UNIVERSIDADE DO MINHO

ESCOLA DE ENGENHARIA

MSDPA – Data Warehousing

Segurança e Privacidade em Sistemas de Armazenamento e Transporte de Dados

Java Cryptography Architecture (JCA)

Rui Manuel Coimbra Oliveira Afonseca

Braga

2006/2007

Índice

Índice de Figuras

1. Introdução

A API (*Application Programming Interface*) de segurança da linguagem de programação Java permite aos programadores incluir nas suas aplicações funcionalidades criptográficas. A base de trabalho para aceder e desenvolver funcionalidades criptográficas é definida pela JCA (*Java Cryptography Architecture*). JCE (*Java Cryptography Extension*) é uma extensão à JCA, inclui funcionalidades criptográficas mais poderosas.

Conhecer a API não é suficiente para desenvolver aplicações seguras, sem falhas de segurança graves. Também é necessário perceber como funcionam as principais técnicas criptográficas e que propriedades (confidencialidade, integridade, etc.) de segurança são garantidas quando usamos determinada técnica criptográfica.

Este documento apresenta a JCA num contexto mais prático, fazendo uso de um caso de exemplo que utiliza as principais técnicas criptográficas. A estrutura deste documento é composta por quatro capítulos. O primeiro é uma pequena introdução ao caso de estudo deste documento. No segundo capítulo, uma breve descrição da JCA. No terceiro capítulo irei começar por apresentar o caso prático de exemplo, de seguida a sua resolução conceptual. Depois irei descrever os protocolos usados nas comunicações entre as entidades. Por fim, descrevo o uso de certificados de chave pública no âmbito do caso apresentado. No último capítulo apresento as principais conclusões, depois do desenvolvimento deste trabalho.

2. Java Cryptography Architecture (JCA)

Este capítulo servirá para apresentar a Java Cryptography Architecture (JCA). Esta abordará os seus princípios básicos de funcionamento, a sua arquitectura e principais objectos.

2.1. Princípios

A JCA fornece uma API para serviços criptográficos. Uma das características da JCA é a independência da plataforma, sendo no entanto, mais uma característica herdada da linguagem JAVA que uma característica específica da JCA. É independente da implementação, possibilitando a escolha mais conveniente dos algoritmos criptográficos utilizados. Por fim, é extensível, ou seja, é relativamente fácil acrescentar novos algoritmos criptográficos á JCA.

2.2. Arquitectura

Os principais componentes da arquitectura JCA são descritos de seguida:

- ***Engine Classes*:** Estas classes definem abstractamente um serviço criptográfico (sem implementação).

- ***Provides***
  - ***Provider SUN***
  - ***Provider IAIK***
  - ***…***

Estes são pacotes (*package*s) que fornecem implementações concretas de serviços criptográficos. O *provider* SUN é fornecido por omissão na distribuição da JCA. No entanto, existem outros mais completos em determinados aspectos, como é o caso do *provider* IAIK. Este integra de forma elegante a API para manipulação de estruturas de dados ASN.1 (Abstract Syntax Notation One). Esta é uma notação para definir tipos de dados, desde os tipos básicos, como inteiros aos tipos estruturados, como sequências. Existindo várias normas que definem a codificação de objectos criptográficos (chaves, certificados, etc.) utilizando a ASN.1.

- ***Java Cryptography Extension (JCE)*:** É uma extensão à JCA que inclui técnicas criptográficas mais poderosas. Sendo originalmente criada para ultrapassar limitações impostas pelo governo do Estados Unidos à exportação de criptografia forte. Foi opcional até a versão 1.4 da Java 2 SDK.

2.3. Principais Objectos

Os principais objectos disponíveis na JCA são:

- ***Signature:*** Esta classe serve para assinar (com a chave privada) e verificar uma assinatura (com a chave publica). Para implementar estas funcionalidades podem ser usados algoritmos de assinaturas digitais como o DSA e RSA com funções de *hash* (MD5 ou SHA1).

- ***X509Certificate:*** Representa um certificado de chave pública X.509, com as características definidas pela norma.

