Segurança e Confiabilidade

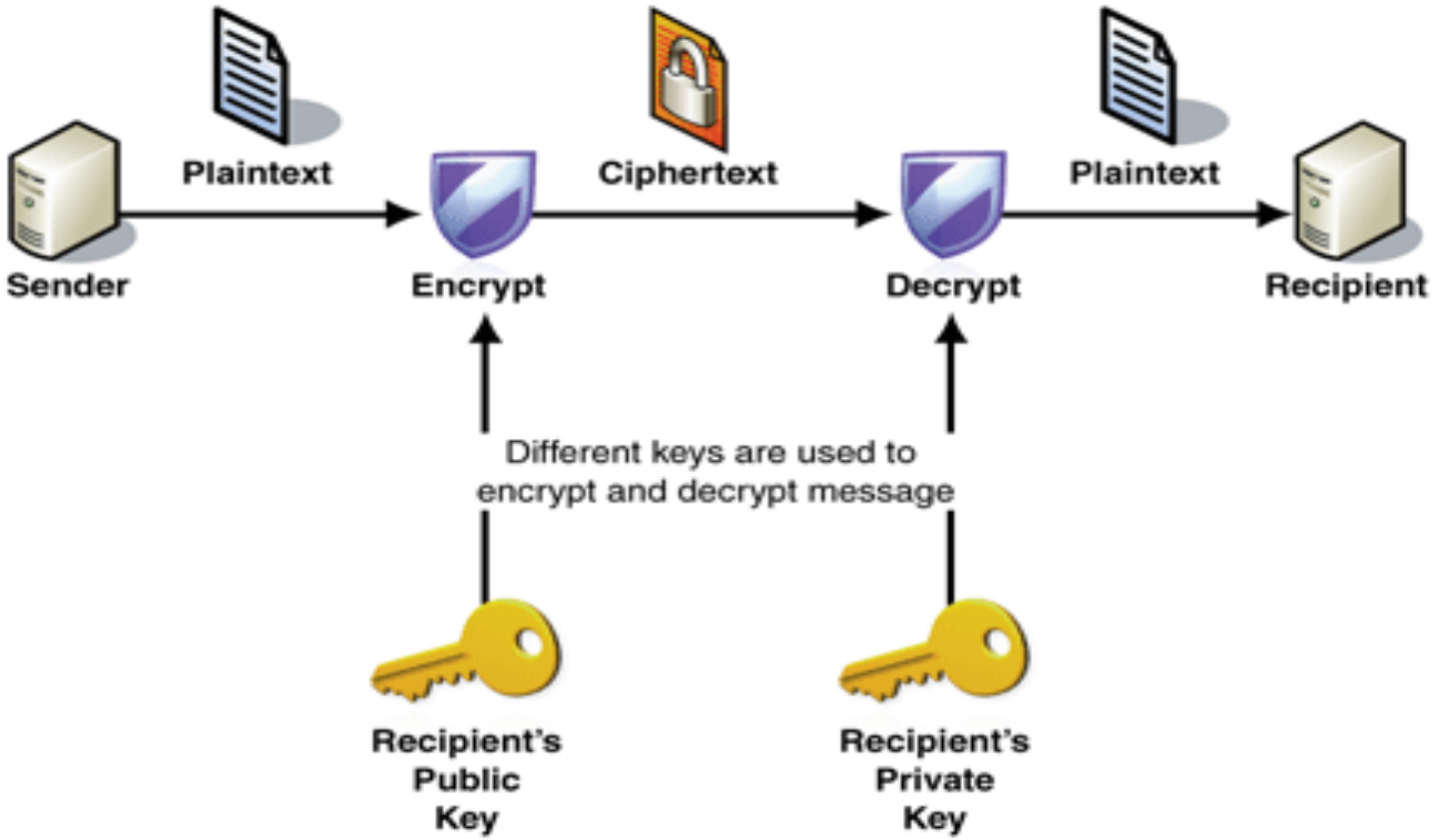
António Casimiro, Alysson Bessani, Alan Oliveira

