Pràctica 4: Gestió de claus

Alumne: Eloi Cortiella Fortuño
Curs: 2n Desenvolupament d'Aplicacions Multiplataforma - 2025/2026
Professor: Marc Fuertes Díez

INDEX

Alumne: **Eloi Cortiella Fortuño**
Curs: 2n Desenvolupament d'Aplicacions Multiplataforma - 2025/2026
Professor: **Marc Fuertes Díez**

Tasca 1

- **Fent servir l'aplicació keytool crear un magatzem de claus "keystore" que contingui un parell de claus asimètriques i estigui protegit per una paraula de pas**

1. Per començar, he creat un magatzem de claus JCEKS amb un parell de claus RSA (pública/privada) i protegit amb contrasenya.

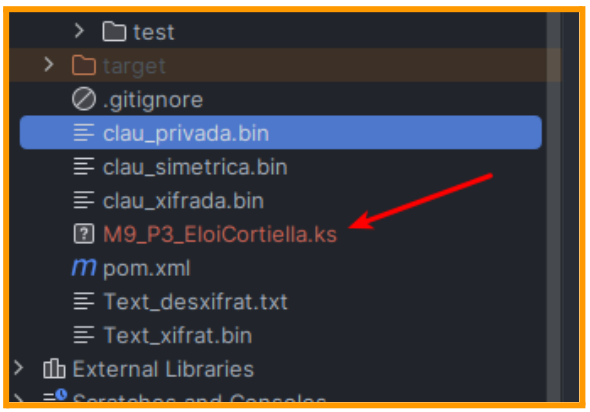
   La comanda utilitzada és:

   keytool -genkeypair -storetype JCEKS -dname "cn=Eloi Cortiella" -alias myKeys -keyalg RSA -keysize 2048 -keypass 654321 -keystore M9_P3_EloiCortiella.ks -storepass 123456 -validity 365

   El resultat de l'execució de la comanda és el següent:

```
→  P1_EloiCortiella git:(master) ✗ keytool -genkeypair -storetype JCEKS -dname "cn=Eloi Cortiella" -alias myKeys -keyalg RSA
-keysize 2048 -keypass 654321 -keystore M9_P3_EloiCortiella.ks -storepass 123456 -validity 365
Generating 2.048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 365 days
        for: CN=Eloi Cortiella
```

   Aquí es pot observar que s'ha creat la keystore correctament:



   A continuació explico la funció de cada paràmetre usat.

Alumne: Eloi Cortiella Fortuño
Curs: 2n Desenvolupament d'Aplicacions Multiplataforma - 2025/2026
Professor: Marc Fuertes Díez

Explicació dels paràmetres principals:

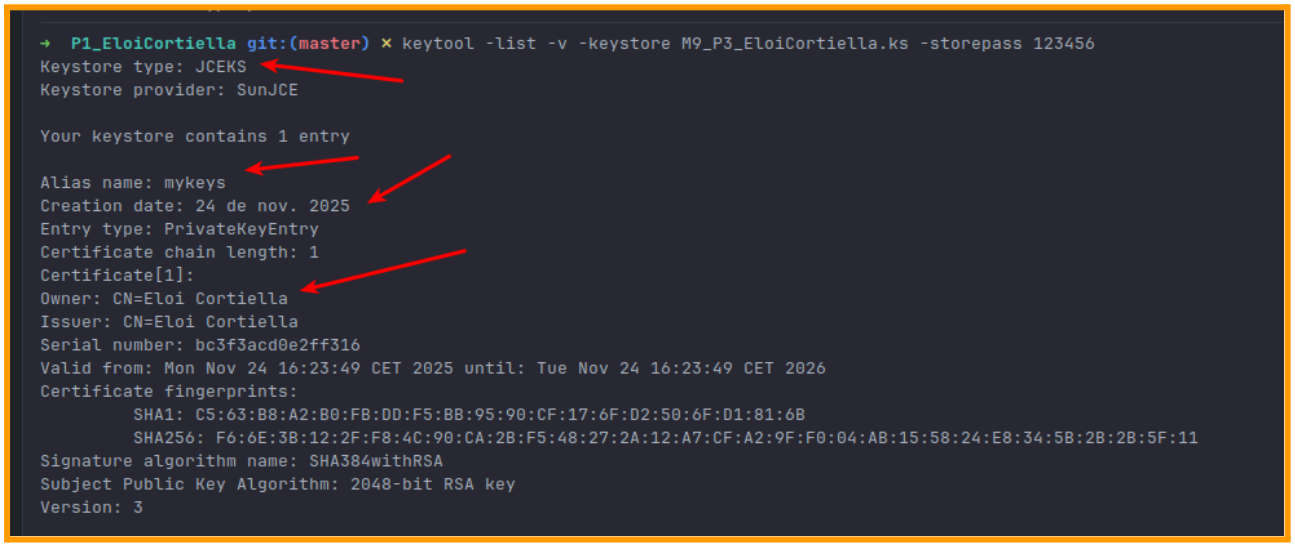
- -**genkeypair:** Indica que es generarà un parell de claus asimètriques (pública i privada).
- **-storetype JCEKS**: El keystore es crea en format **JCEKS**, que és el tipus que espera el mètode load KeyStore de la classe Crypto.
- **-dname "cn=Eloi Cortiella":** Dades del subjecte; nom comú amb el meu nom i cognoms.
- **-alias myKeys:** Àlies que identifica el parell de claus dins del keystore (és el nom que faré servir a l'aplicació Java).
- **-keyalg RSA i -keysize 2048**: Algoritme i mida de les claus asimètriques: RSA de 2048 bits.
- **-keypass 654321:** Contrasenya que protegeix **la clau privada**.
- **-keystore M9_P3_EloiCortiella.ks**: Nom del fitxer de magatzem de claus que es genera.
- **-storepass 123456**: Contrasenya del **keystore**
- **-validity 365**: Dies de validesa del certificat associat al parell de claus.