- ***KeyStore*:** Este pode ser usado para gerir um repositório de chaves e certificados. As chaves privadas têm associadas, uma cadeia de certificados, que autentica a chave pública correspondente. Uma *Keystore* também pode conter certificados de confiança. Para obter uma chave privada deste repositório é necessário fornecer *password*.

De seguida apresento mais três objectos, agora pertencentes a JCE.

- ***Cipher*:** Este objecto serve para cifrar e decifrar de acordo com o algoritmo escolhido. Por exemplo, se definir o seguinte parâmetro: "DES/ECB/PKCS5Padding" significa que vai ser usado o algoritmo DES no modo ECB com *padding* definido por PKCS5Padding.

- ***KeyGenerator*:** Esta classe serve para gerar chaves secretas para algoritmos simétricos.

- ***SealedObject*:** Esta classe permite criar um objecto e proteger o seu conteúdo com um algoritmo criptográfico. Qualquer objecto que implemente a interface "j*ava.io.Serializable*" pode estar contido num *SealedObject*, podendo portanto ser cifrado com um algoritmo como o DES e futuramente decifrado e reconstruído. Os *SealedObject* são claramente vantajosos em muitos cenários, principalmente na comunicação entre máquinas, pois torna o processo de envio e recepção de informação cifrada menos complexo, porque estamos a enviar apenas um objecto, que encapsula toda a complexidade.

3. Caso de Exemplo

Como exemplo vamos ver um caso prático em que são utilizadas as principais técnicas criptográficas actuais.

3.1. Apresentação

3.1.1 Aplicação de Chat

O caso prático apresentado foi proposto pelo Professor Manuel Barbosa do Departamento de Informática da Universidade do Minho, no contexto da disciplina Criptografia Aplicada 2003/2004. Consiste na implementação de uma aplicação de *Chat*, com funcionalidade protegidas por técnicas criptográficas, tais como: identificação de utilizadores, anonimato de utilizadores, estabelecimento de canais seguros de comunicação, mensagens *off-line*, etc. Podemos ver na Figura 1, imagens da aplicação.

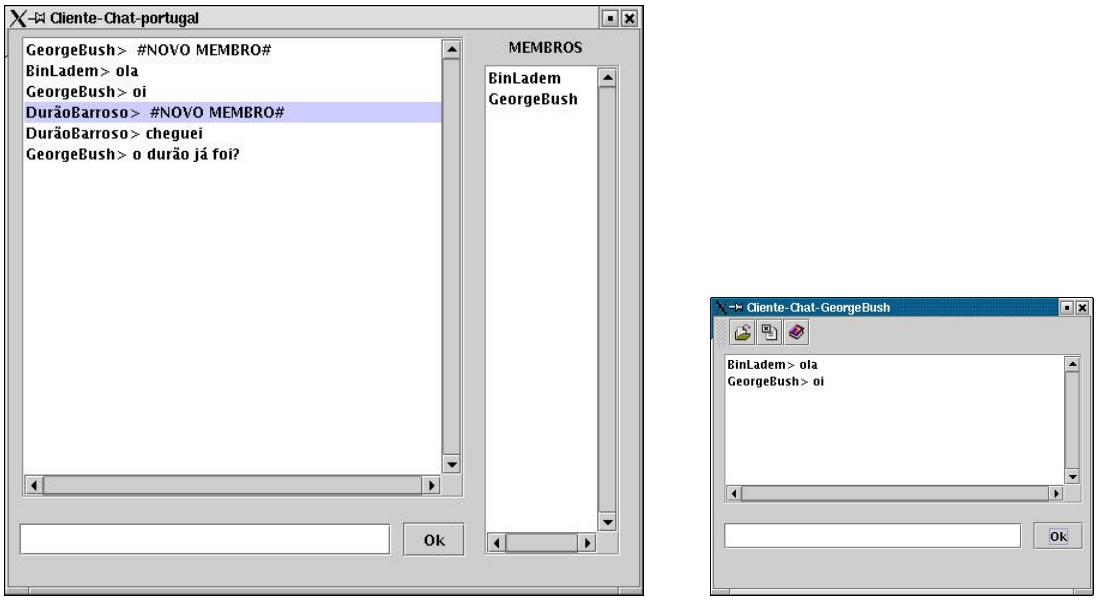

Figura 1 - Sala de conversação (esquerda), conversão privada (direita)