2022/2023

API segurança do Java

Chaves assimétricas
(e keytool)

Criptografia Assimétrica



Pares de chaves assimétricas

- Existem duas interfaces que estendem *Key* para definir chaves assimétricas
  - *public interface PublicKey extends Key*
  - *public interface PrivateKey extends Key*

- Estas interfaces **não** definem quaisquer métodos a mais, e servem apenas para tornar a organização de tipos mais conveniente

- O fornecedor de segurança da Sun (que acompanha a JVM) oferece dois tipos comuns de pares de chaves assimétricas (há outras nas versões mais novas)
  - DSA (*Digital Signature Algorithm*)
  - RSA (*Rivest, Shamir, Adleman*)
- Exemplo:
  - *public interface RSAPrivateKey extends PrivateKey*
  - *public interface RSAPublicKey extends PublicKey*

- Além disso, é oferecido também o *Diffie-Hellman*
  - Ver classe *KeyAgreement*

Classe *KeyPair*

- A classe *java.security.Keypair* contém as chaves pública e privada
  - *public final class KeyPair*

- Usada normalmente quando se precisa de criar e gerir um par de chaves

- Principais métodos
  - *public KeyPair(PublicKey pub, PrivateKey priv)*
    - construtor que recebe ambas as chaves
  - *public PublicKey getPublic( )*
  - *public PrivateKey getPrivate( )*
    - devolve a chave correspondente

(NOTA: o gestor de segurança não é chamado quando se executa o método *getPrivate()*, o que significa que deve haver algum cuidado com o manuseamento desta classe)

Geração de Pares de Chaves Assimétricas (1/2)

- A classe *java.security.KeyPairGenerator é usada para gerar as chaves*
  - *public abstract class KeyPairGenerator extends KeyPairGeneratorSpi*

- Como todas as classes motor, esta classe é abstracta, não existindo portanto uma implementação na API do Java

- As instâncias desta classe para um dado algoritmo devem ser providas pelos fornecedores (ex., no fornecedor Sun tem-se "RSA"), e são acedidas através da invocação de alguns métodos da classe motor:
  - *public static KeyPairGenerator getInstance(String algorithm)*
  - *public static KeyPairGenerator getInstance(String algorithm, String provider)*

Geração de Pares de Chaves Assimétricas (2/2)

- Outros métodos importantes
  - *public void initialize(int strength)*
  - *public abstract void initialize(int strength, SecureRandom random)*
    - inicializa para um dado nível de segurança (tipicamente o número de bits da chave)
  - *public abstract KeyPair generateKeyPair( )*
  - *public final KeyPair genKeyPair( )*
    - gera as chaves utilizando os parâmetros previamente indicados

- Exemplo de geração de par de chaves RSA:

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024);          // 1024 bits
KeyPair kp = kpg.generateKeyPair( );
PublicKey ku = kp.getPublic();
PrivateKey kr = kp.getPrivate();
```

Criptografia híbrida

- A criptografia híbrida pode ser resumida em três passos:
  1. gera-se uma chave secreta aleatória
  2. cifram-se os dados com a chave secreta
  3. cifra-se a chave secreta com a chave pública do recetor

- Métodos específicos que são oferecidos na class *Cipher*, em que no *init()* se devem indicar as operações Cipher.WRAP_MODE e Cipher.UNWRAP_MODE junto com a chave que irá cifrar/decifrar a chave
  - *public final byte[] wrap(Key key)*
    - cifrar uma chave
  - *public final Key unwrap(byte[] key, String algorithm, int type)*
    - decifrar uma chave, em que é preciso fornecer a chave cifrada, o algoritmo usado para a gerar (e.g., AES ou RSA), e o tipo de chave (Cipher.SECRET_KEY, Cipher.PUBLIC_KEY, ou Cipher.PRIVATE_KEY)

Exemplo de *wrap/unwrap* de chaves – RSA (1/2)

