# Algoritmos Probabilísticos
# Algoritmo RSA de criptografía

# Algoritmos Probabilísticos

# Algoritmos no deterministas

- Cuando en un problema hay que tomar una decisión, puede ser mejor tomarla de forma aleatoria en vez de calcular cuál de las alternativas es mejor.

- Esto se puede hacer cuando:
  - El tiempo de cálculo de la decisión óptima es mucho mayor que el requerido en el caso medio al tomar la decisión en forma aleatoria.
  - Problemas con múltiples soluciones correctas.

# Algoritmos no deterministas

- Un algoritmo probabilista puede comportarse de manera distinta al aplicarlo dos veces a un mismo caso.
  - Tanto el tiempo de ejecución como el resultado pueden variar.
- A un algoritmo determinista no se le permite que calcule una solución incorrecta para ningún dato de entrada.
  - Un algoritmo probabilista puede equivocarse siempre que esto ocurra con una probabilidad razonablemente pequeña para cada dato de entrada.
  - Repitiendo la ejecución un número suficiente de veces para el mismo dato, puede aumentarse tanto como se quiera el grado de confianza en obtener la solución correcta.

# Algoritmos no deterministas

- Una respuesta incierta dada por un algoritmo probabilista no sólo se obtiene más deprisa, sino que además podemos confiar en ella.

- El tiempo promedio de un algoritmo determinista se define como el tiempo promedio requerido por el algoritmo cuando se consideran igualmente probables todos los ejemplares posibles de un tamaño dado.

- El tiempo esperado de un algoritmo probabilista se refiere al tiempo promedio que se necesitaría para resolver un ejemplar una vez.

# Generación de números pseudoaleatorios

- Una serie de números es aleatoria cuando no se descubre en ella ninguna propiedad de: generación, repetición, etc.

- Propiedades que deben cumplir las series de números aleatorios:
  - Han de estar uniformemente distribuidos.
  - Han de ser estadísticamente independientes.
  - Que requieran poco espacio en memoria.
  - Que se obtengan rápidamente.

# Generación de números pseudoaleatorios

Sean a y b dos números reales tales que $a < b$.

- Una llamada a *uniforme(a,b) devuelve un número real x* seleccionado aleatoriamente en el intervalo $a \leq x < b$.
- La distribución de *x es uniforme en el intervalo, y las sucesivas* llamadas al generador son independientes de las anteriores.

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Generación de números pseudoaleatorios

Versión 1.0

**uniforme (Real a, Real b ): Real**

**{pre: a < b, existe la función uniforme(0,1) que genera un número aleatorio entre 0 y 1}** **{pos: a ≤ x ≤ b}**

| | | |
|---|---|---|
| 1<br>2 | **x = a + (b-a) * uniforme(0,1)**<br>**regrese x** | **-x: Real: almacena un número aleatorio entre a y b.** |

# Generación de números pseudoaleatorios

- Debido a que generadores  aleatorios no suelen estar disponibles, se emplean generadores pseudoaleatorios.
- Generalmente cada lenguaje tiene un generador de números pseudoaleatorios entre 0 y 1, partiendo de una semilla determinada por el lenguaje o introducida por el programador.
- Los métodos de generación de números pseudoaleatorios más comunes son los llamados *métodos congruenciales.*
- El más famoso es el de Lehmer que depende de tres datos:

$$X(n+1) = (a * X(n) + c) \bmod m$$

- Propiedades:
  - Fácilmente reproducible (comenzando por la misma semilla).
  - Se obtiene rápidamente.
  - Ocupa poco espacio de memoria.

# Clasificación de los algoritmos probabilistas

- Los algoritmos numéricos:
  - Otorgan una solución aproximada.
  - Otorgan un intervalo de confianza.
  - A mayor tiempo de ejecución, mejor es la aproximación.
- Los algoritmos de Monte Carlo:
  - Otorgan la respuesta exacta con elevada probabilidad.
  - En algunas ocasiones arrojan una respuesta incorrecta.
  - No se puede saber si la respuesta es correcta, pero se puede reducir la probabilidad del error dando más tiempo al algoritmo.
- Los algoritmos de Las Vegas:
  - Toman decisiones al azar.
  - Si no encuentran las solución correcta lo admiten.
  - Es posible volver a intentarlo con los mismos datos hasta obtener la solución correcta.