3.1.2 Entidades do Sistema

As entidades presentes no sistema proposto são as seguintes:

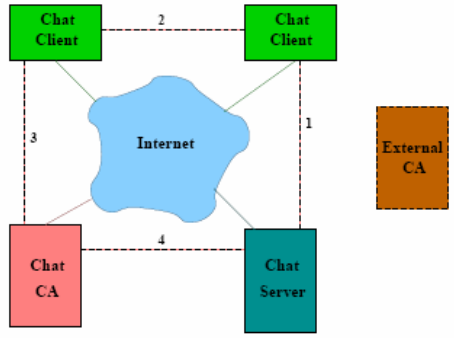

Figura 2 - Entidades do Sistema

- **Autoridade de Certificação Externa:** Esta entidade tem a responsabilidade de certificar a verdadeira identidade dos utilizadores do sistema. A certificação consiste em associar a chave pública ao nome real do utilizador.

- **Autoridade de Certificação Dedicada:** Esta entidade regista as alcunhas dos utilizadores do sistema, garantindo o anonimato aos utilizadores e impondo políticas de exclusão de utilizadores perigosos.

- **Servidor de Chat:** Com funcionalidades de gestão da comunidade de utilizadores. Fornece canais de comunicação em grupo (IRC), fornecendo também o apoio a funcionalidades de comunicação *peer-to-peer* e mensagens *off-line* entre utilizadores.

- **Cliente de Chat:** Fornece uma interface de acesso ao utilizador do chat, com todas as garantias de segurança na protecção dos seus dados.

3.1.3 Ligações Seguras

As comunicações entre entidades do sistema fazem-se por canais abertos. De seguida apresento as ligações possíveis neste sistema e as técnicas criptográficas usadas para proteger essas ligações.

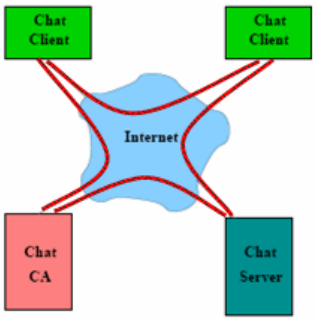

Figura 3 - Comunicações seguras entre entidades

- ***Chat Client – Chat Server:*** Como a comunicação entre os clientes e o servidor serão na maioria dos casos, mensagens de salas de conversação, estas têm de ser rapidamente cifradas e decifradas, pois não era aceitável esperar alguns segundos para que uma simples mensagem de texto surgisse no ecrã destino. Assim é obrigatório o uso de algoritmos simétricos nesta comunicação, pois são claramente mais rápidos que os assimétricos. Portanto, o servidor gera uma chave de sessão que envia ao cliente usando a técnica Envelope Digital.

- ***Chat Client – Chat Client*****:** Mais uma vez a comunicação tem de ser rápida, pois esta comunicação consiste em trocar mensagens numa conversação privada. No entanto, aqui é desejável que a chave de sessão seja acordada entre os dois intervenientes na comunicação. Portanto será usado o protocolo *Station-To-Station*.

- ***Chat Client – Chat CA*****:** Neste cenário são apenas trocadas mensagens de arranque e controlo, como o registo de uma nova alcunha. Portanto, as mensagens trocadas são esporádicas e esperar alguns segundos por uma mensagem não é problemático. Então, a informação é cifrada com a chave pública do destinatário e decifrada no destino com a chave privada correspondente.

- ***Chat Server – Chat CA*****:** Neste cenário são apenas trocadas mensagens de controlo, por exemplo a exclusão de um utilizador perigoso. Tal como no cenário anterior, a informação é cifrada com a chave pública do destinatário e decifrada no destino com a chave privada correspondente.

3.2. Desenho Conceptual

A solução deste problema passa por dividi-lo em quatro camadas de software (Figura 4). Este documento apenas pretende descrever em maior detalhe a camada Criptográfica.