```
public class WrapTest {
    public static void main(String[] args) throws Exception {
        // Passo1: gerar a chave secreta que queremos transmitir
        KeyGenerator kg = KeyGenerator.getInstance("DESede");
        Key sharedKey = kg.generateKey( );

         // Passo 2: cifrar alguns dados
        c = Cipher.getInstance("DESede");
        c.init(Cipher.ENCRYPT_MODE, sharedKey);
        byte[] input = "Stand and unfold yourself".getBytes( );
        byte[] encrypted = c.doFinal(input);

        // Passo3: Obter a chave pública do receptor
        PublicKey ku = (PublicKey) … //obtém a chave pública de RSA de alguma forma
        // preparar o algoritmo de cifra para cifrar a chave secreta
        Cipher c = Cipher.getInstance(“RSA");
        c.init(Cipher.WRAP_MODE, ku);
        // cifrar a chave secreta que queremos enviar
        byte[] wrappedKey = c.wrap(sharedKey);
```

Exemplo de *wrap/unwrap* de chaves – RSA (2/2)

```
        // agora seria enviada a wrappedKey mais os dados cifrados (encrypted)

        // no receptor usaríamos os seguintes passos (re-utilizando algumas estruturas de dados)
        //Passo 1: decifrar a chave secreta
        PrivateKey kr = (PrivateKey) … //obtém a chave privada de RSA de alguma forma
        c = Cipher.getInstance(“RSA");
        c.init(Cipher.UNWRAP_MODE, kr);
        Key unwrappedKey = c.unwrap(wrappedKey, "DESede", Cipher.SECRET_KEY);