# Clasificación de los algoritmos probabilistas

Si preguntáramos ¿Cuándo descubrió América Cristóbal Colón?

- Las respuestas según el algoritmo serían:
- Algoritmo numérico:
  - Entre 1490 y 1500
  - Entre 1485 y 1495
  - Entre 1492 y 1592
  - Entre 1481 y 1491
  - Entre 1489 y 1499
- Algoritmo de Monte Carlo:
  - 1492, 1492, 1492, 1491, 1492, 1492, 357 a.C., 1492, 1492, 1492.
- Algoritmo de Las Vegas:
  - 1492, 1492, ¡Perdón!, 1492, 1492, 1492, 1492, 1492, ¡Perdón!,

# Algoritmos Probabilistas Numéricos

- La aleatoriedad se utilizó por primera vez en algoritmia para resolver aproximadamente problemas numéricos
- En algunos problemas de la vida real el cálculo de la solución exacta no es posible (incertidumbre de los datos, limitaciones de los ordenadores digitales).
- En otros existe una respuesta precisa pero se tardaría demasiado tiempo en obtenerla.
- La respuesta de un algoritmo probabilista numérico es siempre aproximada, pero su precisión esperada mejora a medida que crece el tiempo disponible para el algoritmo.

# Algoritmos Probabilistas Numéricos

- Un algoritmo numérico muy conocido es la integración de Monte Carlo, aunque no es un algoritmo de Monte Carlo.

- Calcular: $I = \int f(x)\,dx$ *donde* $f : R \rightarrow R \geq 0$ *es continua y* $a \leq b$

- Consideremos el rectángulo de la figura de anchura $b-a$ y altura $I/(b-a)$ *(la altura media de f entre a y b)*

- El área del rectángulo es el cálculo a hacer

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Algoritmos Probabilista

Versión 1.0

intDET (función f, Entero n, Real a, Real b ): Real

{pre: a < b} {pos: I es el valor de la integral}

| | | |
|---|---|---|
| 1<br>2<br>3<br>4<br>5 | suma = 0<br>delta = (b-a)/n<br>x = a + delta/2<br>(suma = suma + f(x)<br>x = x + delta)j = 1, n<br>regrese I = suma * delta | -suma: Real: almacena sumas parciales de la función evaluada en diferentes puntos.<br>-delta: Real: almacena el tamaño de los tramos de integración.<br>-x: Real: almacena el valor de los puntos de integración.<br>-n: Entero: número de intervalos de integración.<br>-I: Real: Valor de la integral aproximado.<br>-j: Entero: variable auxiliar para la estructura de repetición. |

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Algoritmos Probabilistas

| | | |
|---|---|---|
| | | Versión 1.0 |
| | intMC (función f, Entero n, Real a, Real b ): Real | |
| {pre: a < b} | | {pos: I es el valor de la integral} |
| 1<br>2<br><br><br>3 | suma = 0<br>(x = uniforme(a,b)<br>suma = suma + f(x) )j = 1, n<br>regrese I = (b-a) * (suma/n) | -suma: Real: almacena sumas parciales de la función evaluada en diferentes puntos.<br>-x: Real: almacena el valor de los puntos de integración.<br>-n: Entero: número de intervalos de integración.<br>-I: Real: Valor de la integral aproximado.<br>-j: Entero: variable auxiliar para la estructura de repetición.. |

# Algoritmos Probabilistas Numéricos

- El uso de los algoritmos probabilistas para integrar se justifica con las integrales múltiples.

