Implementación de claves criptográficas

EDUARD LARA

INDICE

1. IMPLEMENTACIÓN DE CLAVES

Autenticación

La **autenticación** es el proceso de confirmar la autenticidad reclamada por una entidad. Tenemos básicamente dos tipos:

- Autenticación **de autor**: el autor es quien dice ser. Se puede conseguir con una **signatura digital**.
- Autenticación **de dades**: les dades no han estat modificades. Se puede conseguir con un algorisme de **resumen de mensaje** (message digest).

La autenticación no quiere decir que tengamos encriptación.

1. IMPLEMENTACIÓN DE CLAVES

Cifrado

La **criptografía** (del griego "*kryptos*" - escondido, secreta - i "*graphin*" - escritura. Por tanto seria "**escritura oculta**") es el estudio de formas de convertir información desde su forma original hacia un código incomprensible, de forma que sea incomprensible para los que no conozcan esta técnica.

En la terminología de criptografía, encontramos los siguientes aspectos:

- La **información original** que ha de protegerse y que se denomina texto en claro o **texto plano**.
- El **cifrado** es el proceso de convertir el texto plano en un texto ilegible, llamado **texto cifrado** o **criptograma**.
- Las **claves** son la base de la criptografía, y son cadenas de números con propiedades matemáticas. La seguridad depende tanto del algoritmo de cifrado como del nivel de secreto que se le dé a la clave.

1. IMPLEMENTACIÓN DE CLAVES

Tipos de cifrado – Tipos de claves

Existen dos grandes grupos de algoritmos de cifrado, en función de si las claves son únicas o van en pareja:

- **Cifrado simétrico o de clave privada (Ks).** Las claves de cifrado y descifrado son la misma. Dan lugar al modelo de clave privada.
- **Cifrado asimétrico o de clave publica (Kp y Kc).** Las claves de cifrado y descifrado son diferentes, una pública y otra privada, que permite cifrar con una cualquiera y descifrar con la otra. Están relacionadas entre sí de algún modo. Dan lugar al modelo de clave publica. Se utilizan para dos propósitos:
  - **Cifrado con clave pública:** un mensaje cifrado con clave pública sólo se puede descifrar con la clave privada.
  - **Firma digital:** un mensaje firmado con la clave privada puede ser verificado por cualquiera con la clave pública.

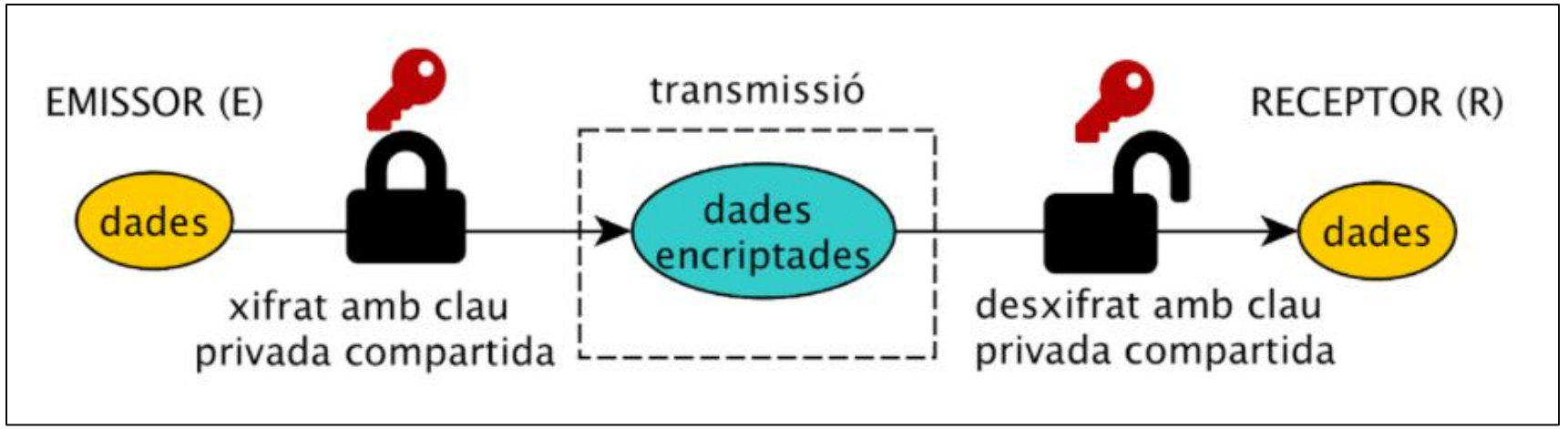
1. IMPLEMENTACIÓN DE CLAVES

Cifrado de bloque o de stream

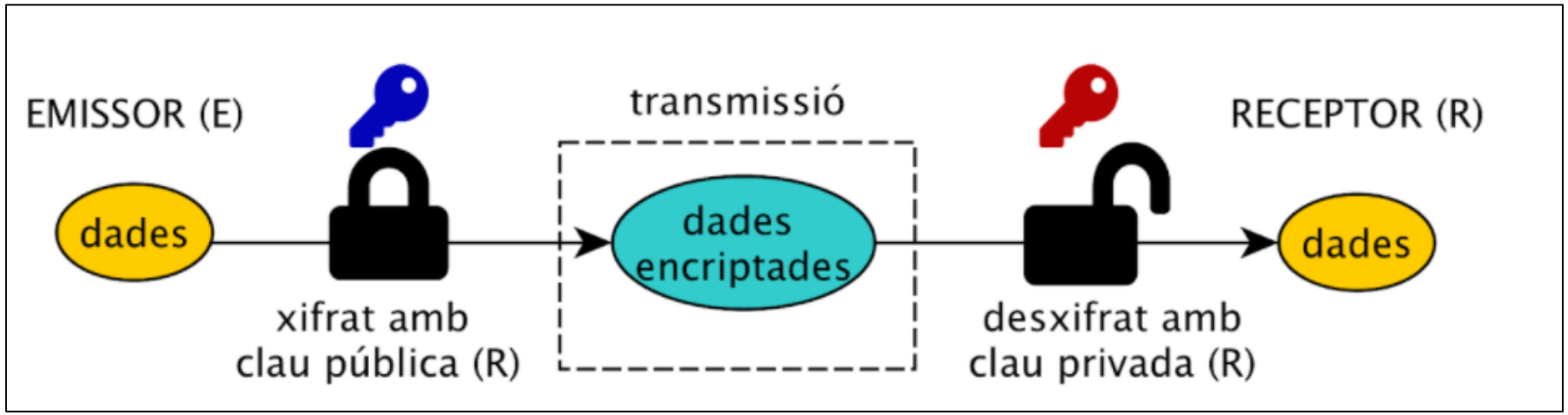
- Bloque: Cifrado de bloques de tamaño fijo de datos. Pueden ser asimétricos o simétricos. Si no es dice lo contrario, hablamos solo de cifrados en bloque.
- Stream: Cifrado de un stream (corriente) de bits o bytes. Son simétricos.

1. IMPLEMENTACIÓN DE CLAVES

Cifrado con clave privada: las dos partes comparten una clave privada



Cifrado con clave pública: Sólo el propietario de las llaves (receptor) puede descifrar los datos



1. IMPLEMENTACIÓN DE CLAVES

	Clave Simétrica	Clave asimétrica
Ventajas	• Algoritmos rápidos sin limitaciones	• Se puede compartir la parte pública de la clave, ya que sin la parte privada no se puede hacer nada.
Desventajas	• Se necesita una clave para cada pareja origen/destino. • El problema es cómo transmitir la clave para que el emisor (cifrador) y el receptor de la información (descifrador) tengan ambos la misma clave	• Los algoritmos son más lentos y tienen limitaciones en la mida del mensaje a cifrar (RSA tiene un mensaje máximo de floor(n/8)-11 bytes.

