UF1.NF1 – 3. Criptografia de clau pública

CFGS Desenvolupament d'Aplicacions Multiplataforma

Mòdul 9: Programació de serveis i processos

Apunts

Índex

Bibliografia

Java™ Cryptography Architecture (JCA) Reference Guide

http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

1. Introducció

El protocol pels algoritmes de clau pública és el següent:

1. B (el receptor) genera una clau privada aleatòria, i calcula la clau pública que correspon a aquesta clau privada.
2. B envia a A (l'emissor) la seva clau pública.
3. A xifra el missatge usant la clau pública de B, i la hi envia a B.
4. B utilitza la seva clau privada per desxifrar el missatge.

Amb això, la criptografia de clau pública evita el problema de posar-se "en secret" A i B d'acord en la clau a utilitzar. Operació que en la criptografia de clau secreta havia de fer-se "fora de protocol".

És important tenir en compte que un espia que escolta tota la comunicació per la xarxa obté la clau pública de B i el missatge xifrat, però no pot desxifrar-ho perquè no té la clau privada, ni la pot calcular a partir de la clau pública.

També és molt típic que les claus públiques dels usuaris d'un sistema criptogràfic es publiquin en una base de dades pública, i quan algú vulgui enviar-los un missatge, agafi la clau pública de la base de dades.

2. Sistemes criptogràfics híbrids

Per desgràcia, els sistemes criptogràfics de clau pública, per si sols, no poden usar-se per xifrar missatges, ja que tenen un problema:

- Els algoritmes de clau pública són molt més lents que els de clau secreta (de l'ordre de 1000 vegades més), a més d'augmentar la mida dels missatges xifrats, la qual cosa dificulta la seva transmissió per mitjans de comunicació relativament lents.

Per solucionar aquests problemes, en la pràctica s'utilitzen sistemes criptogràfics híbrids, que agafen el millor de tots dos mons (clau pública, clau secreta).

Aquests sistemes es basen en la utilització de l'anomenada **clau de sessió**, que no és més que una clau binària que s'usa només durant el que duri una comunicació.

El protocol dels sistemes criptogràfics híbrids és el següent:

1. B envia a A la seva clau pública.
2. A genera una clau de sessió aleatòria (binària), la xifra usant la clau pública de B, i la hi envia a B.
3. B desxifra el missatge enviat per A, usant la seva clau privada, i recupera la clau de sessió.
4. Tots dos es comuniquen mitjançant criptografia de clau secreta usant la clau de sessió.

En principi, no hi ha raó perquè A es pugui comunicar amb un sol interlocutor B. De fet, A pot generar missatges de broadcast xifrats, i només els poden llegir els destinataris legítims.
El protocol per aconseguir que A enviï un missatge de broadcast a B, C, D seria el següent:

1. A genera una clau de sessió aleatòria Ks, i xifra el missatge M usant Ks: C = Xifrar(Ks,M).
2. A xifra Ks amb les claus públiques de B, C, D: C1 = Xifar(K_B,Ks) C2= Xifrar(K_C,Ks) C3=Xifrar(K_D,Ks).
3. A fa un broadcast del missatge juntament amb les claus xifrades.
4. Els destinataris desxifren la clau de sessió amb la seva clau privada i llegeixen el missatge.

3. RSA

RSA és segurament l'algoritme de clau pública més conegut, més provat, i més senzill que es coneix. El seu nom es deu als tres inventors: Ron **R**ivest, Adi **S**hamir i Leonard **A**dleman.

Aquest algoritme es basa en la dificultat de factoritzar nombres grans, és a dir, treure els primers (àtoms) que componen el nombre. Es conjectura que trencar l'algoritme és equivalent a factoritzar un nombre gran. La mida de nombre a factoritzar sol ser de l'ordre de 2^{512}, 2^{1024}, 2^{2048} o 2^{4096}.
Un nombre és primer si no pot ser escrit com a producte d'altres nombres més petits majors que 1.

RSA es pot usar tant per a xifrat com per generar signatures digitals.

4. Mode i padding

Quan vam veure els xifradores de clau secreta, vam veure que usaven diverses maneres, dels quals el més senzill era ECB (Electronic Code Book), però tenia l'inconvenient que sempre que xifràvem un bloc obteníem el mateix text xifrat, amb el que era susceptible d'atacs.