- En un algoritmo determinista en ese caso el número de puntos de muestra necesario para lograr una precisión dada, crece de manera exponencial con la dimensión de la integral.
  - Ejemplo: 100 puntos en una integral simple, se convertirían en 100 x 100 puntos para alcanzar la misma precisión en una integral doble, 1000 x 1000 en una triple, ...

- En el algoritmo probabilista la dimensión de la integral no tiene mucho efecto sobre la precisión de la integral.

El algoritmo probabilista se utiliza en integrales de 4 o más dimensiones,

# Conteo Probabilista

- Empleando la notación binaria ordinaria, se puede contar desde 0 hasta $2^n - 1$.

- Sea *c un registro de n bits, podríamos implementar tres* funciones:
  - *iniciar(c): coloca el contador en cero.*
  - *pulsar(c): le suma 1.*
  - *contar(c): pide su valor actual.*

- Dado que *c sólo puede admitir* $2^n$ *valores distintos, el contador* se ve obligado a tomar un valor que ya haya tenido anteriormente

# Conteo Probabilista

- Si relajamos cualquier requisito que le exija a *contar regresar el* número exacto de llamadas a *pulsar desde el último iniciar, con* una estrategia probabilista podríamos contar el doble.
- Cada vez que se llame a *pulsar, lanzamos al aire una moneda* imparcial. Si sale *cara, sumamos 1 al registro; si sale cruz, no* hacemos nada.
- De esta forma, se puede llegar a contar $2^{n+1}$ *- 2 veces con un* registro de *n bits.*
- Bastan sólo 8 bits para contar más de 5*1076 sucesos.
- La idea consiste en mantener en el registro una estimación del logaritmo del número de llamadas a *pulsar, lo que implica que contar(c) devuelve* $2^c$*-1.*
- Supongamos que $2^c$*-1 es una buena aproximación del número* de llamadas a *pulsar, desde la última inicialización. Luego* decimos que el número de llamadas a *pulsar pasa a ser* $2^{c+1}$*-1* una probabilidad *p y permanece igual con una probabilidad* 1-*p; siendo el valor esperado* $2^c + 2^c p - 1$.

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Algoritmos Probabilistas

| | | |
|---|---|---|
| | | Versión 1.0 |
| | pulsar (registro c) | |
| {pre: existe el registro c} | | {pos: v es el conteo del registro c } |
| 1<br><br><br>3 | (si (tiramoneda = cara)<br>entonces c = c +1<br>) i = 1, n | -i: Entero: variable auxiliar para la estructura de repetición.<br>-n: Entero: número de bits del registro c. |

*contar(c): regresa $2^c$ - 1*

# ALGORITMOS DE LAS VEGAS

- Utilizan “azar” para alcanzar una solución correcta más rápido que un algoritmo determinista.

- Siempre alcanzan una solución correcta o reconocen una solución errónea.

- El problema de selección,
  - Tiene un algoritmo determinista con un tiempo logarítmico en el caso promedio,
  - Puede llegar a ser cuadrático en el peor caso

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Algoritmos Probabilistas

Versión 1.0

*particionar(Arreglo de Enteros[1..n]: T, Entero: i, j, m, u, v)*

*{pre: n > 0}* *{pos: n > 0}*

| | | |
|---|---|---|
| 1 | u, v = i,i | Auxiliar Entero: i, j. Almacenan los limites del arreglo que se esta evaluando.<br><br>Las variables u y v debe ser pasadas como referencia, es necesario que conserven su valor luego de terminar la Corrida |
| 2 | [ele = T[k]<br>  si( ele < m ) entonces<br>  T[k] = T[v]<br>  T[v] = T[u]<br>  T[u] = ele<br>  u, v = u+1, v+1<br>  sino si( ele = m )<br>  T[k] = T[v]<br>  T[v] = ele<br>  v = v+1<br>  f_si<br>]k = i, k <= n | |
| 3 | v = v1 | |