2. MODELO DE CLAVE SECRETA

Definición:

- Tanto el emisor como el receptor del mensaje utilizan la misma clave Ks
- Presenta un buen rendimiento

Problemas:

- Como el emisor y el receptor obtienen la misma clave
- Necesidad de emplear claves distintas para comunicarse con cada uno de los posibles receptores de la información.
- La validez de una clave se pone en entredicho a medida que se va utilizando. Cuantos más mensajes se cifren con la misma clave, más expuesta estará a un análisis estadístico.

Finalidades:

- Cifrar canales de comunicación.
- Cifrar datos para su almacenamiento.

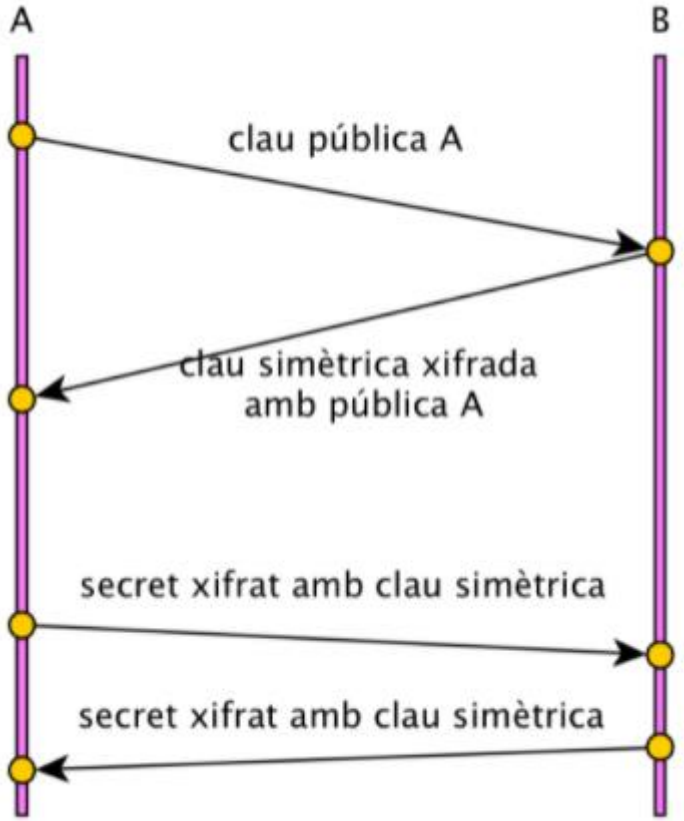
2. MODELO DE CLAVE SECRETA

Intercambio seguro de clave simétrica

- El cifrado simétrico ha de compartir la clave entre las dos partes.
- El asimétrico, no permite cifrar blocs muy grandes.

La solución es combinar los dos tipos de cifrado:

- A comparte (en claro) su clave pública.
- B la utiliza para cifrar la clave simétrica (sólo A puede descifrarla).
- A descifra la clave simétrica, ya que tiene la clave privada A.
- Las dos partes ya tienen la clave simétrica para intercambiar mensajes.



2. MODELO DE CLAVE SECRETA

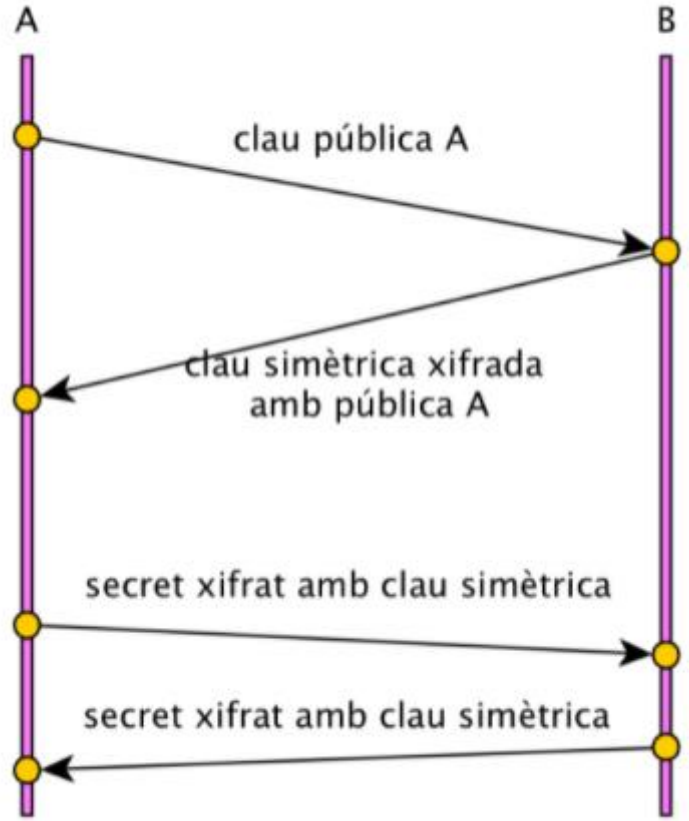
Ventajas intercambio seguro de clave simétrica

• La clave simétrica se comparte de forma segura mediante el cifrado asimétrico

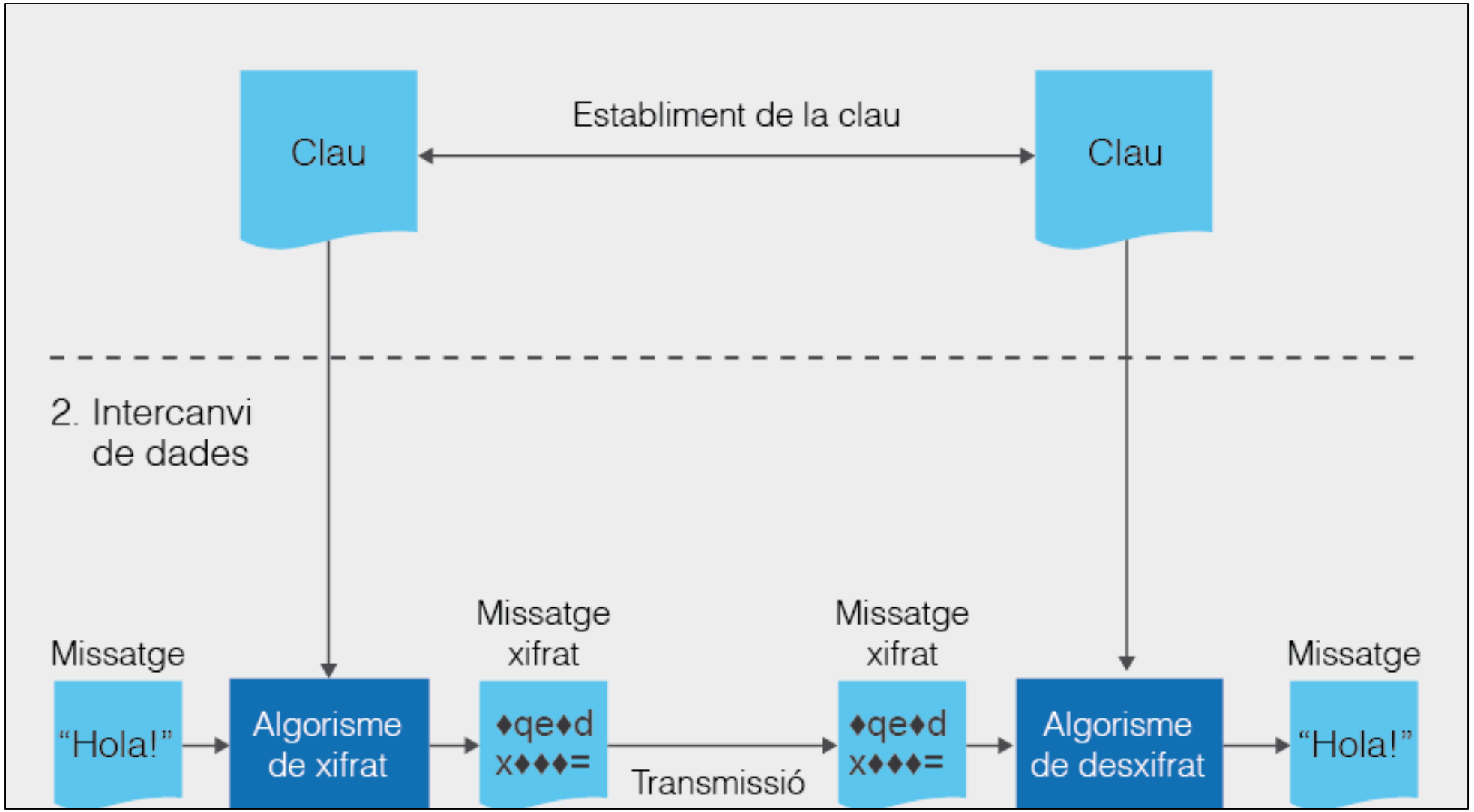
• Después, tota la comunicación está cifrada de forma simétrica.