No obstant això, els xifradors de clau asimètrica gairebé sempre usen ECB, ja que no són susceptibles als mateixos tipus d'atacs que els sistemes de clau secreta. Això es deu al fet que només solen transmetre un bloc, i aquest bloc s'usa per xifrar una clau de sessió.

Respecte al padding, vam veure que els xifradors simètrics solien usar l'algorisme PKCS #5. RSA, usa dues formes de padding diferents:

1. **PKCS #1** Que és la forma estàndard de*padding per RSA. Desgraciadament aquesta forma de padding té problemes de seguretat si no s'usa per xifrar dades totalment aleatòries. És a dir, aquest problema no es presenta quan xifrem claus de sessió, però per xifrar altres tipus d'informació s'usa el següent algoritme de padding.
2. **OAEP (Optimal Asymetric Encription Padding)**. Aquest mecanisme de padding és una millora sobre PKCS #1 que ens permet xifrar tot tipus de dades encara que segueixin patrons molt marcats, com per exemple les capçaleres. Desgraciadament és molt nou i no ha estat encara acabat per RSA Inc. S'espera que amb el temps acabi suplantant a *PCKS #1.

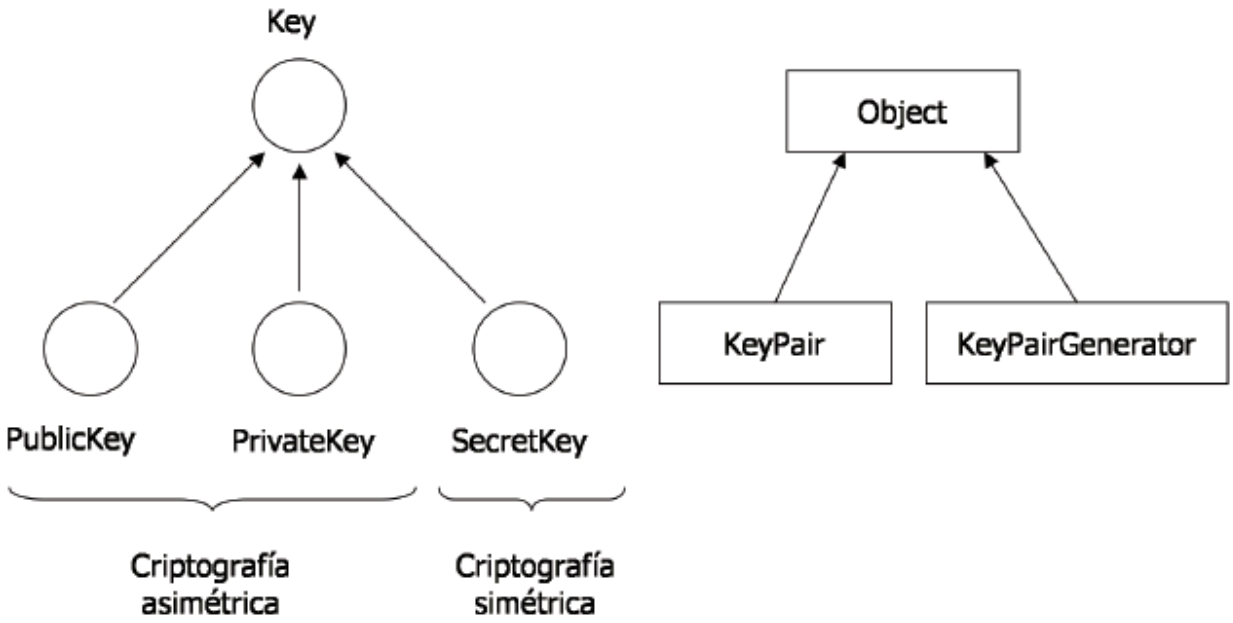
5. Criptografia de clau pública en Java

La majoria dels aspectes de les llibreries criptogràfiques en criptografia de clau pública són idèntics als de la criptografia de clau secreta:

- Bàsicament ens limitem a inicialitzar un objecte Cipher passant-li les claus, i li demanem que xifri o desxifri.