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Algoritmos Probabilistas

Versión 1.0

*seleccion(Arreglo de Enteros[1..n]: T, Entero: k): Entero*

*{pre: n > 0, 0 < k < n+1}* *{pos: n > 0, 0 < k < n+1}*

| | | |
|---|---|---|
| 1 | i, j = 1, n | Auxiliar Entero: i, j, m, u, v. Almacenan los limites del arreglo. |
| 2 | ( i < j )[ | |
| | m = T[1] | |
| | particionar(T, i, j, m, u, v) | particionar: función que particiona un arreglo, coloca los elementos menores a “m” antes de “m” y los mayores que “m” despues de “m” |
| | si ( k < u ) entonces | |
| | j = u1 | |
| | sino si( k > v ) entonces | |
| | i = v+1 | |
| | sino entonces | |
| | i, j= k, k | |
| | f_si | |
| | ] | |
| 3 | Regresar T[i] | |

UNIVERSIDAD DE LOS ANDES
MÉRIDA VENEZUELA

# Algoritmos Probabilistas

Versión 1.0

*particionar(Arreglo de Enteros[1..n]: T, Entero: i, j, m, u, v)*

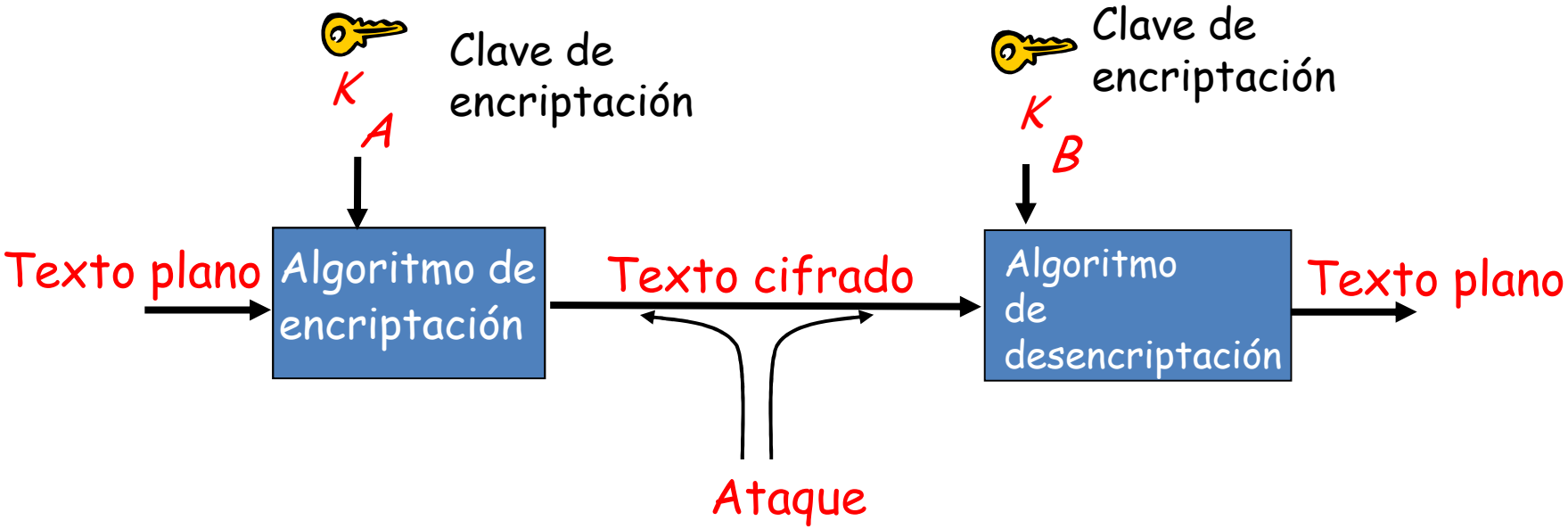
*{pre: n > 0}* *{pos: n > 0}*

| | | |
|---|---|---|
| 1 | i, j = 1, n | Auxiliar Entero: i, j, m, u, v. Almacenan los limites del arreglo. |
| 2 | ( i < j )[<br>  m = T[uniforme(i, j)]<br>  particionar(T, i, j, m, u, v)<br>  si ( k < u ) entonces<br>  j = u1<br>  sino si( k > v ) entonces<br>  i = v+1<br>  sino entonces<br>  i, j= k, k<br>  f_si<br>] | particionar: función que particiona un arreglo, colocando los elementos menores a “m” antes de “m” y los mayores que “m” despues de “m”<br><br>uniforme: función que regresa un valor aleatorio entero, perteneciente al rango [i,j].. |
| 3 | Regresar T[i]. | |