• El cifrado de los mensajes es simétrico, y por tanto:

- Más rápido
- Sin limitaciones cuanto a la mida del mensaje.



2. MODELO DE CLAVE SECRETA



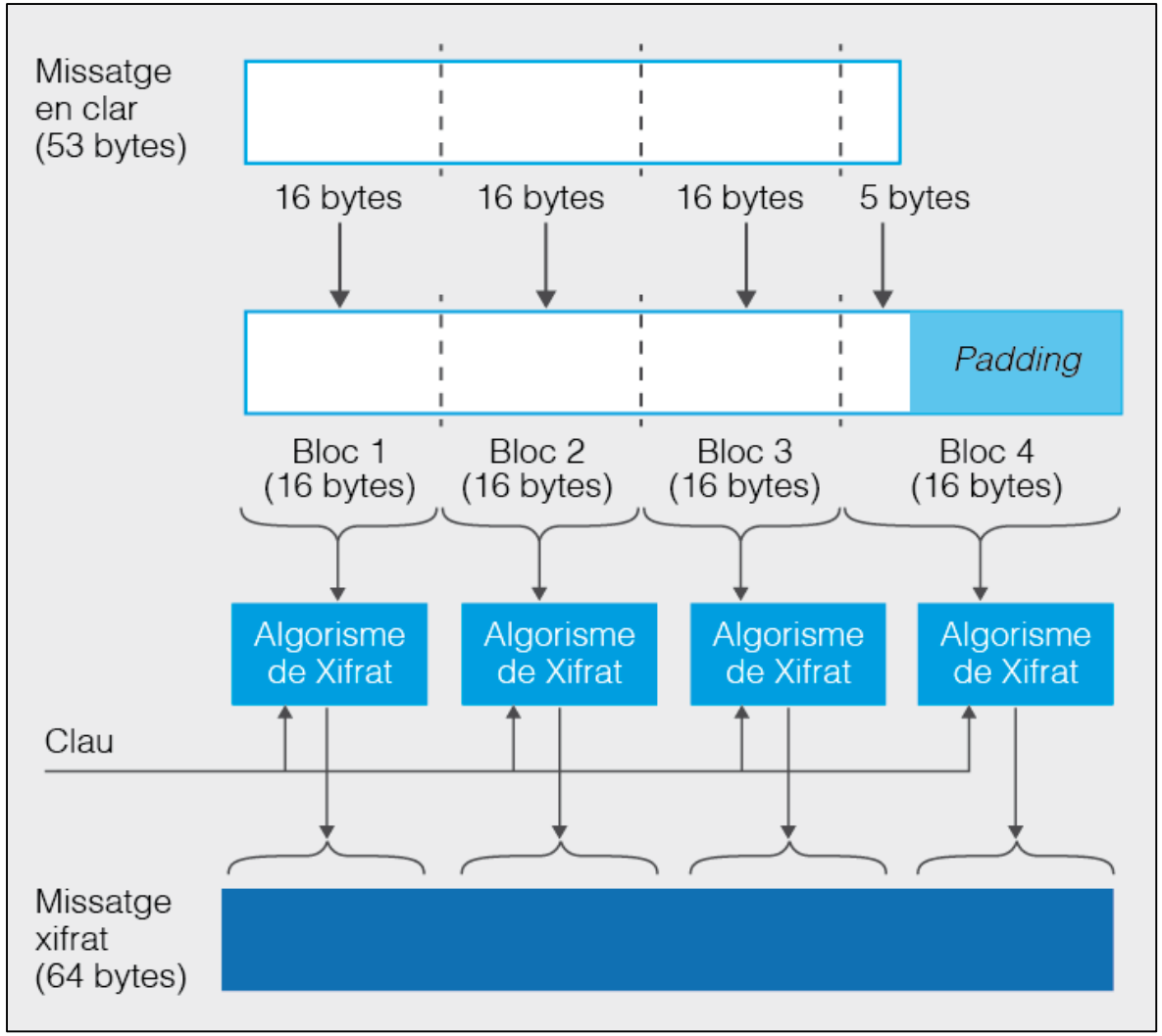
2. MODELO DE CLAVE SECRETA

Algoritmo DES de cifrado simétrico

- Estándar por el Gobierno de los EEUU para comunicaciones desde 1976
- Cifra un texto de una longitud fija en otro texto cifrado de la misma longitud. DES es un cifrador de bloque con ancho de bloque de 64 bits.
- La clave Es de 56 bits. Se suele expresar como un nombre de 64 bits, pero el último bit de cada byte es de paridad, y se saca antes de aplicar el algoritmo.
- El algoritmo se limita a aplicar substituciones (cambios de unos valores por otros) y permutaciones (cambios en la posición que ocupan los bits).
- Las operaciones están escogidas para que el mismo algoritmo sirva para el cifrado y el descifrado, la diferencia es que la clave ha de usarse en orden inverso