        // Passo 2: agora podem-se decifrar os dados
        c = Cipher.getInstance("DESede");
        c.init(Cipher.DECRYPT_MODE, unwrappedKey);
        String newData = new String(c.doFinal(encrypted));
        System.out.println("The string was " + newData);
    }
}
```

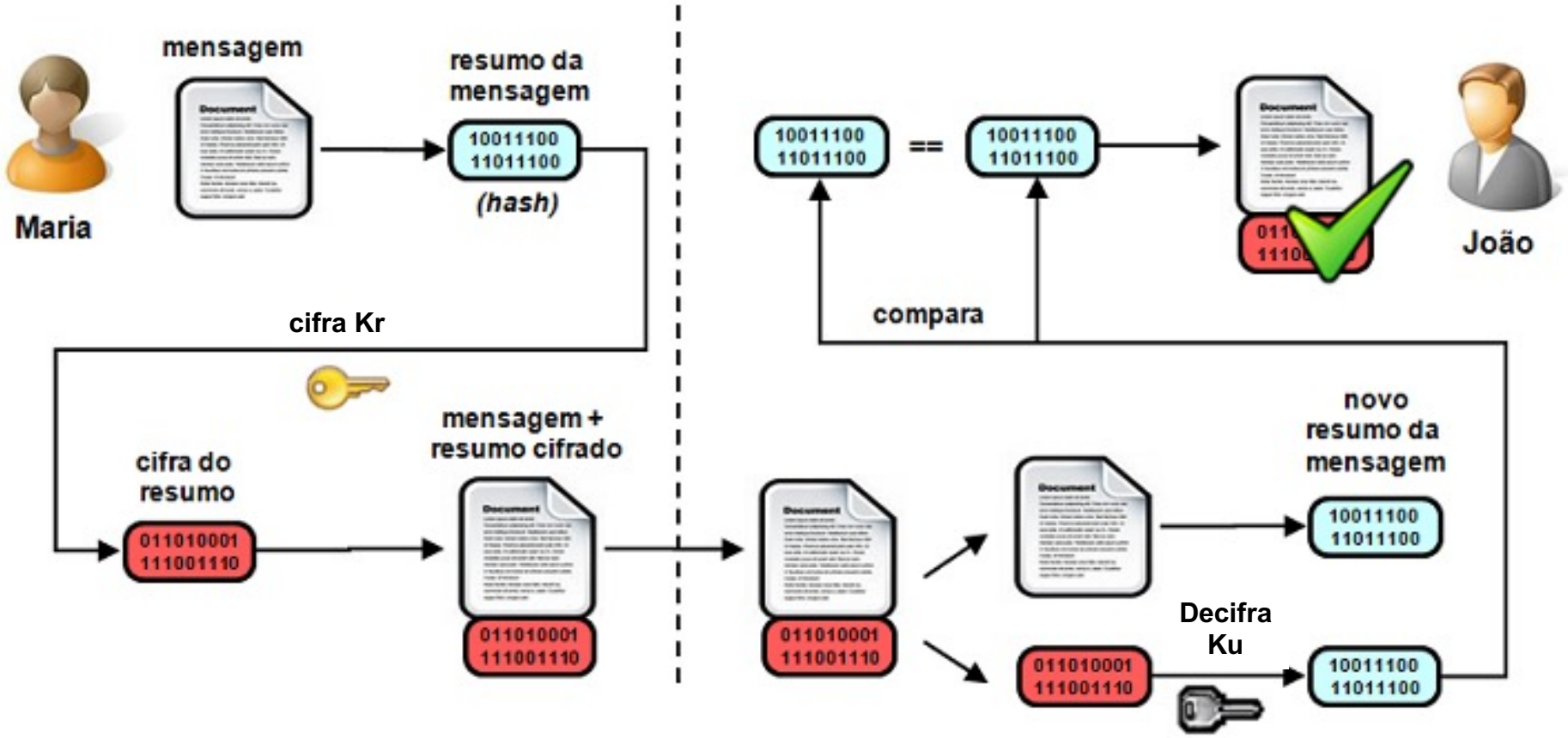
Assinaturas Digitais

- Possibilitam a assinatura com criptografia assimétrica de:
  - objectos que depois podem ser transmitidos entre aplicações
  - classes que depois podem ser agrupadas (*.jar*) e distribuídas por outros utilizadores
- Para a criação das assinaturas existe uma interface programática e uma ferramenta *jarsigner* para assinar arquivos de classes

- As assinaturas são criadas executando-se os seguintes passos:
  1. obtém-se uma síntese dos dados
  2. assina-se (cifra-se) a síntese com a chave privada
- As assinaturas são verificadas também em dois passos:
  1. obtém-se uma síntese dos dados
  2. verifica-se (decifra-se) a assinatura com a chave pública de quem assinou e compara-se com a síntese

"Java Security", cap. 12

Assinaturas Digitais



Classe *Signature* (1)

- A classe *java.security.Signature* abstrai o conceito de assinatura digital, e fornece os métodos para a criação e verificação de assinaturas
  - *public abstract class Signature extends SignatureSpi*

- Como com todas as classes motor é preciso primeiro obter uma instância de um dado algoritmo (e.g., “SHA/DSA“)
  - *public static Signature getInstance(String algorithm)*
  - *public static Signature getInstance(String algorithm, String provider)*

- Em seguida podem ser chamados os seguintes métodos
  - *public final void initSign(PrivateKey privateKey)*
    - inicializar o objeto para que possa ser criada uma assinatura
  - *public void final initVerify(PublicKey publicKey)*
    - inicializar o objeto para que possa ser verificada uma assinatura

Classe *Signature* (2)

- *public final void update(byte b)*
- *public final void update(byte[] b)*
- *public final void update(byte b[], int offset, int length)*
  - adicionar dados que eventualmente serão verificados ou assinados
- *public final byte[] sign( )*
- *public final int sign(byte[] outbuf, int offset, int len)*
  - criar a assinatura (automaticamente o objeto é re-iniciado para que possa criar uma nova assinatura)
- *public final boolean verify(byte[] signature)*
  - verificar a validade da assinatura (o objeto também é re-iniciado)
- *public final void setParameter(AlgorithmParameterSpec param)*
  - indicar os parâmetros do motor de assinatura
- *public final String getAlgorithm( )*
  - devolve o nome do algoritmo realizado

Exemplo de criação de assinatura

Escreve o texto no ficheiro *test* e adiciona uma assinatura no final.