# Algoritmo RSA de criptografía

# Cifrado

- Un mensaje puede cifrarse mediante la aplicación de una regla que transforme el texto en claro del mensaje a un texto cifrado.
- Para esto el receptor debe conocer la regla inversa para transformar el texto cifrado en el texto original.



# Cifrado

*La criptografía es el uso de la transformación de datos para hacerlos incomprensibles a todos, excepto a los usuarios a quienes están destinados.*

- El *problema de la intimidad* trata de cómo *evitar la obtención no autorizada* de información de un canal de comunicaciones.

- El *problema de la autentificación* trata sobre *cómo proporcionar al receptor* de un mensaje prueba de la *identidad del remitente*, que serían el equivalente electrónico de una firma escrita.

# AUTENTIFICACION

- Message Authentication Code (MAC):
  - Jose y Maria escogen *f(m)* para colocar a cada mensaje de texto *m* un numero a=*f(m)*.
  - *f(m)* debe evitar colisiones o manejarse como una *hash* para mapear diferentes mensajes en diferentes números.
  - Jose envía *m* con un numero *a=f(m)*.

    *a* es el *autenticador*.
  - Maria recibe *m*, computa *b=f(m)* y se asegura *a=b*.

# Sistema de Intimidad Criptográfica

- El *remitente* desea transmitir cierto *mensaje no cifrado* (texto simple) a un *receptor* legítimo
- La transmisión se producirá a través de un *canal inseguro*
- El remitente pasa el *texto simple* a una *unidad de codificación* que lo transforma en un *texto cifrado o criptograma:*

No es comprensible para el *espía.*

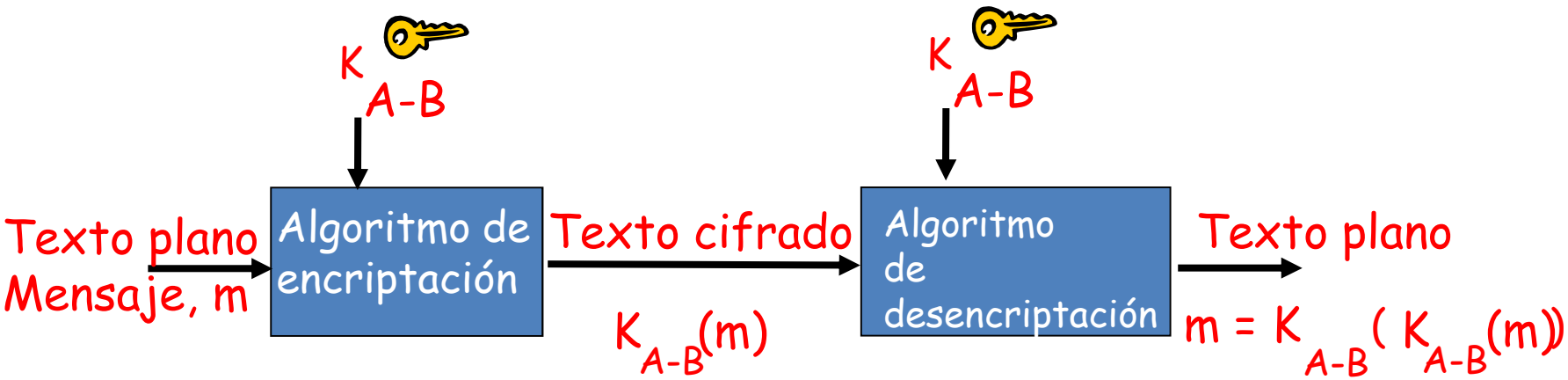
- Se transmite en forma *segura* por un *canal inseguro.*
- El receptor pasa el texto cifrado por una *unidad de descifrado* para regenerar el texto simple.