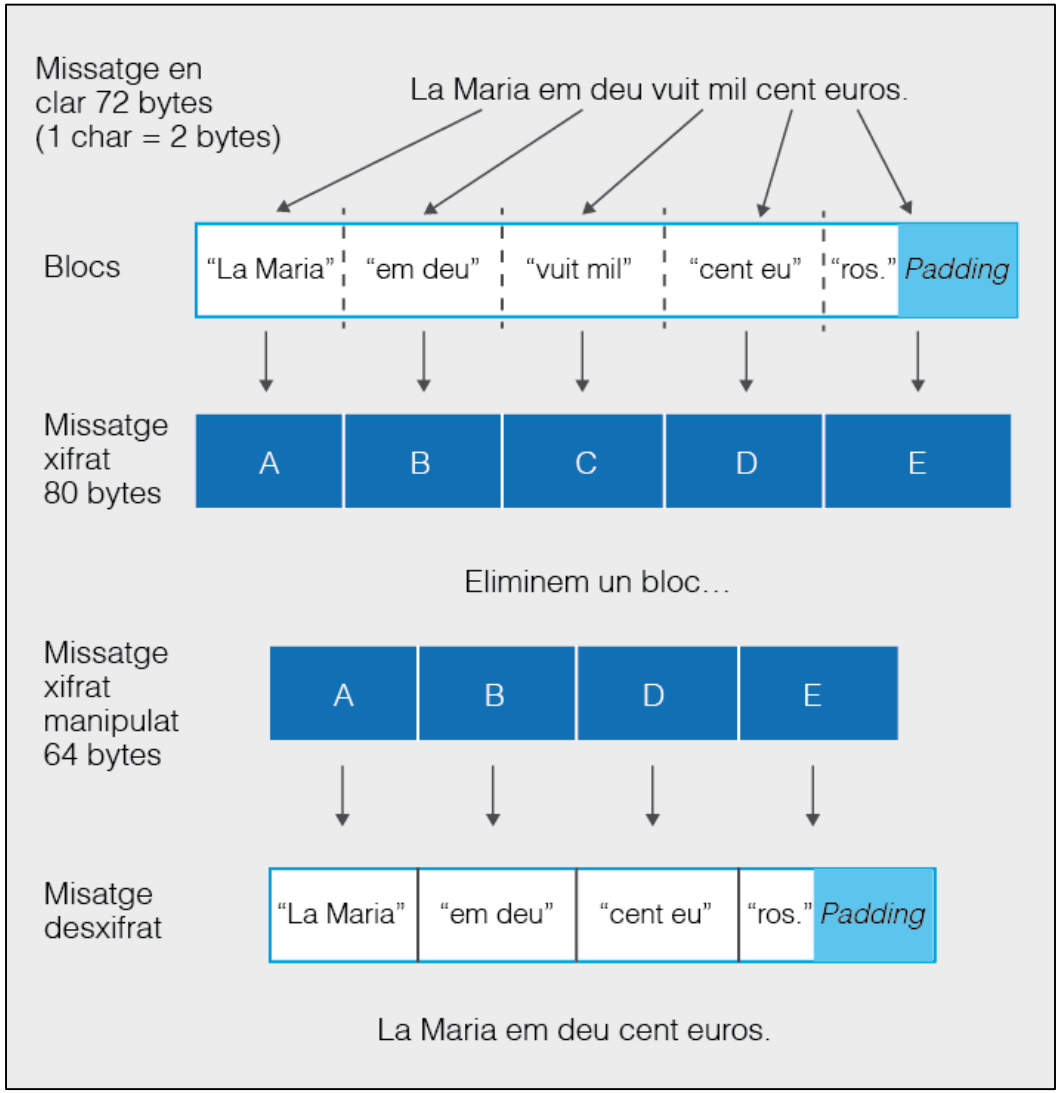
2. MODELO DE CLAVE SECRETA

Algoritmo DES en modo ECB



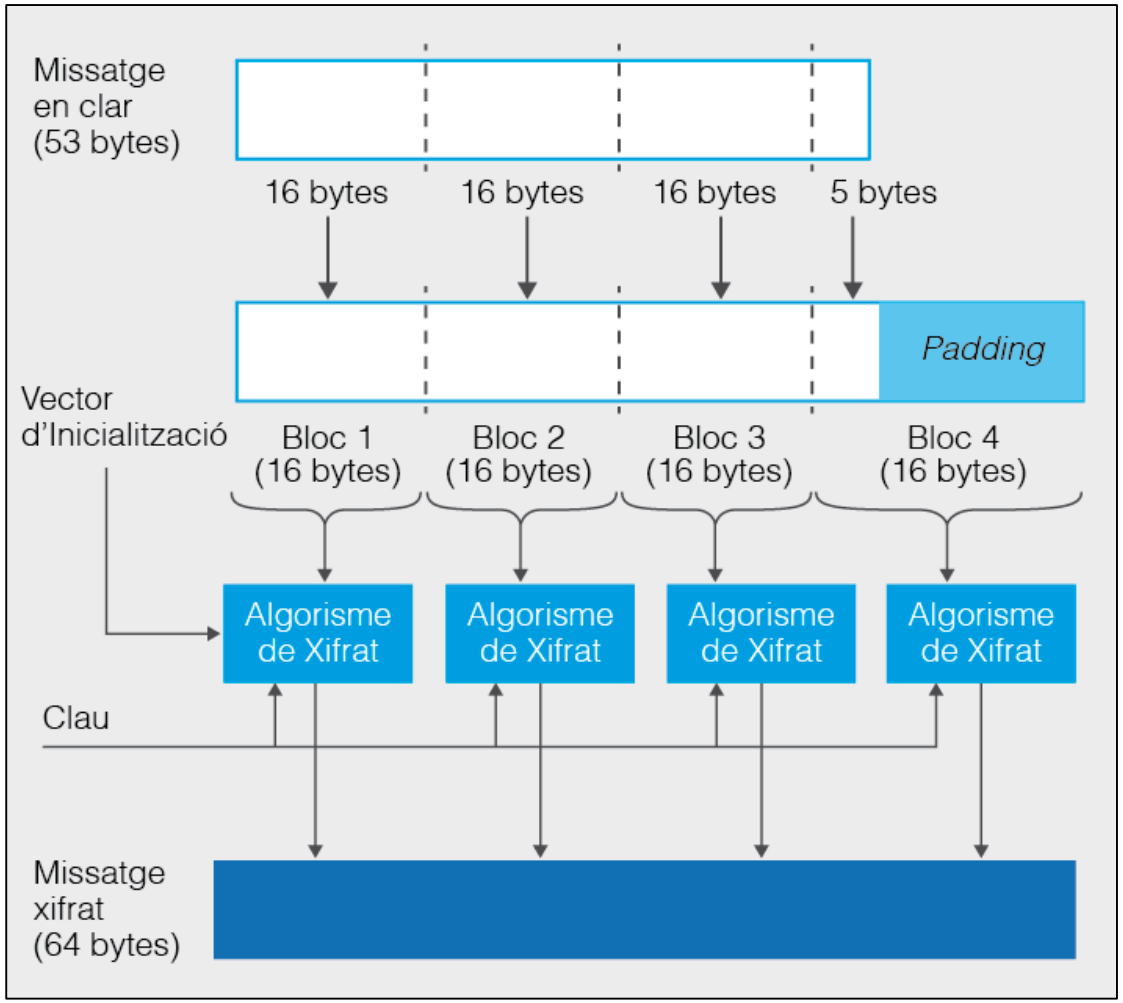
2. MODELO DE CLAVE SECRETA

Problemas con ECB



2. MODELO DE CLAVE SECRETA

Algoritmo DES en Modo CBC



2. MODELO DE CLAVE SECRETA

Otros algoritmos de clave secreta

AES

- Sustituto del DES
- Estándar de cifrado por el Gobierno de EEUU desde el año 2000. Su desarrollo se llevó a cabo de forma publica y abierta
- Sistema de cifrado por bloques
- Maneja longitudes de clave y de bloque variables, entre 128 y 256 bits. Rápido y fácil de implementar.

Blowfish:

- Permite usar una clave variable de hasta 448 bits, y que está optimizado para microprocesadores de 32 bits i de 64 bits.

2. MODELO DE CLAVE SECRETA

- **Claves simétricas basadas en contraseña**
  - Una clave simétrica ha de poder ser intercambiada.
  - Una manera de facilitar la gestión de claves criptográficas es generarlas a partir de una contraseña legible.

3. CRIPTOGRAFIA EN JAVA

- Los motores criptográficos de Java proporcionan mecanismos para firmas digitales, resúmenes de mensajes, etc. Estos motores están implementados para proveedores de seguridad (java.security.Provider), que se pueden visualizar mediante java.security.Security.getProviders(). Cada motor (java.security.Provider.Service) tiene un tipo y un algoritmo.
- El lenguaje Java proporciona herramientas para el manejo de claves y mecanismos de cifrado.
- Los componentes básicos para el cifrado en Java son:
  - La interfaz *Key*.
  - La Interfaz *KeySpec*.
  - La clase *Cipher*.

3. CRIPTOGRAFIA EN JAVA

Interficie Key

La interfície *java.security.Key* és la base de tots els tipus de claus de Java, i actua com un tipus opac, ja que no dóna informació sobre els seus atributs.

La interfície Key únicament té tres mètodes. En conseqüència tot tipus de clau ha d'implementar aquests mètodes:

- String <Key> getAlgorithm() → Retorna el nom de l'algorisme pel qual serveix la clau (p.ex. "DES", "Blowfish", "DSA",…).
- byte[] <Key> getEncoded() → Forma codificada. Retorna una tira de bytes amb la clau binària.
- String <Key> getFormat() → Indica el format usat per a la clau retornada pel mètode anterior (p.ex. "X.509" o "PKCS#8"). Aquest format és necessari per poder interpretar la tira de bytes.