Figura 4 - Desenho conceptual

Podemos observar pela Figura 4 que a camada superior é a camada de interface com o utilizador, de seguida temos a camada operacional que implementa todas as questões relacionadas com o atendimento dos clientes, gestão de salas, auxílio nas comunicações, etc. Depois temos a camada criptográfica, que assegura a confidencialidade dos dados dos utilizadores. Por fim a camada de rede, que tem a função de gerir a comunicação, entre as entidades do sistema, por canais abertos.

3.3. Comunicações Seguras

De seguida apresento as duas técnicas criptográficas que envolvem uma chave de sessão.

3.3.1 Envelope Digital

O envelope digital consiste nos seguintes passos:

1. Entidade “A” cria a chave simétrica secreta K
2. Entidade “A” cifra K com a chave pública da entidade “B”
3. Entidade “A” envia o criptograma (K cifrada) para a entidade “B”
4. Entidade “B” decifra o criptograma com a sua chave privada e obtém K

3.3.2 Station To Station

O protocolo *Station-to-Station* consiste nos seguintes passos:

1. “A” gera um número aleatório grande x
2. “A” calcula $X = g^x \pmod n$ e envia-o ao “B”
3. “B” gera um número aleatório grande y
4. “B” calcula $Y = g^y \pmod n$ e envia-o ao “A”
5. Ambos conseguem calcular $K = X^y \pmod n = Y^x \pmod n$

6. Acordada a chave de sessão K, os agentes assinam digitalmente o par ordenado (X, Y)
7. Estas assinaturas são trocadas entre os agentes, cifradas com a chave acordada
8. Caso as assinaturas sejam recuperadas e verificadas com sucesso o protocolo terminou com sucesso

Os números "g" e "n" são parâmetros públicos do protocolo. Do passo 1 até ao passo 5, os intervenientes acordam a chave secreta K. Estes passos são exclusivos do protocolo de acordo de chaves *Diffie-Hellman*, que é vulnerável ao ataque "*man-in-the-middle*". Por isso que surgem os passos seguintes (6, 7 e 8), que pretendem validar as identidades dos intervenientes eliminado a possibilidade deste ataque ter sucesso. Naturalmente a validação das entidades dos intervenientes tem de ter um processo de certificação envolvido.

Se seguida apresento e descrevo em três excertos, o código Java necessário para implementar o referido protocolo.

```
1.  AlgorithmParameterGenerator apg = AlgorithmParameterGenerator.getInstance("DH");
2.  apg.init(512);
3.  AlgorithmParameters ap = apg.generateParameters();
4.  DHParameterSpec dhsp = (DHParameterSpec) ap.getParameterSpec(DHParameterSpec.class);
5.  SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
6.  BigInteger x = new BigInteger(512, random);
7.  BigInteger gx = dhsp.getG().modPow(x, dhsp.getP());
8.  oos.writeObject(gx);
9.  BigInteger gy = (BigInteger) ois.readObject();
10.   BigInteger k = gy.modPow(x, dhsp.getP());
```

Neste primeiro excerto podemos observar nas linhas 1, 2, 3 e 4 que está a ser instanciada uma variável com os parâmetros públicos do protocolo. Nas linhas 5 e 6 é gerado o número secreto "x", sendo este aleatório e com 512 bits. Na linha 7 é efectuada a operação exponencial para determinar "X", de seguida são trocados os valores "X" e "Y". Na última linha é efectuada a operação exponencial para determinar a chave secreta K.

```
11.   DESKeySpec sks = new DESKeySpec(k.toByteArray());
12.   SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
13.   SecretKey sk = skf.generateSecret(sks);
14.   Cipher enc = Cipher.getInstance("DES/ECB/PKCS5Padding","IAIK")
15.   enc.init(Cipher.ENCRYPT_MODE, sk);
16.   Cipher dec = Cipher.getInstance("DES/ECB/PKCS5Padding");
17.   dec.init(Cipher.DECRYPT_MODE, sk);
18.   Signature sig = Signature.getInstance("SHA1withRSA");
```

```
19.   sig.initSign(mypri);
20.   sig.update(gx.toByteArray());
21.   sig.update(gy.toByteArray());
22.   byte[] assinaturaXY = sig.sign();
23.   SealedObject sigXY = new SealedObject(assinaturaXY, enc);
24.   oos.writeObject(sigXY);
```