**Criptoanálisis** Es el proceso de *intentar regenerar el texto simple a partir del texto cifrado*, pero *desconociendo la clave de ciframiento*

Es la tarea del espía!!!

# Cifrado

- Criptografía convencional.
  - Por sustitución: remplaza cada letra del mensaje original por otra.
  - Con clave de un solo uso
  - Por transposición: cambiar el orden los caracteres en el mensaje.



# Cifrado simétrico

P = e (k,M) ;  M = d (k,P)

- Usa una misma clave para cifrar y para descifrar. Las dos partes que se comunican mediante el cifrado simétrico deben estar de acuerdo en la clave a usar de antemano. Una vez de acuerdo, el remitente cifra un mensaje usando la clave, lo envía al destinatario, y éste lo descifra usando la misma clave. Ejemplos: DES, AES, Etc
- Ejemplo
  - A y B comparten la clave K (011100)
  - A cifra un mensaje m y lo envía a B
    - E(M)=m XOR k
    - k = 011100 m = 010110
    - E(m)= 001010 = c
  - B recibe el mensaje y calcula D(c)=c XOR k
    - D(E(m)) = D(m XOR k) = (m XOR k) XOR k = m

# Cifrado asimétrico o de Clave Publica

- Las *funciones de cifrado y descifrado* están *separadas* y utilizan *distintas claves.*