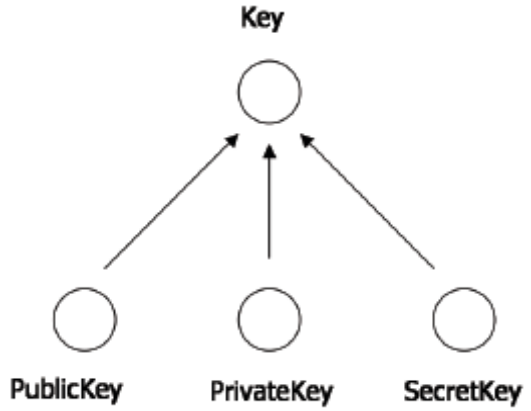
3. CRIPTOGRAFIA EN JAVA

PublicKey, PrivateKey, SecretKey

Tanto SecretKey, como PublicKey i PrivateKey, son subclases de java.security.Key.

No añaden más mètodos, sinó que solo se crean para clasificar los diferentes tipos de claves.

- Les interfícies PublicKey i PrivateKey s'utilitzen en criptografia asimètrica y vienen en el paquete java.security.*
- La interfície SecretKey s'utilitza en criptografia simètrica, y está en el paquete javax.crypto.*



3. CRIPTOGRAFIA EN JAVA

- **Generadores de claves:** Se utilizan para crear objetos de clases que implementan la interfaz *Key*, es decir, para crear claves nuevas.
- **Factorías de claves:** Se emplean para obtener versiones transparentes (*KeySpec: es pot accedir als seus paràmetres*) a partir de de claves opacas (*Key*) y viceversa.

3. CRIPTOGRAFIA EN JAVA

Generación de claves de tipo simétrico (KeyGenerator) :

```
KeyGenerator keyGen = KeyGenerator.getInstance(algorithm);
keyGen.init(size);
SecretKey secretKey = keyGen.generateKey();
```

→ Algoritmos simétricos típicos: DES (56 bits) o AES (128, 192, 256 bits).

Generación de claves de tipo asimétrico (KeyPairGenerator)

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance(algorithm);
kpg.initialize(size);
KeyPair kp = kpg.generateKeyPair();
PublicKey publicKey = kp.getPublic();
PrivateKey privateKey = kp.getPrivate();
```

→ Algoritmo más habitual es RSA (1024, 2048 bits).

3. CRIPTOGRAFIA EN JAVA

- Tanto SecretKey, com PublicKey i PrivateKey tienen un mètode getEncoded(): la clave en format binari.
- Aquest format pot convertir-se, per exemple, a un format fàcil de gestionar amb cadenes. El més conegut és java.util.Base64, que permet intercanviar el format d'una clau Key key:

```
String stringKey = Base64.getEncoder().encode(key.getEncoded());
byte[] encoded = Base64.getDecoder().decode(stringKey);
```

3. CRIPTOGRAFIA EN JAVA

- Generación clave simétrica automática:

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecretKey key = keygen.generateKey();
```

- Generación clave simétrica por teclado:

```
System.out.print("Contrasenya: ");
String password = teclat.nextLine();
byte[] data = password.getBytes("UTF-8");
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] hash = md.digest(data);
byte[] key = Arrays.copyOf(hash, 192/8);
SecretKey clau = new SecretKeySpec(key, "AES");
```

3. CRIPTOGRAFIA EN JAVA

Clase Cipher

Para poder cifrar y descifrar la información, necesitamos un objeto del tipo javax.crypto.Cipher. Se crean especificando el algoritmo que se desea utilizar. Despues se pueden configurar para realizar tanto operaciones de cifrado como descifrado, indicando en el proceso la clave necesaria.

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

El formato del parámetro es algoritmo/modo/padding. Modo y padding son opcionales: si no se indican, se utiliza el modo y padding por defecto.

- Padding: Es una técnica que consiste a añadir datos al comienzo, medio o final de un mensaje antes de ser cifrado. Esto se hace porque los algoritmos están diseñados para tener datos de un tamaño concreto.
- Modo: define como se hace la correspondencia entre los bloques de entrada (en plano) y los de salida (cifrados). Lo más sencillo es el modo ECB: un bloque de entrada va a uno de salida.

3. CRIPTOGRAFIA EN JAVA

Después, hemos de inicializar el objeto utilizando el modo (Cipher.ENCRYPT_MODE o Cipher.DECRYPT_MODE) y la clave de cifrado:

```
cipher.init(Cipher.ENCRYPT_MODE, key); // xifrat
cipher.init(Cipher.DECRYPT_MODE, key); // desxifrat
```

Finalmente, realizamos el cifrado o descifrado:

```
byte[] bytesOriginal = textOriginal.getBytes("UTF-8"); // bytes a encriptar
byte[] bytesXifrat = cipher.doFinal(bytesOriginal);
```

El descifrado podria ser:

```
byte[] bytesDesxifrat = cipher.doFinal(bytesXifrat);
```

Si el contenido a descifrar es una parte del array:

```
byte[] bytesDesxifrat = cipher.doFinal(bytesXifrat, inicio, longitud);
```

3. CRIPTOGRAFIA EN JAVA

Cifrado de streams

No tiene sentido cifrar grandes cantidades de datos con métodos de bloque. Se puede utilizar el cifrado de streams, que siempre es simétrico.

Uso de CipherOutputStream: queremos abrir un arxivo y cifrarlo o descifrarlo:

```
FileInputStream in = new FileInputStream(inputFilename);
FileOutputStream fileOut = new FileOutputStream(outputFilename);
CipherOutputStream out = new CipherOutputStream(fileOut, cipher);
```

Se tendría que copiar el stream in en out.

Uso de CipherInputStream: el stream de salida podria ser un FileOutputStream:

```
FileInputStream fileIn = new FileInputStream(inputFilename);
CipherInputStream in = new CipherInputStream(fileIn, cipher);
FileOutputStream fileOut = new FileOutputStream(outputFilename);
```

3. CRIPTOGRAFIA EN JAVA

Ejemplo cifrado DES

```
import javax.crypto.*;

public class CipherExample {

    public static void main(String[] args) {
        try {
            KeyGenerator keygen = KeyGenerator.getInstance("DES");
            SecretKey key = keygen.generateKey();
            Cipher desCipher = Cipher.getInstance("DES");
            desCipher.init(Cipher.ENCRYPT_MODE, key);
            String mensaje = "Mensaje de prueba";
            String mensajeCifrado = new String(desCipher.doFinal(mensaje.getBytes()));
            System.out.println(mensaje);
            System.out.print(mensajeCifrado);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. CRIPTOGRAFIA EN JAVA

Ejemplo cifrado AES

```
import javax.crypto.*;

public class CipherExample2 {
    public static void main(String[] args) {
    try {
            KeyGenerator keygen = KeyGenerator.getInstance("AES");
            SecretKey key = keygen.generateKey();
            Cipher aesCipher = Cipher.getInstance("AES");
            aesCipher.init(Cipher.ENCRYPT_MODE, key);
            String mensaje = "Mensaje que se cifrará con AES";
            System.out.println(mensaje);
            String mensajeCifrado = new String(aesCipher.doFinal(mensaje.getBytes()));
            System.out.println(mensajeCifrado);
            aesCipher.init(Cipher.DECRYPT_MODE, key);
            String mensajeDescifrado = new String(aesCipher.doFinal(mensajeCifrado.getBytes()));
            System.out.print(mensajeDescifrado);
    } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. CRIPTOGRAFIA EN JAVA