```
public class WriteSignedFile {
    public static void main(String args[]) {
        String data = "This have I thought good to deliver thee, .....";
        FileOutputStream fos = new FileOutputStream("test");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        PrivateKey pk = (PrivateKey) … //obtém a chave privada de alguma forma
        Signature s = Signature.getInstance("MD5withRSA");
        s.initSign(pk);
        byte buf[] = data.getBytes( );
        s.update(buf);
        oos.writeObject(data);
        oos.writeObject(s.sign( ));
        fos.close();
    }
}
```

Exemplo de verificação de assinatura

Verifica se a assinatura do ficheiro gerado no exemplo anterior é correta.

```
public class ReadFileVerifySign {
    public static void main(String args[]) throws Exception {
        FileInputStream fis = new FileInputStream("test");
        ObjectInputStream ois = new ObjectInputStream(fis);
        String data = (String) ois.readObject( );            //não fiz verificação de erro
        byte signature[] = (byte[]) ois.readObject( );       //não fiz verificação de erro
        System.out.println(data);
        Certificate c = … //obtém um certificado de alguma forma (ex., de um ficheiro)
        PublicKey pk = c.getPublicKey( );
        Signature s = Signature.getInstance("MD5withRSA");
        s.initVerify(pk);
        s.update(data.getBytes( ));
        if (s.verify(signature))
            System.out.println("Message is valid");
          else
            System.out.println("Message was corrupted");
        fis.close();
    }
}
```

Classe *SignedObject*

- A classe *java.security.SignedObject* encapsula num mesmo objecto tanto os dados como a sua assinatura
  - *public final class SignedObject implements Serializable*
- Os métodos deste objecto são
  - *public SignedObject(Serializable o, PrivateKey pk, Signature engine)*
    - construtor usado para assinar um objeto com uma dada chave
  - *public Object getContent( )*
    - devolve o objeto guardado no signed object
  - *public byte[] getSignature( )*
    - devolve a assinatura
  - *public String getAlgorithm( )*
    - retorna o nome do algoritmo usado na assinatura
  - *public boolean verify(PublicKey pk, Signature engine)*
    - valida a assinatura do objeto guardado

Serialização de Mensagem: objecto + assin. + certificado (1)

```
public class Message implements Serializable {
    public SignedObject object;
    public transient Certificate certificate;

    public Message(SignedObject object, Certificate certificate) {
        this.object = object;
        this.certificate=certificate;
    }

    private void writeObject(ObjectOutputStream out) throws IOException {
        out.defaultWriteObject( );       //escreve tudo que não é estático ou transiente

        try {
            out.writeObject(certificate.getEncoded( ));
        } catch (CertificateEncodingException cee) {
            throw new IOException("Can't serialize object " + cee);
        }
    }
    //continua no próximo slide
```

Serialização de Mensagem: objecto + assin. + certificado (2)

```
//continuação do slide anterior
private void readObject(ObjectInputStream in) throws IOException {
    try {
        in.defaultReadObject( );    //lê tudo que não é estático ou transiente
    } catch (ClassNotFoundException cnfe) {
        throw new IOException("Can't de−serialize object " + cnfe);
    }
    try {
        byte b[] = (byte []) in.readObject( );
        CertificateFactory cf = CertificateFactory.getInstance("X509");
        certificate = cf.generateCertificate(new ByteArrayInputStream(b));
    } catch (CertificateException ce) {
        throw new IOException("Can't deserialize object " + ce);
    }
}
}
```

Exemplo de Criação e Serialização de uma Mensagem

Cria o ficheiro *test.obj* com a mensagem (classe Message) serializada.

