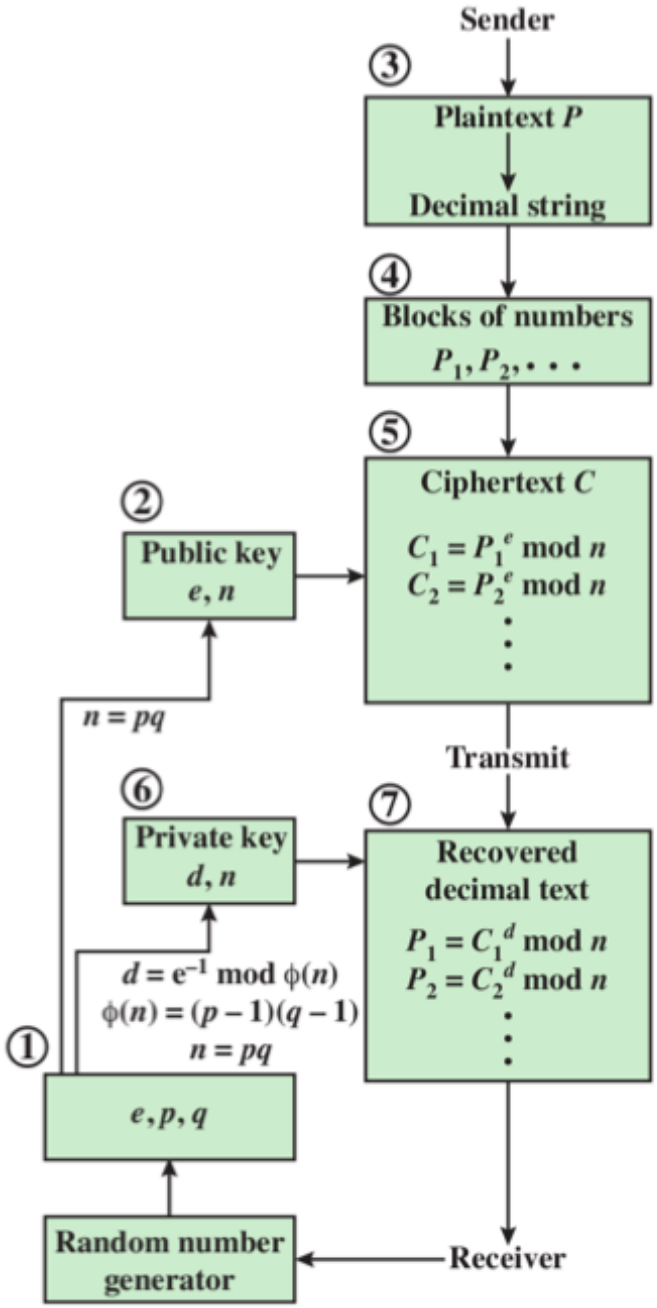
- Cosas que hay que tener en cuenta:
  - Los bloques m_i no deben/pueden ser pequeños, pues entonces el criptoanalista puede construirse una tabla donde tenga todos los posibles m_i y sus respectivos c_i
  - También hay que tener cuidado al elegir el tamaño
    - **Cuanto mayor sea, más seguro, pero 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
  - Estos sistemas se basan en otros principios matemáticos
    - Ejemplo: ECC, con el *problema de logaritmos discretos en curvas elípticas (ECDLP)*

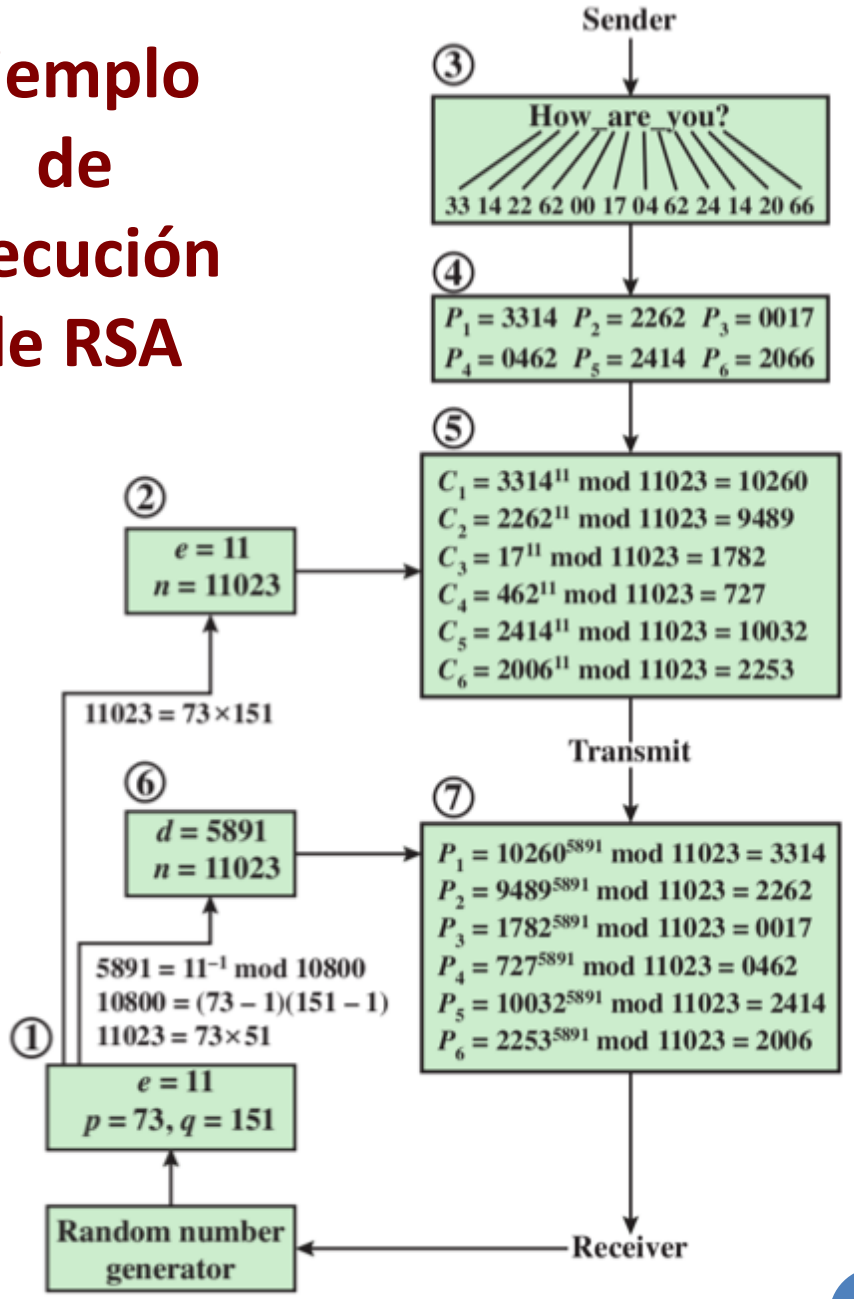


Tamaño de clave simétrica (bits)	Tamaño de clave RSA y DH (bits)	Tamaño de clave de ECC (bits)
128	**1024**	**160**
192	2048	224
256	3072	256
	7680	384
	15360	521

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|