La principal diferència és que aquí hi ha dues claus: La clau privada i la clau pública.

Java té les següents classes per treballar amb aquestes claus:



KeyPair encapsula un grup format per una clau privada i una clau pública.

- Té el constructor:

  KeyPair(PublicKey publicKey, PrivateKey privateKey)

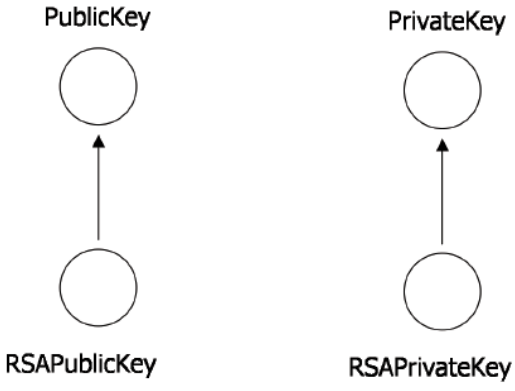
- I els mètodes:

  PublicKey <KeyPair> getPublic()

  PrivateKey <KeyPair> getPrivate()

PublicKey és una interfície derivada de Key que no afegeix més mètodes i simplement existeix per tipificar.

- En el paquet java.security.interfaces.* està la subinterfície RSAPublicKey que afegeix mètodes per accedir a informació pròpia de RSA.



PrivateKey és idèntica a PublicKey, excepte que representa una clau privada.

- java.security.interfaces.* també conté la classe RSAPrivateKey que té mètodes per accedir a informació pròpia de RSA.

KeyPairGenerator és equivalent a KeyGenerator en criptografia simètrica, és a dir, ens proporciona una forma de generar una parella de claus pública / privada.

- Per instanciar un objecte d'aquest tipus, com amb qualsevol altre servei, usem el mètode estàtic getInstance(), que en aquest cas té el prototip:

  static KeyPairGenerator <KeyPairGenerator> getInstance(String algorithm)

- I per generar un parell de claus aleatòries usem:

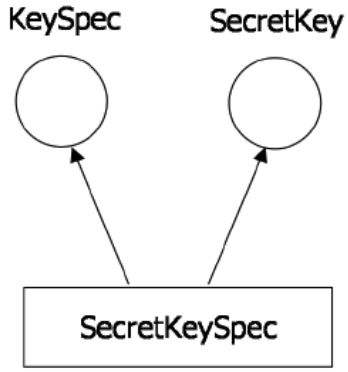
  KeyPair <KeyPairGenerator> getKeyPair()

6. Xifrat d'una clau de sessió

Com ja hem comentat, la principal utilitat de la criptografia asimètrica és xifrar claus de sessió.

Per xifrar una clau de sessió, s'han de fer els següents passos:

1. Crear una clau simètrica de 128 bits:
   KeyGenerator generador_clave_sesion = KeyGenerator.getInstance("Blowfish");
   generador_clave_sesion.init(128);
   Key clave_sesion = generador_clave_sesion.generateKey();
2. Generar un parell de claus RSA:
   KeyPairGenerator generador_clave_asimetrica = KeyPairGenerator.getInstance("RSA");
   generador_clave_asimetrica.initialize(1024);
   KeyPair par_claves = generador_clave_asimetrica.generateKeyPair();
3. Crear un xifrador RSA demanant ECB com a mode i PKCS#1 com padding:
   Cipher cifrador = Cipher.getInstance("RSA/ECB/PKCS1Padding");
   cifrador.init(Cipher.ENCRYPT_MODE, par_claves.getPublic());
4. Per xifrar la clau simètrica, hem de convertir-la en un array de bytes utilitzant getEncoded():
   byte[] bytes_clave_sesion = clave_sesion.getEncoded();
   byte[] clave_sesion_cifrada = cifrador.doFinal(bytes_clave_sesion);
5. Per desxifrar, podem reiniciar el xifrador al mode desxifrat i per reconstruir la clau de sessió, creem un objecte de tipus SecretKeySpec, que és un tipus transparent que implementa la interfície KeySpec i que té el constructor:
   SecretKeySpec(byte[] key, String algorithm)