```
public class WriteMessage {
    public static void main(String args[]) throws Exception {
        FileOutputStream fos = new FileOutputStream("test.obj");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        KeyStore ks = … //obtém uma keystore de alguma forma
        Certificate certs[] = ks.getCertificateChain(args[0]);
        PrivateKey pk = (PrivateKey) ks.getKey(args[0], args[1].toCharArray( ));
        SignedObject signedObject = new SignedObject(
          "This have I thought .....", pk,Signature.getInstance("MD5withRSA"));
        Message m = new Message(signedObject,certs[0]);
        oos.writeObject(m);
        fos.close();
    }
}
```

Gestão de chaves

"KeyStores"

Ferramenta "keytool"

Gestão de chaves

- Os algoritmos criptográficos requerem a utilização de chaves durante a sua execução (as sínteses seguras são uma excepção)
- O *sistema de gestão de chaves* tem a responsabilidade do armazenamento e manutenção das chaves quer através de ferramentas próprias, quer através da linguagem de programação
- A gestão de chaves no Java é baseada no conceito de *Keystores*
- As *Keystores* são manipuladas através da ferramenta ***Keytool***, e de uma interface da API Java

"Java Security", cap. 10

Principais conceitos (1/2)

- ***Keystore***
  - Um **ficheiro** que armazena um conjunto de chaves e certificados
  - Normalmente encontra-se localizado na diretoria do utilizador com o nome *.keystore*

- Tipos de entradas (*key entries*):
  - *KeyStore.PrivateKeyEntry*
    - Entradas que contêm uma chave privada e um caminho de certificação para a respectiva chave pública.
  - *KeyStore.SecretKeyEntry*
    - Entradas que contêm uma chave secreta
  - *KeyStore.TrustedCertificateEntry*
    - Entradas que contêm um certificado pertencente a outro utilizador – certificados nos quais o dono da *keystore* confia

- Alias
  - é um identificador de uma entrada na *Keystore (tipicamente uma abreviatura do nome completo)*

Principais conceitos (2/2)

- ***DN** (Distinguish Name)* -- um nome em formato X.500
  - exemplo: `CN=Jose Luis, OU=DI, O=Faculdade Ciencias UL, L=Lisboa, ST=Lisboa, C=PT`

- *Formatos das keystores*
  - na API Java, uma *Keystore* é um motor com diversas realizações possíveis:
    - **JKS** – formato por omissão; só permite armazenar entradas com chaves assimétricas e entradas de certificados **(por *default* é esse!)**
    - **JCEKS** – fornecido pelo JCE; armazena também chaves secretas
    - **PKCS12** – não fornece todos os serviços, e serve basicamente para importar certificados/chaves codificadas de acordo com este padrão
      - NOTA: quer o JKS como o JCEKS cifram as chaves privadas, no entanto a segunda concretização usa criptografia mais forte
      - Para alterar o algoritmo de omissão deve-se editar o ficheiro *$JREHOME/lib/security/java.security :* *keystore.type=JCEKS*

Na versão 9 ou posterior:
*$JREHOME/*conf/security/*java.security*

- *Autoridades de certificação de confiança:*
  - Certificados com chaves públicas de CA bem conhecidas (e.g., Verisign)
  - Localizado em *$JREHOME/lib/security/cacerts*

A ferramenta *Keytool*

- Uma ferramenta para administrar chaves fornecidas com o JRE
  - criar chaves
  - importar certificados digitais
  - exportar chaves
  - etc...

- Usa uma interface de linha de comando
  - Ver em *$JREHOME/bin/keytool*

- Quando se executa o comando sem argumentos, este indica uma lista de opções

- Quando se corre uma das opções ele pede uma palavra de passe que por omissão tem o valor ***changeit***

Opções da keytool

- c:\Program Files\Java\jre_XX_\bin>keytool
- Key and Certificate Management Tool

Commands:

```
-certreq            Generates a certificate request
-changealias        Changes an entry's alias
-delete             Deletes an entry
-exportcert         Exports certificate
-genkeypair         Generates a key pair
-genseckey          Generates a secret key
-gencert            Generates certificate from a certificate request
-importcert         Imports a certificate or a certificate chain
-importkeystore     Imports one or all entries from another keystore
-keypasswd          Changes the key password of an entry
-list               Lists entries in a keystore
-printcert          Prints the content of a certificate
-printcertreq       Prints the content of a certificate request
-printcrl           Prints the content of a CRL file
-storepasswd        Changes the store password of a keystore
```

Use "keytool -command_name -help" for usage of command_name

Exemplos de uso da Keytool

- Listar as entradas de uma keystore

  keytool -list -keystore ..\lib\security\cacerts

```
Enter keystore password:
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 104 entries

verisignclass2g2ca [jdk], Aug 25, 2016, trustedCertEntry,
Certificate fingerprint (SHA1): B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
digicertassuredidg3 [jdk], Aug 25, 2016, trustedCertEntry,
Certificate fingerprint (SHA1): F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
verisignuniversalrootca [jdk], Aug 25, 2016, trustedCertEntry,
Certificate fingerprint (SHA1): 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
```

Enter keystore password: *changeit*

Exemplos de uso da Keytool

- Criação de uma chave secreta com nome (alias) *secKey e guardar na keystore myKeys. Depois ver o conteúdo da keystore*

keytool -genseckey -alias secKey -storetype JCEKS -keystore myKeys

```
Enter keystore password:
Re-enter new password:
Enter key password for <secKey>
        (RETURN if same as keystore password):
Re-enter new password:

Warning:
The JCEKS keystore uses a proprietary format. It is r
eytool -importkeystore -srckeystore C:\Users\iberia\S
```

Password para proteger o ficheiro da keystore

Dar a path para onde armazenar o ficheiro da keystore

Password para proteger a chave secreta

keytool -list -storetype JCEKS -keystore myKeys

```
Enter keystore password:
Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 1 entry

seckey, Mar 21, 2020, SecretKeyEntry,
```

alias

Diz que é uma chave secreta

Exemplos de uso da Keytool

- Criação de um par de chaves pública-privada DSA (a chave pública vai para um certificado)

  keytool -genkeypair -alias keyDSA -storetype JCEKS -keystore myKeys

```
Enter keystore password:
What is your first and last name?
  [Unknown]:  Ib Med
What is the name of your organizational unit?
  [Unknown]:  FCUL
What is the name of your organization?
  [Unknown]:  ULisboa
What is the name of your City or Locality?
  [Unknown]:  Lisboa
What is the name of your State or Province?
  [Unknown]:  Lisboa
What is the two-letter country code for this unit?
  [Unknown]:  PT
Is CN=Ib Med, OU=FCUL, O=ULisboa, L=Lisboa, ST=Lisboa, C=PT correct?
  [no]:  yes

Enter key password for <keyDSA>
        (RETURN if same as keystore password):
```

CN

OU

O

L

ST

C

maiúsculas

Exemplos de uso da Keytool

- Criação de um par de chaves pública-privada RSA (a chave pública vai para um certificado)

  keytool -genkeypair -alias keyRSA -keyalg RSA -keysize 2048 -storetype JCEKS -keystore myKeys

```
Enter keystore password:
What is your first and last name?
  [Unknown]:  Ib Med RSA
What is the name of your organizational unit?
  [Unknown]:  FCUL
What is the name of your organization?
  [Unknown]:  Ulisboa
What is the name of your City or Locality?
  [Unknown]:  Lx
What is the name of your State or Province?
  [Unknown]:  Lx
What is the two-letter country code for this unit?
  [Unknown]:  PT
Is CN=Ib Med RSA, OU=FCUL, O=Ulisboa, L=Lx, ST=Lx, C=PT correct?
  [no]:  yes