Nas linhas 11, 12, 13 é gerada a chave de sessão para usar no algoritmo criptográfico DES. Nas linhas 14 e 15 está a ser instanciada a variável "enc", que servirá para cifrar usando o algoritmo DES no modo ECB. Nas linhas 16 e 17 está a ser realizada a tarefa análoga, mas neste caso para decifrar. Na linha 18 é obtida uma instância do objecto "*Signature*" para de seguida assinar e verificar usando o algoritmo de chave pública RSA e função de *hash* SHA1. Na linha 19 é inicializada com a chave privada. Nas linhas 20, 21 e 22 é assinado o par "X, Y". Por fim é criado o objecto "*SealedObject*", que contem a assinatura cifrada. Por último é enviado o objecto para o outro interveniente.

```
25.   SealedObject sigXY2 = (SealedObject) ois.readObject();
26.   byte[] assinaturaXY2 = (byte[]) sigXY2.getObject(dec);
27.   sig.initVerify(pub);
28.   sig.update(gx.toByteArray());
29.   sig.update(gy.toByteArray());
30.   if( sig.verify(assinaturaXY2) ) {
31.      return true;
32.   }
33.   else {
34.      return false;
35.   }
```

Neste extracto, a linha 25 serve para ler o objecto "*SealedObject*" que contem o par (X,Y) enviado pelo outro interveniente. Na linha seguinte é obtido o conteúdo do objecto anterior. Nas linhas seguintes, é verificada a validade da assinatura usando a chave pública do outro interveniente.

3.4. Certificados X.509

As duas entidades mais importantes no contexto dos certificados são as Autoridades de Certificação e os Clientes. Os papéis que desempenham são descritos de seguida:

- **Autoridades de certificação:** Emitem certificados, a pedido dos clientes, que podem ser também outras CA's. Revogam certificados, disponibilizando essa informação via

CRL (*Certificate Revocation Lists*) ou pedidos OCSP (*Online Certificate Status Protocol*). Fazem a gestão dos certificados emitidos.

- **Clientes:** Criam pedidos de certificados para apresentar junto de uma CA, estando contido neste pedido a sua chave pública. Verificam a validade de um certificado (chave pública) que pretendem usar numa comunicação segura, quer seja para cifrar uma mensagem ou validar que o texto foi assinado pela entidade que tem a chave privada correspondente. Fazem a gestão da(s) sua(s) chave(s) privada(s).

3.4.1 Criar Certificado

Os passos apresentados de seguida são realizados pela CA e são apenas os passos indispensáveis para a criação de um certificado X.509, pois estes certificados podem conter mais informação, nomeadamente extensões específicas para um determinado fim.

```
1. KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
2. SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
3. kpg.initialize(1024,sr);
4. KeyPair clikp = kpg.generateKeyPair();
5. PrivateKey cliPri = clikp.getPrivate();
6. PublicKey cliPub = clikp.getPublic();
```

O bloco anterior consiste na geração do par de chaves do algoritmo de chave pública RSA, com 1024 bits.

```
7. X509Certificate x509cert = new X509Certificate();
8. GregorianCalendar date = (GregorianCalendar)Calendar.getInstance();
9. x509cert.setValidNotBefore(date.getTime());
10.   date.add(Calendar.MONTH, p_duracao);
11.   x509cert.setValidNotAfter(date.getTime());
```

Na linha 7 está a ser criado um objecto que representa um certificado X.509. Nas linhas seguintes está a ser definida a data de validade do certificado, data de início e fim.

```
12.   x509cert.setIssuerDN(p_certCA.getSubjectDN());
```

Na linha 12 é definido a identidade da autoridade de certificação que vai assinar este certificado. Esta informação é obtida a partir do certificado da CA.

```
13.   Name subject = new Name();
14.   subject.addRDN(ObjectID.country, p_pais);
```

```
15.   subject.addRDN(ObjectID.organization, p_organization);
16.   subject.addRDN(ObjectID.organizationalUnit, p_organizationUnit);
17.   subject.addRDN(ObjectID.commonName, p_nome);
18.   x509cert.setSubjectDN(subject);
```

Este bloco consiste na definição da identidade do futuro titular do certificado. A identidade deste é composta pelo país, organização, unidade organizacional e nome.

```
19.   x509cert.setSerialNumber(p_serial);
20.   x509cert.setPublicKey(cliPub);
21.   x509cert.sign(AlgorithmID.sha1WithRSAEncryption, p_mypri);
```

Por fim, o bloco anterior define o número de série do certificado (para controlo), associa a chave pública ao certificado e finalmente assina o certificado com a chave privada da CA.