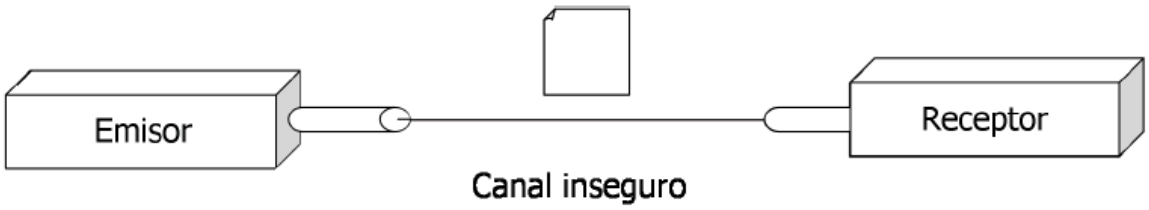


   Però a la vegada SecretKeySpec implementa SecretKey, amb el que pot actuar com a tipus opac sense la necessitat de passar l'objecte per un mètode que el transformi de tipus opac a tipus transparent com passava amb PBEKeySpec.
   És a dir, la forma de desxifrar la clau de sessió xifrada, seria:

```
cifrador.init(Cipher.DECRYPT_MODE,par_claves.getPrivate());
byte[] clave_descifrada = cifrador.doFinal(clave_sesion_cifrada);
Key clave_sesion_descifrada = new SecretKeySpec(clave_descifrada,"Blowfish");
if (clave_sesion.equals(clave_sesion_descifrada))
        System.out.println("Operacion correcta");
else
        System.out.println("Algo fue mal");
```

7. Xifrat d'un arxiu amb RSA

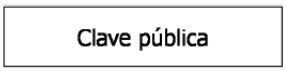
Anem a fer un programa que permet a l'emissor enviar fitxers a un receptor a través d'un canal insegur. Per a això l'emissor xifra els fitxers abans d'enviar-los usant la clau pública del receptor.



Aquest protocol també es podria usar per enviar missatges a través d'un socket.

Passos:

1. Crear un parell de claus privada / pública i guarda cadascuna en un fitxer.
   La clau privada s'ha de guardar sempre xifrada (per clau de sessió o password).



Fichero de la clave pública

Fichero de la clave privada

Fichero de datos encriptado

   Per poder guardar les claus en fitxer, les convertim en un array de bytes amb:

   byte[] <Key> getEncoded()

   Igual que en la criptografia simètrica, getEncoded() és el mètode que s'encarrega de codificar la claus en el seu corresponent format i retornar-nos-la en un array de bytes.

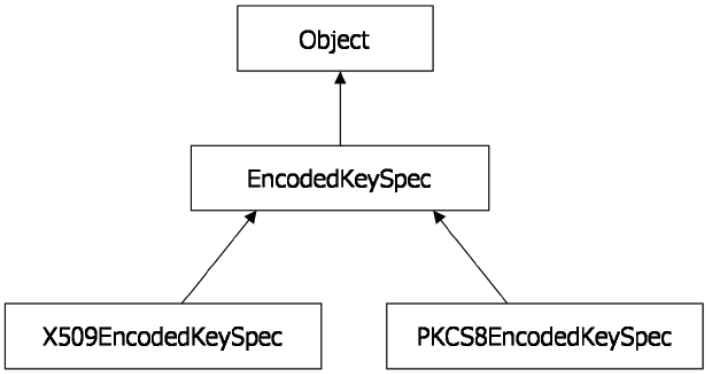
   Para obtenir les claus pública i privada necessitem utilitzar una classe de tipus transparent diferent per a cadascuna:

   - java.security.spec.X509EncodedKeySpec és la classe que ens permet descodificar una clau pública o certificat i per crear un objecte d'aquest tipus usem el constructor:

     X509EncodedKeySpec(byte[] array)

   - java.security.spec.PCKS8EncodedKeySpec ens permet descodificar claus privades, i per crear l'objecte usem el constructor

     PCKS8EncodedKeySpec(byte[] encoded)