Enter key password for <keyRSA>
        (RETURN if same as keystore password):
```

Exemplos de uso da Keytool

- *Ver o conteúdo da keystore*

  keytool -list -storetype JCEKS -keystore myKeys

```
Enter keystore password:
Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 3 entries

keyrsa, Mar 21, 2020, PrivateKeyEntry,
Certificate fingerprint (SHA1): 67:B0:C8:CB:B6:CC:F4:E9:5F:14:B4:99:F4:53:25:5A:3D:BA:08:00
keydsa, Mar 21, 2020, PrivateKeyEntry,
Certificate fingerprint (SHA1): AB:2C:98:B1:EE:79:AA:AD:AA:E0:1F:EC:C3:8B:E7:44:41:48:05:00
seckey, Mar 21, 2020, SecretKeyEntry,
```

Contem a chave privada e o certificado que contem a chave pública

aliases

Exemplos de uso da Keytool

- Remoção de uma chave

  keytool -delete -alias keyDSA -storetype JCEKS -keystore myKeys

- *Ver o conteúdo da keystore*

  keytool -list -storetype JCEKS -keystore myKeys

```
Enter keystore password:
Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 2 entries

keyrsa, Mar 21, 2020, PrivateKeyEntry,
Certificate fingerprint (SHA1): 67:B0:C8:CB:B6:CC:F4:E9:5F:14:B4:99:F4:53:25:5A:3D:BA:08:00
seckey, Mar 21, 2020, SecretKeyEntry,
```

Exemplos de uso da Keytool

❖ Extração do certificado de uma chave pública da keystore

keytool -exportcert -alias keyRSA -storetype JCEKS -keystore myKeys -file keyRSApub.cer

```
Enter keystore password:
Certificate stored in file <C:\Users\iberia\SC\TP6\keyRSApub.cer>
```

Dar a path para onde armazenar o ficheiro com o certificado

❖ Inclusão de um certificado na *keystore*

keytool -importcert -alias newcert -file keyRSApub.cer -storetype JCEKS -keystore myKeys

Dar a path do ficheiro com o certificado

API para gestão de chaves

- Permite a gestão de keystores a partir de um programa Java
- A classe *java.security.KeyStore* é um motor que representa uma keystore em memória
  - *public class KeyStore*
- Usar os seguintes métodos para se obter uma instância (e.g., “JKS”)
  - *public static final KeyStore getInstance(String type)*
  - *public static final KeyStore getInstance(String type, String provider)*
- Exemplos de métodos (existem bastantes mais):
  - *public final void load(InputStream is, char[] password)*
    - inicializar com a informação a ler do input stream e a password da keystore
  - *public final Enumeration aliases( )*
    - retorna a lista com todos os *aliases* na *keystore*
  - *public final Certificate getCertificate(String alias)*
    - devolve o certificado associado ao alias
  - *Public final Key getKey(String alias, char[] password)*
    - devolve a chave secreta ou privada do alias (tem de se passar a password)

API para gestão de chaves

- Como obter um certificado da keystore ?

```
FileInputStream kfile = new FileInputStream(“myKeys");  //keystore
KeyStore kstore = KeyStore.getInstance("JCEKS");
kstore.load(kfile, "123456".toCharArray());            //password da keystore
Certificate cert = kstore.getCertificate(“keyRSA");      //alias da keypair
```

- Obter uma chave privada da keystore:

```
Key myPrivateKey = kstore.getKey(“keyRSA", "123456".toCharArray());
```

- Obter o certificado contido num ficheiro de certificado (.cer):

```
FileInputStream fis = new FileInputStream("keyRSApub.cer");
CertificateFactory cf = CertificateFactory.getInstance("X509");
Certificate cert = cf.generateCertificate(fis);
```

- Obter uma chave pública do certificado (da keystore ou do ficheiro .cer):

```
PublicKey pk = cert.getPublicKey( );
```