Módulo 7

Segurança

Lição 7

Criptografia

Versão 1.0 - Jan/2008

Autor
Aldwin Lee
Cheryl Lorica

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Angelo de Oliveira
Bruno da Silva Bonfim
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Guilherme da Silveira Elias
Leandro Souza de Jesus
Lucas Vinícius Bibiano Thomé
Luiz Fernandes de Oliveira Junior
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach e Vinícius G. Ribeiro (Especialista em Segurança)
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Criptografia é o estudo da transformação de mensagens claras para torná-las secretas. Nos tempos modernos, tem se tornado um ramo da teoria da informação como um estudo matemático da informação e especialmente sua transmissão de um lugar para outro. É uma parte central de diversos campos da segurança da informação e assuntos relacionados, particularmente, autenticação e controle de acesso. Um dos objetivos primários da criptografia é esconder o significado das mensagens, mas, geralmente, sem esconder sua existência. Criptografia também contribui para a ciência da computação, particularmente em técnicas usadas em computadores e redes de segurança para garantir funcionalidades, tais como, o controle de acesso a informação de forma confidencial.

Dois conceitos importantes que devem ser analisados são **criptografar** e **decriptar**. Criptografar é, essencialmente, o processo pelo qual certas informações dentro de arquivos ou programas são alteradas por algoritmos matemáticos. Uma mensagem criptografada é chamada de ***ciphertext***. Decriptar, por outro lado, é o processo de voltar um *ciphertext* a mensagem original. Criptografar e decriptar são feitas através do uso de uma seqüência de caracteres conhecidos como chaves.

Ao final desta lição, o estudante será capaz de:

- Debater sobre os tipos de algoritmos criptográficos
- Conhecer as classes da arquitetura de criptografia Java

2. Algoritmos Criptográficos

Esses são os três principais tipos de algoritmos baseados na chave em uso: algoritmos simétricos, algoritmos assimétricos e algoritmos híbridos.

Algoritmos simétricos são aqueles em que a chave para criptografar e a chave para decriptar pode ser calculadas isoladamente. Em muitos casos, a chave para criptografar e decriptar são as mesmas. Algoritmos simétricos requerem que o remetente e o receptor concordem primeiramente qual será a chave, antes que a mensagem seja enviada. Um dos algoritmos simétricos mais usados é o *Data Encryption Standard* ou DES. O padrão DES usa a mesma chave para criptografar e decriptar baseada no uso do operador "ou exclusivo" (XOR) e em operações de troca de *bits*. A maior vantagem desse algoritmo é a sua velocidade e popularidade. Sua maior limitação é que tem um tamanho de chave relativamente pequeno (56 *bits*) e a segurança da comunicação prioriza o requisito de chaves compartilhadas.

O padrão DES foi criado em 1976, e previa-se o seu uso até 1981. Foi descontinuado em 2002, sendo substituído pelo AES – *Advanced Encryption Standard*. Há diversas classes Java que empregam esse algoritmos, a grande vantagem deste novo padrão é o tamanho de sua chave (256 *bits*).

Algoritmos assimétricos (também chamados de algoritmos de chave pública) diferem dos simétricos no fato de que a chave para criptografar é diferente da chave para decriptar. A chave para criptografar não é secreta e pode ser pública desde que não seja usada para decifrar a mensagem. A chave para decriptar é usada para decodificar a mensagem, e deve pertencer ao receptor. A chave para criptografar é chamada de chave pública e a chave para decriptar é chamada de chave particular. A seguir, temos uma lista dos algoritmos criptográficos assimétricos mais comuns:

- RSA

RSA (*Rivest, Shamir and Adleman*), a sigla refere-se aos autores do algoritmo. É um algoritmo popular usado tanto para criptografar quanto para assinaturas digitais. O algoritmo é baseado na dificuldade em fatorar, ou seja, encontrar os "Fatores Primos" de um determinado número.

- DSA

DSA (*Digital Signature Algorithm*), ou seja, Algoritmo de Assinatura Digital. É usado apenas para assinaturas digitais.

- Diffie-Hellman

Inventado em 1976 por Whitfield Diffie e Martin Hellman, esse algoritmo é usado primariamente na distribuição de chaves.

- RC2 e RC4

RC (*Rivest Cipher*) foi desenvolvido pelo conhecido criptógrafo *Ron Rivest*. O RC2 é uma chave variável *block cipher*, com tamanho de 64 *bits*, enquanto o RC4 é uma chave variável *Stream Cipher*.

Algoritmo híbrido mescla a capacidade das duas classes anteriores. Essencialmente, essa classe de algoritmo usa pares de chaves pública e particular (assimétricas) para autenticar e concordar com uma chave de sessão. A criptografia é então feita usando um algoritmo simétrico com aquela chave de sessão.

3. Arquitetura de Criptografia Java

Java Cryptography Architecture (Arquitetura de Criptografia Java), ou JCA, refere-se ao *framework* para acessar e desenvolver a funcionalidade criptográfica para a plataforma Java. Inclui componentes da biblioteca *Java Security* relacionados à criptografia, assim como um conjunto de convenções e especificações. A JCA foi criada com foco na independência de implementação e de algoritmo e interoperabilidade.

3.1. Classes Provider

```
java.security.Provider
```

É uma classe abstrata para implementação de algoritmos criptográficos específicos. Refere-se ao **provedor de segurança** que será implementado pela aplicação. Estas implementações podem incluir: implementações nos algoritmos da assinatura digital, algoritmos de *Message Digest*, algoritmos de criptografia e esquemas de *padding*. Os seguintes métodos estão incluídos nesta classe:

```
public String getName()
```

Retornar o nome do provedor de segurança.

```
public double getVersion()
```

Retornar a versão do número do provedor de segurança.

```
public String getInfo()
```

Retornar as informações do provedor de segurança.

```
public String toString()
```

Retornar uma String específica do provedor de segurança, contendo seu nome e versão.

Ao criarmos classes para realizar operações de segurança, devemos estender a classe *Provider* e registrar essa classe na infra-estrutura de segurança. Apesar da classe *Provider* ser abstrata, nenhum de seus métodos são abstratos. Isso significa que para implementar uma classe real, de uma certa forma, tudo o que será necessário é estender a classe *Provider* e fornecer um construtor apropriado. A subclasse deve implementar um construtor, já que não existe nenhum construtor padrão na classe *Provider*. Esse construtor deve ser fornecido conforme a seguinte assinatura:

```
protected Provider(String name, double version, String info)
```

Construir um provedor de segurança com um determinado nome, versão e informações.

A seguir temos um exemplo de implementação de uma classe *Provider.*

```
import java.security.*;

public class SomeProvider extends Provider {
  public SomeProvider() {
    super("Someone", 1.0, "Someone Security Provider v1.0");
  }
}
```

Para adicionar qualquer funcionalidade ao provedor de segurança, colocaremos algumas associações, tais como:

```
import java.security.*;

public class SomeProvider extends Provider {
  public SomeProvider( ) {
    super("Someone", 1.0, "Someone Security Provider v1.0");
    put("KeyGenerator.XOR", "xxx.yyy.SomeXORKeyGenerator");
```

```
        put("KeyPairGenerator.XYZ","xxx.yyy.SomeKeyPairGenerator");
        put("KeyFactory.XYZ", "xxx.yyy.SomeKeyFactory");
        put("MessageDigest.XYZ", "xxx.yyy.SomeMessageDigest");
        put("Signature.XYZwithSHA", "xxx.yyy.SomeSignature");
        put("Cipher.XOR", "xxx.yyy.SomeXORCipher");
        put("KeyManagerFactory.XYZ", "xxx.yyy.SomeSSLKeyManagerFactory");
      }
    }
```

É necessário que o provedor de segurança mapeie o nome do *engine* e do algoritmo com o nome da classe que implemente essa determinada operação.

3.2. Classe Security

Essa classe é usada para administrar os provedores de segurança e centralizar as propriedades e métodos de segurança.

```
java.security.Security
```

Essa classe é definida como *final*, contém métodos estáticos e o construtor *private*, portanto, nunca poderá ser estendida ou gerar objetos. Estes são os métodos desta classe:

```
public static int addProvider(Provider provider)
```

Adicionar um novo provedor de segurança à lista de provedores. O provedor é adicionado ao final de uma coleção interna de provedores.

```
public static int insertProviderAt(Provider provider, int position)
```

Inserir um novo provedor de segurança para a lista interno de provedores. O provedor é adicionado em uma posição especifica, outros provedores tem seu índice modificado se necessário para criar espaço para esse provedor.

```
public static int removeProvider(String name)
```

Remover um determinado provedor de segurança através do seu nome passado como argumento.

```
public static Provider[] getProviders()
```

Retornar uma cópia da lista de provedores de segurança que são controlados pela classe. Observe que essa é apenas uma cópia da lista, retirar ou reordenar seus elementos não causa nenhum efeito na classe *Security*.

```
public static Provider getProvider(String name)
```

Retornar um provedor de segurança através de um determinado nome passado como argumento. Se o nome do provedor não estiver na lista contida na classe Security, esse método retorna *null*.

```
public static String getProperty(String key)
```

Obter a propriedade da classe *Security* com uma determinada chave. Essas propriedades contidas na classe *Security* foram lidas do arquivo *java.security*. O arquivo *java.security* tem várias outras propriedades que também podem ser recuperadas por esse método.

```
public static void setProperty(String property, String value)
```

Estabelecer uma dada propriedade para um determinado valor passado como argumento do método.

```
public static String getAlgorithmProperty(String algName, String propName)
```

Procurar em todos os provedores de segurança por uma determinada propriedade na forma *Alg.propName.algName* e retorna com o primeiro correspondente encontrado.

Como um exemplo da utilização da classe *Security*, veremos uma classe que permite visualizar uma lista com todos os provedores de segurança em uma JVM em particular:

```
import java.security.*;
import java.util.*;

public class ExamineSecurity {
  public static void main(String args[]) {
    try {
      Provider p[] = Security.getProviders( );
      for (int i = 0; i < p.length; i++) {
        System.out.println(p[i]);
        for (Enumeration e = p[i].keys(); e.hasMoreElements( );)
          System.out.println("\t" + e.nextElement( ));
      }
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

3.3. Classes Engine

As classes de *Engine* (também conhecidas como *Service Provider Interfaces* ou *SPI*) são desenvolvidas para que os usuários possam empregar a terça parte dos *Security Providers*. Fornecem funcionalidade para calcular uma *message digest* de um dado especificado, assinar, verificar as assinaturas digitais e gerar pares de chaves (pública e particular). As classes *Engine* incluem as seguintes classes:

`java.security.MessageDigest`

Implementar operações para criar e verificar um *message digest*.

`java.security.Signature`

Fornecer um *Engine* para criar e verificar assinaturas digitais.

`java.security.KeyPairGenerator`

Gerar e fornecer informações sobre os pares de chaves.

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.