Una vegada que tinguem un objecte d'un d'aquests dos tipus, per tornar a reconstruir la PublicKey i PrivateKey, l'hi hem de passar a un KeyFactory:

```
X509EncodedKeySpec clau_publica_spec = new X509EncodedKeySpec(buffer);
KeyFactory kf = KeyFactory.getInstance("RSA");
PublicKey clau_publica = kf.generatePublic(clau_publica_spec);
```

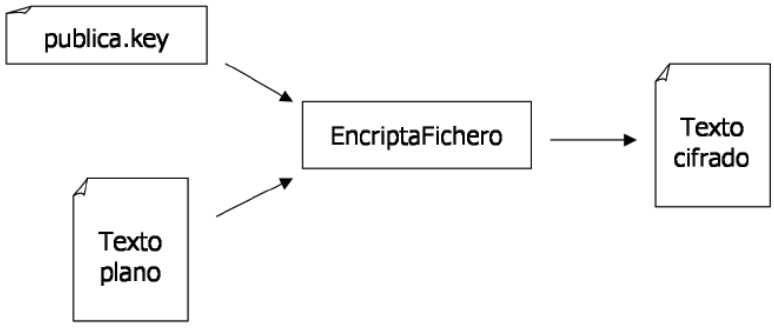
I anàlogament amb la clau privada:

```
PKCS8EncodedKeySpec clau_privada_spec = new PKCS8EncodedKeySpec(buffer);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey clau_privada = kf.generatePrivate(clau_privada_spec);
```

Buffer és la variable on s'emmagatzema les dades recuperades de l'arxiu que conté la clau (pública o privada) en forma d'array de bytes).

2. Crear el programa que xifra les dades abans d'enviar-les.

   L'emissor genera una clau de sessió amb la qual xifra el fitxer i després xifra la clau de sessió amb la clau pública. Aquest programa rep el fitxer de la clau pública i el fitxer a xifrar, i retorna el fitxer xifrat.



   Passos:

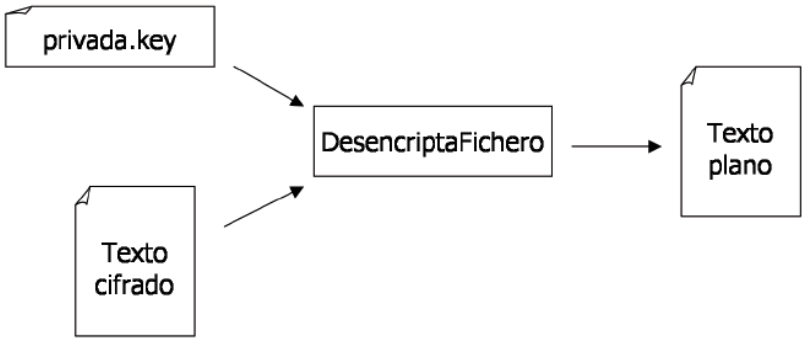
   a) Recuperar la clau pública del fitxer.
   b) Generar clau de sessió.
   c) Guardar clau de sessió xifrada amb la clau pública al fitxer xifrat.
   d) Generar IV aleatori.
   e) Guardar les dades xifrades al fitxer xifrat. Podem utilitzar la classe CipherOutputStream que permet escriure dades a un arxiu xifrades a partir d'una clau.

      ```
      public CipherOutputStream(OutputStream os,Cipher c)
      ```

      L'objecte Cipher ha d'estar inicialitzat en mode xifrat.

3. Crear el programa que desxifra les dades.

   El programa rep el fitxer de la clau privada i el fitxer a desxifrar, i retorna un fitxer amb el text pla desxifrat. Hem de recordar que la clau privada està protegida amb password, després aquest programa haurà de demanar un password per desxifrar-la.



   Passos:

   a) Recuperar la clau privada del fitxer.
   b) Recuperar la clau de sessió del fitxer.
   c) Desxifrar la clau de sessió amb la clau privada.
   d) Recuperar l'IV del fitxer.
   f) Desxifrar les dades i guardar-les al fitxer amb text pla. Podem utilitzar la classe CipherOutputStream que permet escriure dades a un arxiu desxifrades a partir d'una clau.

      public CipherOutputStream(OutputStream os,Cipher c)

      L'objecte Cipher ha d'estar inicialitzat en mode desxifrat.