Encriptación simétrica

```
public static void main(String[] args) {
    try {
        // Genera la clave
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        kgen.init(128);
        SecretKey clave = kgen.generateKey();
        // Inicializa el cifrador
        Cipher encriptador = Cipher.getInstance("AES/ECB/PKCS5Padding");
        encriptador.init(Cipher.ENCRYPT_MODE, clave);
        // Cifra el mensaje e imprime el cifrado en hexadecimal
        byte[] text_clar = "Ola k ase".getBytes("UTF-8");
        byte[] text_xifrat = encriptador.doFinal(text_clar);
        System.out.println("Texto cifrado: "
                           + DatatypeConverter.printHexBinary(text_xifrat));
        // Inicializa el descifrador
        Cipher desencriptador = Cipher.getInstance("AES/ECB/PKCS5Padding");
        desencriptador.init(Cipher.DECRYPT_MODE, clave);
        // Descifra el mensaje e imprime el texto descifrado
        text_clar = desencriptador.doFinal(text_xifrat);
        System.out.println("Texto original: " + new String(text_clar));
    } catch (NoSuchAlgorithmException e) {
```

PRACTICA 3

Encriptación de ficheros con clave simétrica

Crea una aplicación Java que proteja un fichero ficheros mediante una contraseña, usando algoritmos criptográficos de cifrado. Utiliza el algoritmo AES en modo de trabajo ECB o CBC y con método de relleno PKCS5Padding.

El funcionamiento es el siguiente:

1) La aplicación primer pide el nombre del fichero
2) Si existe un fichero con este nombre, entonces pide una contraseña.
3) Se genera una clave AES de 192 bits a partir de la contraseña y se usa para cifrar los datos del fichero.
4) Si el nombre del fichero introducido no acaba en .aes, la aplicación supone que se quiere cifrar el fichero con la contraseña introducida. Si el nombre del fichero introducido acaba en .aes, entonces interpreta que se quiere descifrar este fichero. Se debe de inicializar el cifrador según su modo correspondiente: DECRYPT o ENCRYPT.

PRACTICA 3

5) Crea un fichero nuevo con el nombre y extensión antiguos, pero acabado en .aes si es un proceso de encriptación, o un fichero nuevo con el mismo nombre y extensión, pero quitando .aes si es un proceso de desencriptado. En este sentido, que un nombre de fichero acabe o no en .aes es el indicador que permite a la aplicación saber si ha de cifrar o descifrar.
6) Se leen todos los datos del fichero antiguo, y se encriptan/desencriptan de golpe
7) Se van copiando los nuevos datos encriptados/desencriptados en el nuevo fichero. La API a utilizar es read y write.

```
// Es llegeixen les dades i es van escrivint al nou fitxer xifrades/desxifrades
byte[] dades = new byte[1024];
FileOutputStream out = new FileOutputStream(nomFitxerNou);
FileInputStream in = new FileInputStream(nomFitxer);
```

7) El fichero inicial se borra del sistema, sobreescribiéndolo primero con ceros (opcional)

4. ALGORITMOS DE HASH

Un algorisme de *hash*, o resum (*digest*, en anglès), és una transformació criptogràfica que es pot aplicar a un conjunt de dades de manera que compleix sempre un seguit de condicions.

- Donada una mateixa entrada, sempre resulti en exactament la mateixa sortida i que dues sortides diferents vinguin sempre d'entrades diferents.
- No ha de ser reversible. A partir només d'un resultat, ha de ser pràcticament impossible endevinar quina ha estat l'entrada original que l'ha generat.

Algoritmos de *hash*

- SHA-1 (*Secure Hash Algoritm*, o Algorisme Segur de *Hash*). Aquest, a partir d'una entrada de dades de mida arbitrària, genera un resultat de 160 bits.
- SHA-256. Tal com diu el seu nom, aquest genera 256 bits de sortida.
- SHA-3, una nova versió encara millor i més segura.

4. ALGORITMOS DE HASH

Programació amb funcions de Hash

- En Java se emplea la clase *MessageDigest* (*java.security.MessageDigest*).
- Para generar un hash se siguen los siguientes pasos:
  - Obtindre una instància de la classe *MessageDigest*.
  - Introducció de dades en la instància.
  - Còmput del resum.

4. ALGORITMOS DE HASH

Ejemplo de cálculo Hash

```
public class MessageDigestExample {

    public static void main(String[] args) {
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            byte[] c1 = "Primera cadena".getBytes();
            byte[] c2 = "Segunda cadena".getBytes();
            byte[] c3 = "Tercera cadena".getBytes();
            md.update(c1); md.update(c2); md.update(c3);
            byte[] resumen = md.digest();
            md.reset();
            byte[] c4 = "Cuarta cadena".getBytes();
            md.update(c4);
            resumen = md.digest();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4. ALGORITMOS DE HASH

Encriptación simetrica

```
public static void main(String[] args) {
    byte[] IV = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
                 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
    try {
        // Genera la clave a partir del hash SHA-256 de la contraseña "Pepito"
        // Solo se queda con los primeros 128 bits (7 bytes) del hash
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] clau_hash = md.digest( "Pepito".getBytes("UTF-8") );
        clau_hash = Arrays.copyOf(clau_hash, 128/8);
        SecretKey clave = new SecretKeySpec(clau_hash, "AES");

        // Inicializa el cifrador
        Cipher encriptador = Cipher.getInstance("AES/CBC/PKCS5Padding");
        encriptador.init(Cipher.ENCRYPT_MODE, clave, new IvParameterSpec(IV));
        // Cifra el mensaje e imprime el cifrado en hexadecimal
        byte[] text_clar = "Ola k ase".getBytes("UTF-8");
        byte[] text_xifrat = encriptador.doFinal(text_clar);
        System.out.println("Texto cifrado: "
                           + DatatypeConverter.printHexBinary(text_xifrat));
        // Inicializa el descifrador
        Cipher desencriptador = Cipher.getInstance("AES/CBC/PKCS5Padding");
        desencriptador.init(Cipher.DECRYPT_MODE, clave, new IvParameterSpec(IV));
        // Descifra el mensaje e imprime el texto descifrado
        text_clar = desencriptador.doFinal(text_xifrat);
        System.out.println("Texto original: " + new String(text_clar));
```

PRACTICA 4

Rompiendo un mensaje cifrado.

- Se ha interceptado un mensaje de texto cifrado. La codificación en binario del mensaje interceptado, escrita como secuencia de valores hexadecimales, es la siguiente:

```
byte[] enc = {  (byte)0x2d, (byte)0xb3, (byte)0x14, (byte)0x90,
                (byte)0xbd, (byte)0x0a, (byte)0xb1, (byte)0x7c,
                (byte)0x45, (byte)0x2f, (byte)0x40, (byte)0x67,
                (byte)0xc0, (byte)0x81, (byte)0xe2, (byte)0x79,
                (byte)0x4d, (byte)0xa8, (byte)0x54, (byte)0x86,
                (byte)0x3a, (byte)0xf7, (byte)0x0e, (byte)0x64,
                (byte)0x97, (byte)0xc1, (byte)0x78, (byte)0xef,
                (byte)0x84, (byte)0xdd, (byte)0x9a, (byte)0xf1};
```

- Únicamente se sabe que para generar se ha usado una clave AES de 192 bits basada en una contraseña generada mediante el algoritmo SHA-256.
- Se desconoce la clave. Sólo se sabe que es una combinación de cuatro números (por ejemple 4587).
- Haz un programa que nos permita averiguar el contenido del mensaje original.

5. MODELO DE CLAVE PUBLICA

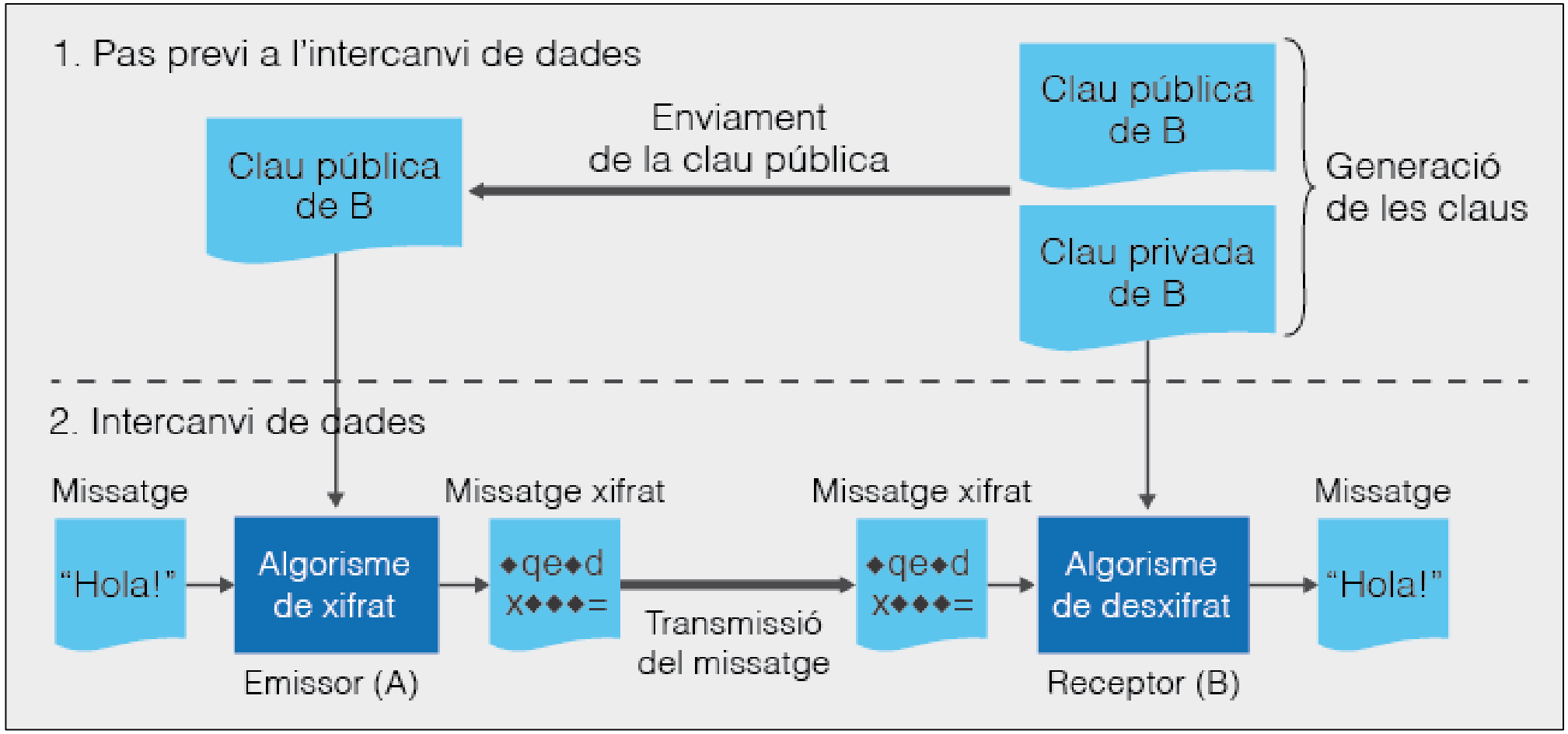
Diseñado para evitar el problema de la distribución de las claves entre el emisor y el receptor.

Dos claves diferentes:

- Clave privada Kc: que solo lo conoce el usuario.
- Clave publica Kp: publicada por todo el mundo que lo desee.
- Relacionadas entre sí por una función de sentido único. Debe ser muy difícil calcular una clave a partir de la otra
- Sistema asimétrico, pues se emplean claves diferentes para encriptar y desencriptar la información.
- Si alguien desea enviar un mensaje, buscará la clave pública de aquel al que desea enviárselo, y lo cifrará con dicha clave.
- La única forma de desencriptarlo es utilizando la clave privada del receptor

5. MODELO DE CLAVE PUBLICA

Modelo de clave pública



5. MODELO DE CLAVE PUBLICA

Algoritmo RSA

- Debe su nombre a sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman, del MIT.
- Algoritmo asimétrico de cifrado por bloques, que utiliza:
  - Una clave publica, conocida por todos.
  - Una clave otra privada, guardada en secreto por su propietario.
  - La relación de claves se basa en el producto de dos números primos muy grandes elegidos al azar. No hay maneras rápidas conocidas de factorizar un numero grande en sus factores primos.

5. MODELO DE CLAVE PUBLICA

Ejemplo algoritmo RSA

```
public static void main(String[] args) {
    try {
        KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");
        KeyPair keypair = keygen.generateKeyPair();
        Cipher rsaCipher = Cipher.getInstance("RSA");
        rsaCipher.init(Cipher.ENCRYPT_MODE, keypair.getPrivate());
        String mensaje = "Mensaje de prueba del cifrado asimetrico";
        String mensajeCifrado = new String(rsaCipher.doFinal(mensaje.getBytes()));
        rsaCipher.init(Cipher.DECRYPT_MODE, keypair.getPublic());
        String mensajeDescifrado = new String(rsaCipher
                                .doFinal(mensajeCifrado.getBytes()));
    } catch (Exception e) {
        e.printStackTrace();
    }
```

5. MODELO DE CLAVE PUBLICA

FUNCIONAMIENTO CIFRADO ASIMETRICO

El cifrado simétrico es mucho más rápido que el asimétrico y permite textos de mayor longitud. El problema es como enviar la clave simétrica al destinatario del mensaje para que lo descifre. La solución es enviarle el mensaje cifrado con la clave simétrica, y a continuación enviar la clave simétrica "envuelta“ o "cifrada con la clave pública del destinatario del mensaje, para que sólo el destinatario con su clave privada pueda obtener la clave simétrica para descifrar el mensaje

5. MODELO DE CLAVE PUBLICA

FUNCIONAMIENTO CIFRADO ASIMETRICO

- Se genera una clave simétrica AES de 128 bits para cifrar el mensaje
- Obtenemos las claves publicas y privadas mediante la clase KeyPairGenerator que genera un par de claves asimétricas RSA de 2048 bits.

```
try {
    // Obtengo clave simétrica y preparo para cifrar
    KeyGenerator kgen = KeyGenerator.getInstance("AES");
    kgen.init(128);
    SecretKey clave = kgen.generateKey();
    Cipher encriptador = Cipher.getInstance("AES/ECB/PKCS5Padding");
    encriptador.init(Cipher.ENCRYPT_MODE, clave);

    // Ciframos el mensaje
    byte[] text_clar = "Ola k ase".getBytes("UTF-8");
    byte[] text_xifrat = encriptador.doFinal(text_clar);
    System.out.println("Texto cifrado: " + DatatypeConverter.printHexBinary(text_xifrat));

    // Obtengo clave pública y privada y preparo para envolver
    KeyPairGenerator kgen2 = KeyPairGenerator.getInstance("RSA");
    kgen2.initialize(2048);
    KeyPair claves = kgen2.genKeyPair();
    encriptador = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    encriptador.init(Cipher.WRAP_MODE, claves.getPublic());
```

5. MODELO DE CLAVE PUBLICA

FUNCIONAMIENTO CIFRADO ASIMETRICO

- Una vez cifrado el mensaje con cifrado simétrico se envía el mensaje y la clave para descifrarlo. Pero antes se envuelve (cifra) la clave simétrica con la clave pública.
- En destino se descifra la clave simétrica con la clave la privada y descifra el mensaje con la clave simétrica obtenida

```
        // Envuelvo la clave
        byte[] clau_xifrada = encriptador.wrap(clave);
        System.out.println("Clave envuelta: " + DatatypeConverter.printHexBinary(clau_xifrada));

