API segurança do Java

Chaves assimétricas
(e keytool)

1

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
  - DSA (*Digital Signature Algorithm*)
  - RSA (*Rivest, Shamir, Adleman*)
- Além disso, é oferecido também o *Diffie-Hellman*
  - Ver classe *KeyAgreement*
- Exemplo:
  - *public interface RSAPrivateKey extends PrivateKey*
  - *public interface RSAPublicKey extends PublicKey*

2

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)

- Outro 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;
```


5

Criptografia híbrida

- A criptografia híbrida pode ser resumida em três passos:
  - gera-se uma chave secreta aleatória
  - cifram-se os dados com a chave secreta
  - 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., DES ou RSA), e o tipo de chave (Cipher.SECRET_KEY, Cipher.PUBLIC_KEY, ou Cipher.PRIVATE_KEY)

6

Exemplo de *wrap/unwrap* de chaves (1)

```
public class WrapTest {
   public static void main(String[] args) throws Exception {
      // gerar a chave que queremos transmitir
      KeyGenerator kg = KeyGenerator.getInstance("DESede");
      Key sharedKey = kg.generateKey( );
      // vamos usar uma chave baseada numa password para a cifrar
      String password = "Come you spirits that tend on mortal thoughts";
      byte[] salt = { (byte) 0xc9, (byte) 0x36, (byte) 0x78, (byte) 0x99, (byte) 0x52,
                      (byte) 0x3e, (byte) 0xea, (byte) 0xf2 };
      PBEKeySpec keySpec = new PBEKeySpec(password.toCharArray( ));
      SecretKeyFactory kf = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
      SecretKey passwordKey = kf.generateSecret(keySpec);
      // preparar o algoritmo de cifra
      PBEParameterSpec paramSpec = new PBEParameterSpec(salt, 20);
      Cipher c = Cipher.getInstance("PBEWithMD5AndDES");
      c.init(Cipher.WRAP_MODE, passwordKey, paramSpec);

      // cifrar a chave secreta que queremos enviar
      byte[] wrappedKey = c.wrap(sharedKey);
```


Exemplo de *wrap/unwrap* de chaves (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);

      // agora seria enviada a wrappedKey mais os dados cifrados
      // no receptor usaríamos os seguintes passos (re-utilizando algumas estruturas de dados)
      c = Cipher.getInstance("PBEWithMD5AndDES");
      c.init(Cipher.UNWRAP_MODE, passwordKey, paramSpec);
      Key unwrappedKey = c.unwrap(wrappedKey, "DESede", Cipher.SECRET_KEY);

      // 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 e compara-se com a síntese

"Java Security", cap. 12

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 void final initVerify(PublicKey publicKey)*
    - inicializar o objecto para que possa ser verificada uma assinatura
  - *public final void initSign(PrivateKey privateKey)*
    - inicializar o objecto para que possa ser criada 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 objecto é re-iniciado para que possa criar uma nova assinatura)
- *public final boolean verify(byte[] signature)*
  - verificar a validade da assinatura (o objecto 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 correcta.

```
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();
   }
}
```


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 directoria 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, S=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
    - **JCEKS** – fornecido pelo JCE; armazena também chaves secretas
    - **PKCS12** – norma
      - 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 *$JREHOME/lib/security/java.security : keystore.type=JCEKS*
- *Autoridades de certificação de confiança:*
  - Certificados com chaves públicas de CA bem conhecidas (e.g Verisign)
  - Localizado em *$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***

Exemplos de uso da Keytool

- Listar as entradas de uma key store
  keytool -list -keystore ..\lib\security\cacerts
- Criação de uma chave secreta com nome *seckey*
  keytool -genseckey -alias secKey -storetype PKCS12 *-keystore myKeys*
  keytool -list -storetype PKCS12 *-keystore myKeys*
- Criação de um par de chaves público privadas (a pública vai para um certificado)
  keytool -genkeypair -alias keypair -storetype PKCS12 *-keystore myKeys*
  keytool -genkeypair -alias keyRSA -keyalg RSA -keysize 2048 -storetype PKCS12 *-keystore myKeys*
- Remoção de uma chave
  keytool -delete -alias keypair -storetype PKCS12 *-keystore myKeys*

Exemplos de uso da Keytool

- Extração do certificado de uma chave publica da keystore
  keytool -exportcert -alias keyRSA -storetype PKCS12 *-keystore myKeys -file keyRSApub.cer*
- Inclusão de um certificado na *keystore*
  keytool -importcert -alias newcert -file keyRSApub.cer -storetype PKCS12 *-keystore myKeys*

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 o objeto com a informação a ler do input stream
  - *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 associada ao alias (tem de se passar a password)

API para gestão de chaves

- Como obter um certificado da keystore ?

```
FileInputStream kfile = new FileInputStream("keystore.teste");  //keystore
KeyStore kstore = KeyStore.getInstance("PKCS12");
kstore.load(kfile, "123456".toCharArray());                     //password
Certificate cert = kstore.getCertificate("alice");              //alias do utilizador
```

- Obter uma chave privada da keystore:

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