$$P = e\ (Kpub, M)$$

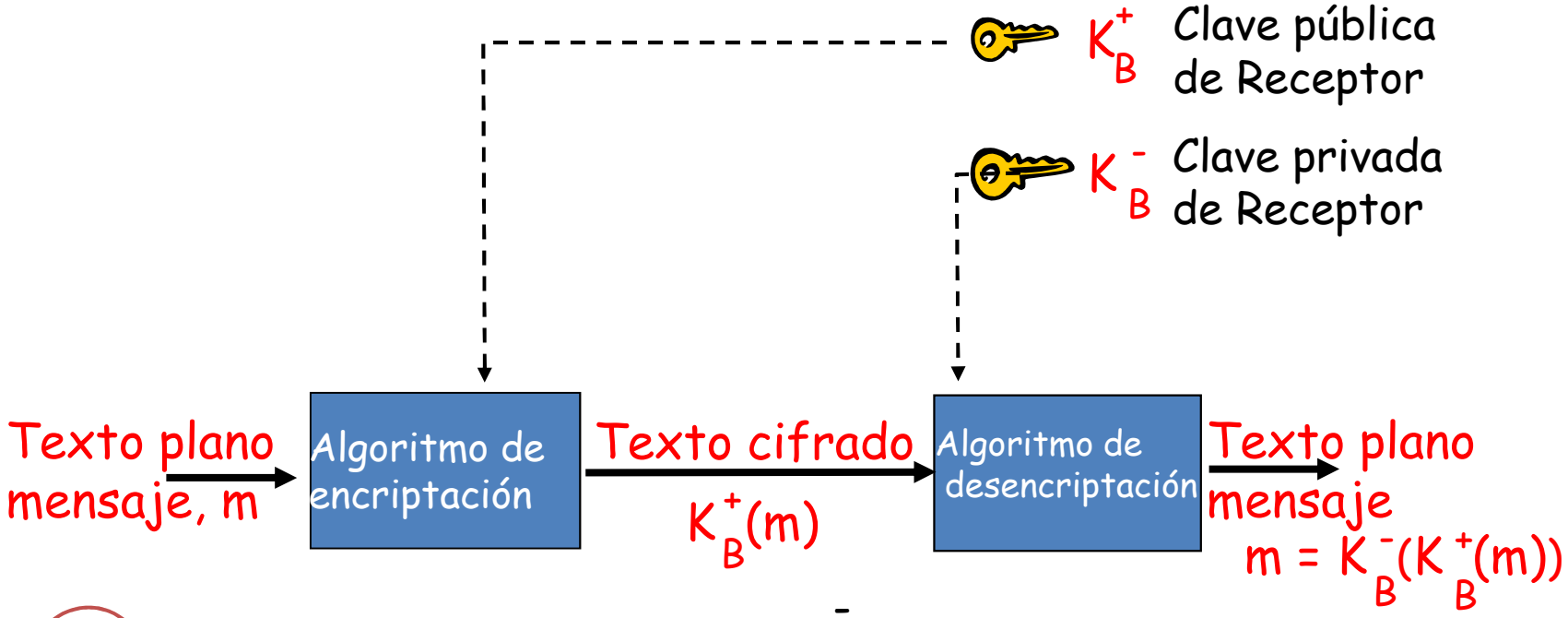
$$M = d\ (Kprv, P)$$

- No es *computacionalmente* posible (en un tiempo *"razonable"*) determinar la *clave de desciframiento "d"* a partir de la *clave de ciframiento "e".*

- *"e"* puede hacerse *pública* sin comprometer la seguridad de *"d"*, que permanece *privada:*
  - Se simplifica el problema de la distribución de claves.

Ejemplos: RSA, DH, DSA, Curvas Elípticas

# Cifrado asimétrico o de Clave Publica




1. Se necesita $K_B^+( )$ y $K_B^-( )$ tal que:

$$K_B^-(K_B^+(m)) = m$$

2. Dada la clave pública $K_B^+$, debería ser imposible computar una clave privada $K_B^-$

# Cifrado asimétrico

- Cada entidad debe poseer una llave publica y una privada y se denotan como:
  - Pa llave publica de A
  - Sa llave privada de A
  - Pb llave publica de B
  - Sb llave privada de B
- La funciones de clave publica y la privada para un ente x son funciones inversas entre si:
  - M=Sa(Pa(M))
  - M=Pa(Sa(M))

Los sistemas de clave publica se basan en que todos conocen las claves publicas de los entes con los cuales se van a comunicar. Estas llaves pueden se obtenidas de un directorio publico o del ente con el cual se va ha establecer la comunicación

# Cifrado y Firma Digital

- **Cifrado (secreto):**
  - Texto sin cifrar
  - Algoritmo de Cifrado (Ej. RSA), usando clave publica del receptor
  - transmisión
  - Algoritmo de descifrado (inverso de cifrado) usando clave privada del receptor
  - Texto sin cifrar

Cifrado: $P = e(K_{pub}, M)$

Descifrado: $M = d(K_{prv}, P)$

- **Firma digital (autenticidad):**
  - Texto sin cifrar
  - Algoritmo de Cifrado (Ej. RSA), usando clave privada del emisor
  - transmisión
  - Algoritmo de descifrado (inverso de cifrado) usando clave publica del emisor
  - Texto sin cifrar $P = e(K_{prv}, M)$

$M = d(K_{pub}, P)$

# Cifrado y Firma Digital

- **Autenticidad y secreto**:
  - Texto sin cifrar
  - Algoritmo de Cifrado (Ej. RSA), usando clave privada del emisor
  - Algoritmo de Cifrado (Ej. RSA), usando clave publica del receptor
  - transmisión
  - Algoritmo de descifrado (inverso de cifrado) usando clave privada del receptor
  - Algoritmo de descifrado (inverso de cifrado) usando clave publica del emisor
  - Texto sin cifrar

# RSA

El sistema RSA crea sus claves de la siguiente forma:

- Se buscan dos números primos lo suficientemente grandes: p y q con p≠q (de entre 100 y 300 dígitos).
- Se obtienen los números n = p*q y Ø = (p-1)*(q-1).
- Se busca un número e (impar) tal que no tenga múltiplos comunes con Ø. (Seleccione un entero de forma aleatoria e tal que 1< *e < Ø, MCD(Ø, e) = 1).*
- El numero e es llamado exponente publico RSA
- El sistema RSA crea sus claves de la siguiente forma
  - Se calcula el exponente privado de RSA

    d = e-*1 (mod Ø), (Sea d un numero tal que (e*d)-1* sea divisible entre Ø).
  - Publique la llave publica (e,n) y guarde la llave privada (d,n)

# RSA: encriptación, desencriptación

0. Dados $(n,e)$ y $(n,d)$ calculados anteriormente.

1. Para encriptar patrón de bit, $m$, calcular:

$c = m^e \bmod n$ (es decir, el resto cuando $m^e$ se divide por $n$).

2. Para desencriptar el patrón de bit recibidos, $c$, calcular:

$m = c^d \bmod n$ (es decir, el resto cuando $c^d$ se divide por $n$).

¡Magia!

$$m = \underbrace{(m^e \bmod n)}_{c}{}^d \bmod n$$

# Ejemplo RSA

Receptor elige $p$=5, $q$=7. Entonces $n$=35, $z$=24.

$e$=5 (entonces $e$, Ø primo relativo).
$d$=29 (entonces $ed$-1 es divisible de forma exacta entre Ø)

Encriptación:

| letra | $m$ | $m^e$ | $c = m^e \bmod n$ |
|---|---|---|---|
| l | 12 | 1524832 | 17 |

Desencriptación:

| $c$ | $c^d$ | $m = c^d \bmod n$ | letra |
|---|---|---|---|
| 17 | 481968572106750915091411825223071697 | 12 | l |

# RSA

## Ejemplo

- Generación de llaves
  - p=11, q=23
  - m= p * q = 253 ; (p-1)*(q-1)= (11-1)*(23-1)= 220
  - e= 3
  - d = 147
- Cifrado
  - Mensaje = “adios“ -> 0 3 8 14 18
  - 0^3, 3^3, 8^3, 14^3, 18^3, mod 253 = 0 27 6 214 13
- Descifrado (0 27 6 214 13 )
  - 0^147, 27^147, 6^147, 214^147, 13^147, mod 253 = 0 3 8 14 18
  - 0 3 8 14 18 -> “ adios

Este algoritmo se basa en la exponenciación modular, el cual es un algoritmo que se puede implementar en tiempo polinomial de la longitud de la entrada

$$O(\log^3 n) \approx O(k^3),$$

donde $n$ es la entrada y $k$ es su longitud.

# Tiempos de búsqueda sistemática a un millón de tentativas por segundo

| Largo Clave | 4 bytes | 6 bytes | 8 bytes |
|---|---|---|---|
| Letras minúsculas (26) | 0.5 s. | 5 mín. | 2.4 días |
| Caracteres alfanuméricos (62) | 15 s. | 16 h | 6.9 años |
| Caracteres ASCII (256) | 1.2 h | 8.9 años | 580000 años |

# EXPONENCIACIÓN MODULAR

El problema radica en calcular $m^n \bmod z$, para $n$, $m$ enteros suficientemente grandes. La solución se obtiene desarrollando un algoritmo divide y vencerás junto con las siguientes propiedades de aritmética modular

$$mn \bmod z = [(m \bmod z)(n \bmod z)] \bmod z$$

$$m^n \bmod z = (m \bmod z)^n \bmod z$$

# EXPONENCIACIÓN MODULAR

- Luego el algoritmo divide y vencerás será

- Si $n$ es par, hacemos $n = 2k_1$, donde $k_1 \in$ N. Así

$$m^n \bmod z = m^{2k_1} \bmod z = \left(m^2 \bmod z\right)^{k_1} \bmod z$$

- Si $n$ es impar, hacemos $n = 2k+1$, donde $k \in$ N. Así

$$m^n \bmod z = m^{2k+1} \bmod z = \left(m^{2k} m \bmod z\right)$$
$$= \left[\left(m^{2k} \bmod z\right)\left(m \bmod z\right)\right] \bmod z$$