        // Desencripto la clave simétrica
        encriptador = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        encriptador.init(Cipher.UNWRAP_MODE, claves.getPrivate());
        clave = (SecretKey) (encriptador.unwrap(clau_xifrada, "AES", Cipher.SECRET_KEY));

        // Desencripto mensaje con la clave simétrica
        encriptador = Cipher.getInstance("AES/ECB/PKCS5Padding");
        encriptador.init(Cipher.DECRYPT_MODE, clave);
        text_clar = encriptador.doFinal(text_xifrat);
        System.out.println("Texto original: " + new String(text_clar));
    } catch (NoSuchAlgorithmException e) {
```

PRACTICA 5

1) Genera el par de claves pública y privada, mediante una instancia RSA. Inicializa las llaves con 2048 bits

2) Inicializa el cifrador con una instancia *RSA/ECB/PKCS1Padding.* Prepara el encriptador con la clave publica.

3) Cifra el mensaje e imprime el cifrado en hexadecimal (usa la función *DatatypeConverter.printHexBinary* new String(text_desxifrat, StandardCharsets.UTF_8);

4) Inicializa el descifrador. Usa la misma instancia anterior *RSA/ECB/PKCS1Padding, pero ahora utiliza la clave privada*

5) Descifra el mensaje e imprime el texto descifrado.

6. FIRMA DIGITAL

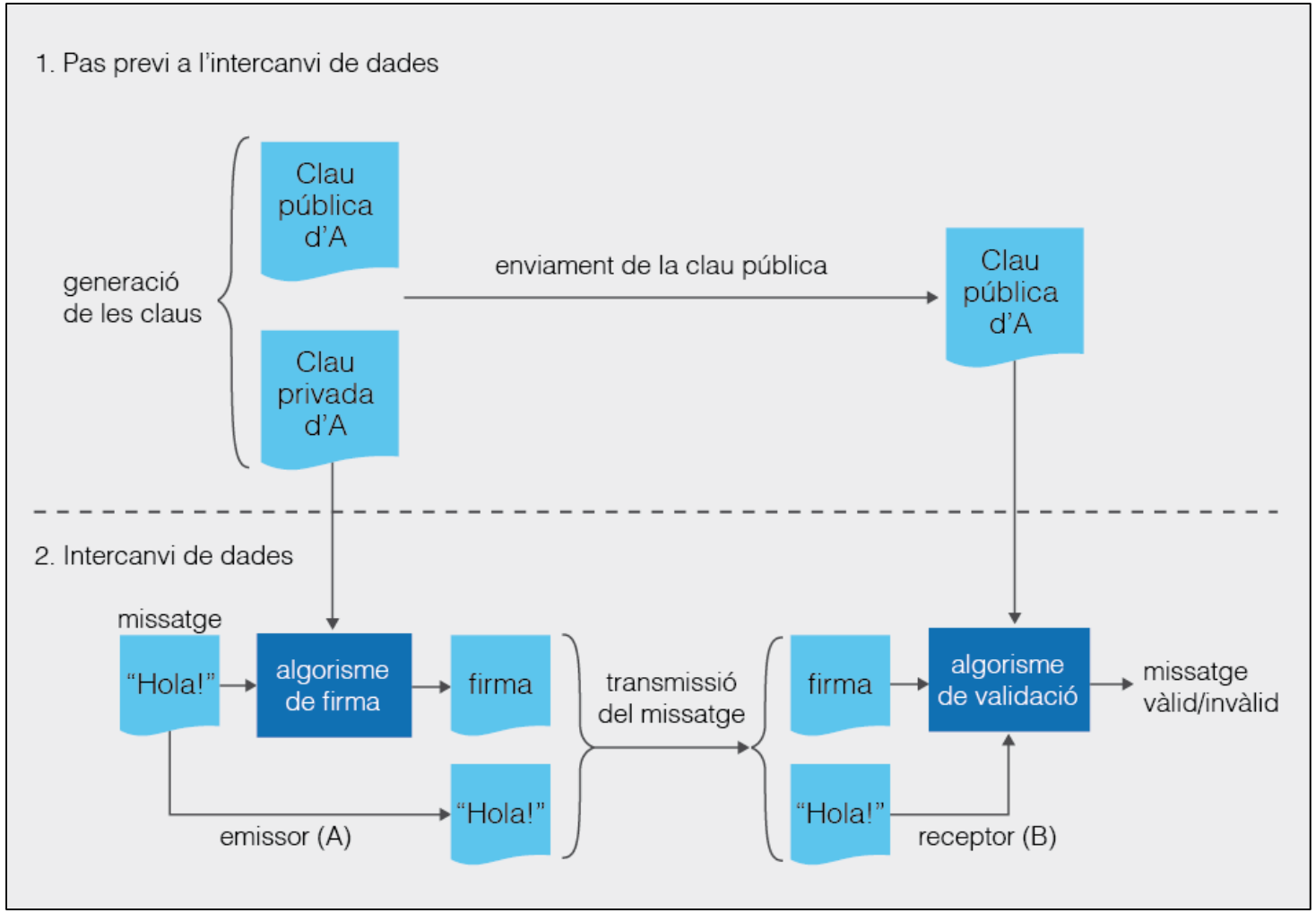
La firma digital asocia la identidad del emisor al mensaje que se transmite a la vez que comprueba la integridad del mensaje, mediante:

1.- Aplicación de una función *hash*.

2.- Empleo de un sistema de clave pública para codificar mediante la clave privada el resumen obtenido anteriormente para autenticar el mensaje.

- Una firma digital es un message digest cifrado con una clave privada de un par de claves pública / privada.
- Se envían al destinatario dos cosas: el mensaje, y la firma.
- El destinatario comprueba el mensaje recibido, con la firma y la clave pública del emisor. Cualquiera que esté en posesión de la clave pública puede verificar la firma digital.
- La clase Java Signature (java.security.Signature) puede crear una firma digital para datos binarios.

6. FIRMA DIGITAL



6. FIRMA DIGITAL

Ejemplo de firma Digital

```
public static void main(String[] args) {
    try {
        // Obtengo clave publica y privada
        KeyPairGenerator kgen = KeyPairGenerator.getInstance("RSA");
        kgen.initialize(2048);
        KeyPair claves = kgen.genKeyPair();

        // Inicializa la firma con la clave privada
        Signature sellador = Signature.getInstance("SHA1withRSA");
        sellador.initSign(claves.getPrivate());

        // Sella el mensaje e imprime la firma en hexadecimal
        byte[] text = "Ola k ase".getBytes("UTF-8");
        sellador.update(text);
        byte[] signatura = sellador.sign();
        System.out.println("Firma digital: " + DatatypeConverter.printHexBinary(signatura));

        // Inicializa el desellador
        Signature desellador = Signature.getInstance("SHA1withRSA");
        desellador.initVerify(claves.getPublic());

        // Comprueba que la firma realmente autenticaba el mensage y el emisor
        // imprime si el mensaje y origen eran autenticos
        desellador.update(text);
        System.out.println("¿Correcto? " + desellador.verify(signatura));

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

PRACTICA 6

1) Genera el par de claves mediante el algoritmo DSA. Inicializa con 2048 bits
2) Crea una instancia de signature, usando método estático getInstance. Usa el algoritmo de firma digital SHA256WithDSA.
3) Inicializa la instancia de firma llamando a su método initSign(). Usa la clave privada de un par de claves privada/pública y una instancia de SecureRandom.
4) Creación de la firma digital, llamando al método update una o más veces, terminando con una llamada a sign ().
5) Verifica la firma digital creada por otra persona. Crea una signature2 con el mismo algoritmo SHA256WithDSA. Inicializa la signature2 en modo de verificación (en lugar del modo de firma) mediante initVerify, usando la clave pública del par de claves pública/privada como parámetro.
Mediante la funcion update desellaremos el mensaje inicial. Y ahora podremos verificar con la funcion verify si la firma enviada es correcta o no