Se o cliente apresentar um pedido de certificado, os passos que envolvem a geração do par de chaves (pública e privada) e a definição dos atributos que identificam o titular do certificado são realizados pelo cliente, estando contido no pedido a identidade do titular e sua chave pública. Ficando a CA responsável por efectuar os restantes passos.

3.4.2 Validar Certificado

O processo de validação de um certificado pode envolver até três passos, sendo sempre obrigatório efectuar o primeiro. Os dois últimos complementam o primeiro e são efectuados apenas quando existir toda a informação necessária, e.g. endereço do servidor OCSP.

1. **Validar cadeia de certificados:** Este passo é obrigatório e consiste na verificação da validade do certificado, nomeadamente a data e assinatura, e validade dos certificados que estejam na sua cadeia de certificação, até ao certificado de confiança.
2. **Verificar que não está na CRL:** Este passo consiste em consultar a CRL da CA que emitiu o certificado, para verificar se este pertence a CRL.
3. **Verificar o resultado do pedido OCSP:** Este passo permite obter informação sobre um certificado de forma imediata (*online*). Caso o certificado tenha sido revogado, o servidor OCSP responderá "*Revoked*".

```
1. //construir a cadeia de certificados: p_Array_cert
2. SimpleChainVerifier scv = new SimpleChainVerifier();
3. scv.addTrustedCertificate( p_trusted_cert );
4. if (scv.verifyChain( p_Array_cert ) ) {
5. { return true; }
6. else { return false; }
```

O bloco anterior apenas demonstra o primeiro passo na verificação de um certificado. Ou seja, depois de criada a cadeia de certificação é verificada a validade dos certificados da cadeia. Para tal, é necessário criar um objecto "*SimpleChainVerifier*" (linha 2), depois fornecer o certificado de confiança (linha 3) e por fim fornecer e verificar a cadeia de certificados que validam o certificado (linha 4, 5, e 6). Naturalmente, o certificado que se pretende validar é o primeiro da cadeia.

4. Conclusões

As principais conclusões que resultam duma análise prática da JCA podem ser divididas nas suas vantagens e desvantagens. Uma vantagem é a sua independência na implementação dos algoritmos criptográficos. Pois podemos escolher facilmente o que mais nos convém. No entanto, por vezes esta vantagem traduz-se em mais algumas linhas de código, pois é necessário em situações pontuais testar a implementação que está a ser usada. Nomeadamente na mudança da classe "`Object`" para a classe "`X509Certificate`". Outra vantagem é o facto de os principais algoritmos criptográficos estarem contemplados, possibilitando o desenvolvimento de qualquer aplicação segura. Uma desvantagem no desenvolvimento de aplicações seguras em JCA está na necessidade de alguns conhecimentos na área da criptografia. Mas penso que essa desvantagem pode ser encontrada em qualquer linguagem de programação, pois é sempre preciso perceber todos os conceitos envolvidos. Por fim, o nível de detalhe que a JCA fornece é bom por uma lado, pois permite grande controlo, mas por outro lado pode originar falhas graves na segurança da aplicação.

Referências

[01] **http://java.sun.com/j2se/1.4/docs/api/index.html**
API (Application Programming Interface) specification for the Java 2 Platform, Standard Edition, version 1.4

[02] **http://java.sun.com/security/index.html**
Java Security Homepage

[03] **http://jcewww.iaik.at/**
IAIK JCE 3.0 API Documentation

[04] Apontamentos Teóricos da Disciplina: Segurança e Privacidade em Sistemas de Armazenamento e Transporte de Dados 2006/2007

[05] Apontamentos Teóricos da Disciplina: Criptografia Aplicada 2003/2004