Un cop s'ha creat, he comprovat el contingut de dins amb la següent comanda:

keytool -list -v -keystore M9_P3_EloiCortiella.ks -storepass 123456

El resultat ha estat exitós:

A la sortida es pot veure que el keystore és de tipus JCEKS, que existeix l'àlies myKeys i que el certificat generat té una validesa d'un any.



Tasca 2

- **Modificar la Pràctica 3 per agafar les claus pública i privada del magatzem de claus creat a l'apartat 1 per:**
  - ***Posar la clau publica al cos del certificat***
  - ***Signar el cos del certificat amb la clau privada.***

Primer, carrego el keystore M9_P3_EloiCortiella.ks amb la contrasenya del magatzem i n'obtinc la clau pública i la privada a partir de l'àlies myKeys:

```
Crypto crypto = new Crypto();

// Carregar la keystore
System.out.println("Carregant keystore: " + KEYSTORE_PATH);
KeyStore ks = crypto.loadKeyStore(KEYSTORE_PATH, KEYSTORE_PASSWORD);

// Obtenir clau pública i privada a partir de l'àlies
if (!ks.containsAlias(KEY_ALIAS)) {
    System.err.println("No s'ha trobat l'àlies '" + KEY_ALIAS + "' al keystore.");
    return;
}

// Certificat associat a l'àlies → d'aquí traiem la clau pública
Certificate certEntrada = ks.getCertificate(KEY_ALIAS);
if (certEntrada == null) {
    System.err.println("No hi ha certificat associat a l'àlies '" + KEY_ALIAS + "'.");
    return;
}
PublicKey clauPublica = certEntrada.getPublicKey();

// Clau privada associada a l'àlies
Key key = ks.getKey(KEY_ALIAS, KEY_PASSWORD.toCharArray());
if (!(key instanceof PrivateKey)) {
    System.err.println("L'àlies '" + KEY_ALIAS + "' no té una clau privada associada.");
    return;
}
PrivateKey clauPrivada = (PrivateKey) key;
```

Amb això ja no genero un KeyPair nou amb randomGenerate, sinó que la clau pública i privada es llegeixen del magatzem creat amb keytool, tal com indica l'enunciat de la pràctica.

Alumne: Eloi Cortiella Fortuño
Curs: 2n Desenvolupament d'Aplicacions Multiplataforma - 2025/2026
Professor: Marc Fuertes Díez

A continuació construeixo el cos del certificat i hi afegeixo la clau pública en format Base64:

```
private static String generarCosCertificat(PublicKey clauPublica) {  1 usage  Eloi-Cortiella
    String clauPublicaBase64 = Base64.getEncoder()
            .encodeToString(clauPublica.getEncoded());

    StringBuilder sb = new StringBuilder();
    sb.append("=== CERTIFICAT DIGITAL M9_P4 ===\n");
    sb.append("Alumne: Eloi Cortiella\n");
    sb.append("Cicle: DAM2 - Programació de Serveis i Processos\n");
    sb.append("Assignatura: M9 - PSP (Seguretat i Criptografia)\n\n");

    sb.append("CLAU PUBLICA (Base64):\n");
    sb.append(clauPublicaBase64).append("\n");

    return sb.toString();
}
```

Aquest text es desa al fitxer certificat_M9_P4_EloiCortiella.txt.
D'aquesta manera, la clau pública del keystore queda incrustada dins del cos del certificat.

El cos del certificat es signa amb la clau privada recuperada del keystore, fent servir el mètode signData de la classe Crypto:

```
// Signar el cos del certificat amb la clau privada
byte[] signatura = crypto.signData(
        cosCertificat.getBytes(StandardCharsets.UTF_8),
        clauPrivada
);
```

La signatura obtinguda es guarda en Base64 al fitxer certificat_M9_P4_EloiCortiella.sig, que es podria verificar posteriorment amb la clau pública.

En executar el main de GestorClaus s'obté, per consola, el següen resultat:

```
/home/alumnat/.jdks/openjdk-23.0.2/bin/java -javaagent:/home/alum
Carregant keystore: M9_P3_EloiCortiella.ks
Claus carregades correctament des del keystore.
✔ Certificat generat i signat correctament.
   - Cos: certificat_M9_P4_EloiCortiella.txt
   - Signatura (Base64): certificat_M9_P4_EloiCortiella.sig
```

```java
private static void guardarCertificatICodi(String cosCertificat, byte[] signatura) throws Exception {
    // Cos del certificat en text pla
    try (FileOutputStream fos = new FileOutputStream( name: "certificat_M9_P4_EloiCortiella.txt")) {
        fos.write(cosCertificat.getBytes(StandardCharsets.UTF_8));
    }

    // Signatura en Base64 per poder veure-la bé com a text
    String signaturaBase64 = Base64.getEncoder().encodeToString(signatura);
    try (FileOutputStream fos = new FileOutputStream( name: "certificat_M9_P4_EloiCortiella.sig")) {
        fos.write(signaturaBase64.getBytes(StandardCharsets.UTF_8));
    }
}
```

I al directori de treball apareixen els fitxers:

- **certificat_M9_P4_EloiCortiella.txt** → cos del certificat amb la clau pública.
- **certificat_M9_P4_EloiCortiella.sig** → signatura digital en Base64.

Aqui mostro com es veu amb l'explorador de fitxers de Ubuntu, on clarament s'observa que s'ha fet correctament: