Demoiselle Signer

Componente para certificação digital ICP-BRASIL

Ednara Oliveira

Emerson Saito

Fabiano Kuss

Humberto Pacheco

José Rene Campanario

Julian Santos

Sobre o Demoiselle Signer

O Demoiselle Signer é um componente para facilitar a geração de assinatura digital. O componente implementa o padrão de assinatura conforme as política da ICP-Brasil, de acordo com as *resoluções da ICP-BrasiL* [http://www.iti.gov.br/legislacao/documentos-principais].

O componente é sub-dividido em módulos de acordo com suas funcionalidades:

- **core:** fornece as interfaces basicas de todas as funcionalidades, como acesso ao certificado (token, arquivo), operações de carregamento e validações de certificado e API para extraçao de dados de um certificado ICPBrasil
- **policy-impl-cades:** permite geração e validação de assinaturas digitais (conforme uma política) no formato CADES
- **policy-engine:** mecanismo para carregamento das políticas de assinaturas definidas pela ICP-BRASIL
- **chain-icp-brasil:** possui funcionalidade para montagem das cadeias de autoridades certificadores ICPBrasil válidas
- **timestamp:** disponibiliza as funcionalidades para obtenção de carimbos de tempo (fornecidos por uma autoridade de carimbo de tempo)
- **cryptography:** provê funcionalidades de criptografia

Funcionalidades ainda não implementadas no componente:

- Assinatura no padrão XADES (assinatura em XML)
- Assinatura no padrão PADES (assinatura em PDF)

Nota

Caso queira baixar uma versão desta documentação em formato PDF clique

aqui: [http://demoiselle.io/signer/docs/pdf/signer-reference-3.2.6.pdf]

Parte I. Core

Este componente provê uma API para facilitar o tratamento de Certificados Digitais em aplicações Java. Seus objetivos são o carregamento, validação, e obtenção de dados para certificados digitais.

Configuração do Signer-Core

1.1. Uso do componente

Para utilizar o componente *Signer-Core* num projeto Java, basta adicionar a sua dependência no arquivo `pom.xml`, conforme o seu gerenciador de projetos:

- *Apache-Maven* [https://maven.apache.org/]

```
      <dependency>
    <groupId>org.demoiselle.signer</groupId>
    <artifactId>core</artifactId>
    <version>3.2.6</version>
   </dependency>
```

- *Apache Buildr* [https://buildr.apache.org/]

```
'org.demoiselle.signer:core:jar:3.2.6'
```

- *Apache Ivy* [http://ant.apache.org/ivy/]

```
<dependency org="org.demoiselle.signer" name="core" rev="3.2.6" />
```

- *Groovy Grape* [http://docs.groovy-lang.org/latest/html/documentation/grape.html]

```
@Grapes(@Grab(group='org.demoiselle.signer', module='core', version='3.2.6'))
```

- *Gradle/Grails* [https://github.com/grails/grails-gradle-plugin]

```
<dependency org="org.demoiselle.signer" name="core" rev="3.2.6" />
```

- *Scala SBT* [http://www.scala-sbt.org/]

```
libraryDependencies += "org.demoiselle.signer" % "core" % "3.2.6"
```

- *Leiningen* [https://leiningen.org/]

```
[org.demoiselle.signer/core "3.2.6"]
```

Caso não esteja utilizando nenhum outro tipo de gerenciador (estava morando numa caverna nos últimos dez anos), pode baixar o .jar do repositório:

http://repo1.maven.org/maven2/org/demoiselle/signer/core/

Funcionalidades relativas ao Certificado

O componente de segurança disponibiliza o CertificateManager que permite manipular objetos de certificado X.509 para extrair informações e validar seu conteúdo. Para trabalhar com o CertificateManager basta instanciá-lo passando o objeto X.509 no construtor. Se não for informado, serão carregados os validadores *CRLValidator* e *PeriodValidator* . A validação ocorre no momento da instanciação do objeto CertificateManager. Segue abaixo a criação do CertificateManager.

```
CertificateManager cm = new CertificateManager(x509);
```

É possível desativar o carregamento dos validadores mudando a instrução para:

```
CertificateManager cm = new CertificateManager(x509, false);
```

Caso seja necessário implementar os próprios validadores de certificado basta mudar a instrução para:

```
/* Neste caso os validadores padrao tambem serao carregados. */
CertificateManager cm = new CertificateManager(x509, validator1, validator2, validatorN);
```

ou

```
/* Neste caso os validadores padrao nao serao carregados. */
CertificateManager cm = new CertificateManager(x509, false, validator1, validator2, validatorN);
```

É possível também criar um CertificateManager e passar um arquivo do tipo PEM que represente um objeto X509Certificate, conforme mostrado abaixo.

```
File certFile = new File("certificado.pem");
CertificateManager cm = new CertificateManager(certFile);
```

Também é possível criar um CertificateManager que carrege um certificado direto de um token.

```
String pinNumer = "pinNumber do token";
CertificateManager cm = new CertificateManager(pinNumber);
```

2.1. O Certificado Digital

2.1.1. Extração de Informações utilizando anotações

Os certificados no formato X.509 podem conter várias informações armazenadas que podem ser obtidas através de um OID (Object Identifier). OID são usados extensivamente em certificados de formato X.509, como por exemplo, para designar algoritmos criptográficos empregados, políticas de certificação e campos de extensão. Cada autoridade certificadora pode definir um conjunto de OID para armazenar suas informações. O componente de segurança implementa extensões de OID para ICP-Brasil e Default.

Para extrair informações basta criar uma classe com os atributos que se deseja preencher com informações do certificado X.509. Cada atributo deve ser anotado com o seu OIDExtension. Para executar a carga das informações basta passar a classe/objeto para o CertificateManager.

```
class Cert {

    @ICPBrasilExtension(type=ICPBrasilExtensionType.CPF)
    private String cpf;

    @ICPBrasilExtension(type=ICPBrasilExtensionType.NOME)
    private String nome;

    @DefaultExtension(type=DefaultExtensionType.CRL_URL)
    private List<String> crlURL;

    public String getCpf() {
        return cpf;
    }

    public String getNome() {
        return nome;
    }

    public List<String> getCrlURL() {
        return crlURL;
    }

}
```

Em seguida basta efetuar o carregamento da classe.

```
CertificateManager cm = new CertificateManager(x509);
Cert cert = cm.load(Cert.class);
```

2.1.1.1. DefaultExtension

Os OIDs default de um certificado que podem ser obtidos por essa anotação são:

- BEFORE_DATE
- AFTER_DATE

- CERTIFICATION_AUTHORITY
- CRL_URL
- SERIAL_NUMBER
- ISSUER_DN
- SUBJECT_DN
- KEY_USAGE
- PATH_LENGTH
- AUTHORITY_KEY_IDENTIFIER
- SUBJECT_KEY_IDENTIFIER

2.1.1.2. ICPBrasilExtension

Os OIDs definidos pela ICP-Brasil que podem ser obtidos são:

- CPF
- CNPJ
- CEI_PESSOA_FISICA
- CEI_PESSOA_JURIDICA
- PIS_PASEP => Ver NIS
- NOME
- NOME_RESPONSAVEL_PESSOA_JURIDICA
- EMAIL
- DATA_NASCIMENTO
- NUMERO_IDENTIDADE
- ORGAO_EXPEDIDOR_IDENTIDADE
- UF_ORGAO_EXPEDIDOR_IDENTIDADE
- NUMERO_TITULO_ELEITOR
- ZONA_TITULO_ELEITOR
- SECAO_TITULO_ELEITOR
- MUNICIPIO_TITULO_ELEITOR
- UF_TITULO_ELEITOR
- NOME_EMPRESARIAL
- TIPO_CERTIFICADO

- NIVEL_CERTIFICADO

Nota

Em computação, um identificador de objeto, do inglês object identifier (OID), é um identificador usado para nomear um objeto (comparar com URN).[1] Estruturalmente, um OID consiste de um nó em um espaço de nomes atribuído hierarquicamente, formalmente definido usando o padrão ASN.1 do ITU-T, x.690. Números sucessivos de nós, começando na raiz da árvore, identificam cada nó na árvore. Projetistas configuram novos nós registrando-os sob a autoridade de registro de nós.[2] A raiz da árvore contem os três seguintes arcos:

- I: ITU-T
- II: ISO
- III: conjunto-iso-itu-t

Em programação de computador, um identificador de objeto geralmente toma a forma de um inteiro ou ponteiro específico de implementação que identifica unicamente um objeto. Entretanto, IDOs são uma abordagem específica para criação globalmente de identificadores de objeto únicos em um sistema distribuído. Referências

- *[1]* [https://standards.ieee.org/develop/regauth/tut/oid.pdf]
- *[2]* [http://www.alvestrand.no/objectid]

Fonte: *https://pt.wikipedia.org/wiki/Identificador_de_objeto*

2.1.2. Extração de Informações utilizando Classes

Uma outra maneira de obter os valores necessários do certificado é através das classes de apoio fornecidas pelo componente. Caso deseje obter apenas informações, básicas, podemos utilizar a classe `BasicCertificate`.

A seguir temos o exemplo de utilização, onde passamos um certificado para a classe e em seguida obtemos exibimos algumas informações no console.

```
BasicCertificate bc = new BasicCertificate(certificate);
logger.log(Level.INFO, "Nome.....................[{0}]", bc.getNome());
logger.log(Level.INFO, "E-mail...................[{0}]", bc.getEmail());
logger.log(Level.INFO, "Numero de serie.........[{0}]", bc.getSerialNumber());
logger.log(Level.INFO, "Nivel do Certificado....[{0}]", bc.getNivelCertificado());
```

Para obter informações mais específicas de um certificado de um e-CPF, e-CNPJ ou de equipamento, devemos utilizar a classe `CertificateExtra`.

A seguir temos alguns exemplos de de utilização.

O exemplo a seguir recupera o CPF e o número RIC de um certificado digital do tipo e-CPF.

```
CertificateExtra ce = new CertificateExtra(certificate);
logger.log(Level.INFO, "CPF.....................[{0}]", ce.getOID_2_16_76_1_3_1().getCPF());
```

```
logger.log(Level.INFO,                                                    "RIC.....................
[{0}]", ce.getOID_2_16_76_1_3_9().getRegistroDeIdentidadeCivil());
```

O exemplo a seguir recupera o CNPJ de um certificado digital do tipo e-CNPJ.

```
CertificateExtra ce = new CertificateExtra(certificate);
logger.log(Level.INFO, "CNPJ.....................[{0}]", ce.getOID_2_16_76_1_3_3().getCNPJ());
```

O exemplo a seguir recupera o nome do responsável de um certificado digital do tipo Equipamento.

```
CertificateExtra ce = new CertificateExtra(certificate);
logger.log(Level.INFO, "Nome.....................[{0}]", ce.getOID_2_16_76_1_3_2().getNome());
```

2.2. Validadores

2.2.1. CRLValidator

O CRLValidator verifica se o certificado está na lista de certificados revogados da autoridade certificadora. Cada certificado pode conter uma ou mais links para os arquivos de CRL. O mecanismo de obtenção dos arquivos de crl é implementado pelos Repositórios de CRL.

2.2.2. PeriodValidator

Verifica a data de validade do certificado.

2.3. Repositório de CRL

O Repositório de CRL disponibiliza uma lista de ICPBR_CRL (CRLs padrão ICP Brasil). Esta lista é obtida pelos arquivos de crl referentes a um certificado digital. A obtenção e armazenamentos dos arquivos de crl são implementados de dois modos: Online ou Offine.

2.3.1. Repositório Online

O Repositório Online nao utiliza um diretório para armazenamento dos arquivos crl, efetuando diretamente a consulta no endereço web da crl.

2.3.2. Repositório Offline

O Repositório offline utiliza um diretório onde é mantida uma lista de crl e um arquivo de índice. O arquivos de índice identificam a url do certificado e o nome do arquivos armazenado no file system, como no exemplo abaixo:

```
73bc162ad833c4da45ea60ac8ac016cc=https\://thor.serpro.gov.br/LCR/LCRPRA1.crl
75bc176ad833c4da05ea70ac8ac016ca=http\://ccd.serpro.gov.br/lcr/ACPRv1.crl
43bc194ad833c4da95ea90ac8ac016cb=http\://ccd2.serpro.gov.br/lcr/ACPRv2.crl
```

O diretório e o nome do arquivo de índice devem ser configurados através de chaves informadas em variáveis de ambiente:

- *signer.repository.crl.path*
- *signer.repository.crl.index*

Por padrão essas chaves são inicializadas na seguintes forma:

- *signer.repository.crl.path=/tmp/crls*
- *signer.repository.crl.index=.crl_index*

Programaticamente é possível modificar as propriedades por meio da classe `Configuration.`

```
Configuration config = Configuration.getInstance();
config.setCrlIndex(".crl_index");
config.setCrlPath("/tmp/crls/");
```

Quando o arquivo de crl se encontra com data vencida ou não existe o arquivo no diretório, o repositório Offline realiza o download do arquivo de crl e o armazena no diretório de crl.

2.3.3. Configuração

Para modificar o modo de uso do repositório (online ou offline) deve ser configurada a chave *security.certificate.repository.online.*

O valor padrão é true, mas é possível modificar programaticamente conforme abaixo.

```
Configuration config = Configuration.getInstance();
config.setOnline(false);
```

Funcionalidades relativas ao Keystore

3.1. Introdução

A *RSA Laboratories* [www.rsalabs.com/] definiu algumas especificações de uso de criptografia e assinatura digital conhecidas pelo prefixo *PKCS* [https://brazil.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm]. Duas delas estão relacionadas ao tipo de keystore (chaveiro) que é o recipiente que armazena um par de chaves criptográficas. São elas PKCS#11 e PKCS#12.

PKCS#11 define uma API genérica para acesso a hardware criptográfico, comumente chamados de Token (pendrive) ou Smartcard (cartão e leitora).

PKCS#12 define um formato de arquivo digital usado para guardar chaves privadas acompanhadas de seus certificados digitais.

A linguagem Java suporta a utilização desses formatos e com isso define o que chamamos de KeyStore. Um KeyStore é usado para armazenar um ou mais certificados digitais e também o par de chaves, com isso é possível utilizar os padrões da RSA através da mesma interface. A partir de um objeto KeyStore instanciado é possível navegar pelos certificados digitais contidos no KeyStore por meio dos apelidos (alias) destes certificados.

O componente Demoiselle-Signer visa facilitar o uso destes KeyStores, seja PKCS#11 ou PKCS#12. A maneira como se carrega um KeyStore do tipo PKCS#11, que é um dispositivo em hardware, difere quando trabalhamos com sistemas operacionais diferentes e, em alguns casos,até mesmo versões de JVM.

No ambiente Windows, é possível utilizar a API padrão do sistema operacional de carregamento de KeyStore PKCS#11, chamada *MSCAPI* [https://en.wikipedia.org/wiki/Microsoft_CryptoAPI] que controla os certificados instalados de uma forma mais genérica, mas para isso precisamos também saber a versão da JVM instalada. Isso é necessário porque na versão 1.6 a implementação JCE já comporta o tratamento nativo na plataforma e na versão 1.5 ou inferior é necessário utilizar uma biblioteca para trabalhar com a API nativa do Windows.

Em ambiente Unix-like é possível carregar um KeyStore PKCS#11 a partir de um driver específico, mas é preciso saber o fabricante e o caminho do driver no sistema operacional.

Para carregamento de KeyStore formato PKCS#12, ou seja, em arquivo, o processo de carregamento é o mesmo para os diversos sistemas operacionais.

As funcionalidades do componente estão acessíveis por meio da fábrica *org.demoiselle.signer.core.keystore.loader.factory.KeyStoreLoaderFactory* de objetos do tipo *org.demoiselle.signer.core.keystore.loader.KeyStoreLoader.*

O uso da fábrica é importante, mas não é obrigatório. A importância dela se deve à funcionalidade de descobrir qual a melhor implementação para o carregamento de KeyStore baseando-se em configurações. Utilizando a fábrica não é necessário escrever códigos específicos para um determinado sistema operacional, pois a fábrica identifica qual o sistema operacional e a versão da JVM para fabricar a melhor implementação.

Exemplo de uso da fábrica de objetos KeyStoreLoader

```
KeyStoreLoader keyStoreLoader = KeyStoreLoaderFactory.factoryKeyStoreLoader();
```

Exemplo de uso da fábrica de objetos KeyStoreLoader para KeyStore PKCS#12

```
KeyStoreLoader keyStoreLoader = KeyStoreLoaderFactory.factoryKeyStoreLoader(new File("/usr/
keystore.p12"));
```

3.2. Carregamento de KeyStore PKCS#12

Para carregar um KeyStore a partir de uma arquivo no formato PKCS#12 basta utilizar a classe *org.demoiselle.signer.core.keystore.loader.implementation.FileSystemKeyStoreLoader.*

Abaixo temos exemplos de uso.

```
KeyStore keyStore = (new FileSystemKeyStoreLoader(new File("/usr/
keystore.p12"))).getKeyStore("password");
```

```
KeyStore keyStore = KeyStoreLoaderFactory.factoryKeyStoreLoader(new File("/usr/
keystore.p12")).getKeyStore("password");
```

3.3. Carregamento de KeyStore PKCS#11 em ambiente Linux

Para carregar um KeyStore PKCS#11 basta utilizar a classe *org.demoiselle.signer.core.keystore.loader.implementation.DriverKeyStoreLoader*

Para configuração de drivers favor acessar a área de Configuração do componente em *Seção 3.5, "Lista de Drivers"*.

Abaixo temos exemplos de uso.

```
KeyStore keyStore = (new DriverKeyStoreLoader()).getKeyStore("PIN NUMBER");
```

```
KeyStore keyStore = KeyStoreLoaderFactory.factoryKeyStoreLoader().getKeyStore("PIN NUMBER");
```

Caso se queira instanciar um KeyStore a partir de um driver específico que não esteja na lista de driver configurada, é possível informar o driver como parâmetro para a classe, veja o exemplo:

```
KeyStore keyStore = (new DriverKeyStoreLoader()).getKeyStore("PIN NUMBER", "Pronova", "/usr/
lib/libepsng_p11.so");
```

```
KeyStore keyStore = (new DriverKeyStoreLoader()).getKeyStore("PIN NUMBER", "/usr/lib/
libepsng_p11.so");
```

Importante

Este código também funciona em ambiente Windows, bastando especificar o driver correto a ser utilizado.

3.4. Carregamento de KeyStore PKCS#11 em ambiente Windows

Para carregar um KeyStore utilizando a API nativa do Windows basta utilizar a classe *br.gov.frameworkdemoiselle.certificate.keystore.loader.implementation.MSKeyStoreLoader.*

Abaixo temos exemplos de uso.

```
KeyStore keyStore = (new MSKeyStoreLoader()).getKeyStore(null);
```

```
KeyStore keyStore = KeyStoreLoaderFactory.factoryKeyStoreLoader().getKeyStore(null);
```

Importante

Este recurso só funciona em JVM 1.6 ou superior. Caso deseje executar em um ambiente com o Java mais antigo, desabilite a camada MSCAPI e faça o acesso diretamente pelo driver. Para saber como proceder, consulte *Seção 3.9, "Desabilitar a camada de acesso SunMSCAPI"*.

3.5. Lista de Drivers

Uma das configurações mais importantes desse componente é a lista de drivers PKCS#11 e seus respectivos arquivos. O componente já possui uma lista pré-estabelecida conforme a tabela a seguir.

Tabela 3.1. Drivers pré definidos para Linux

Caminho (Path) do Driver
/usr/lib/libaetpkss.so
/usr/lib/libgpkcs11.so
/usr/lib/libgpkcs11.so.2
/usr/lib/libepsng_p11.so
/usr/lib/libepsng_p11.so.1
/usr/local/ngsrv/libepsng_p11.so.1
/usr/lib/libeTPkcs11.so
/usr/lib/libeToken.so
/usr/lib/libeToken.so.4
/usr/lib/libcmP11.so

Caminho (Path) do Driver
/usr/lib/libwdpkcs.so
/usr/local/lib64/libwdpkcs.so
/usr/local/lib/libwdpkcs.so
/usr/lib/watchdata/ICP/lib/libwdpkcs_icp.so
/usr/lib/watchdata/lib/libwdpkcs.so
/opt/watchdata/lib64/libwdpkcs.so
/usr/lib/opensc-pkcs11.so
/usr/lib/pkcs11/opensc-pkcs11.so
/usr/local/ngsrv/libepsng_p11.so.1.2.2
/usr/lib/libneoidp11.so

Tabela 3.2. Drivers pré definidos para windows

Caminho (Path) do Driver
WINDOWS_HOME/system32/ngp11v211.dll
WINDOWS_HOME/system32/aetpkss1.dll
WINDOWS_HOME/system32/gclib.dll
WINDOWS_HOME/system32/pk2priv.dll
WINDOWS_HOME/system32/w32pk2ig.dll
WINDOWS_HOME/system32/eTPkcs11.dll
WINDOWS_HOME/system32/acospkcs11.dll
WINDOWS_HOME/system32/dkck201.dll
WINDOWS_HOME/system32/dkck232.dll
WINDOWS_HOME/system32/cryptoki22.dll
WINDOWS_HOME/system32/acpkcs.dll
WINDOWS_HOME/system32/slbck.dll
WINDOWS_HOME/system32/cmP11.dll
WINDOWS_HOME/system32/WDPKCS.dll
WINDOWS_HOME/System32/Watchdata/Watchdata Brazil CSP v1.0/WDPKCS.dll
/Arquivos de programas/Gemplus/GemSafe Libraries/BIN/gclib.dll
/Program Files/Gemplus/GemSafe Libraries/BIN/gclib.dll
/system32/SerproPkcs11.dll

Tabela 3.3. Drivers pré definidos para Mac

Caminho (Path) do Driver
/usr/lib/libwdpkcs.dylib
/usr/local/lib/libwdpkcs.dylib
/usr/local/lib/libetpkcs11.dylib
/usr/local/lib/libaetpkss.dylib

3.6. Configuração de Token / SmartCard em tempo de execução

É possível, porém, adicionar mais drivers em tempo de execução. Para isso é necessário trabalhar com a classe `org.demoiselle.signer.core.keystore.loader.configuration.Configuration`.

```
Configuration.getInstance().addDriver("Nome do Driver", "Path do Driver");
```

Este código irá procurar pelo driver e caso ele exista, ou seja, o path do arquivo for válido, o driver será colocado a disposição para futuro uso pelas implementações de carregamento de KeyStore.

Caso seja necessário verificar os drivers já informados, podemos usar a seguinte construção:

```
Map<String, String> drivers = Configuration.getInstance().getDrivers();
```

3.7. Configuração de Token / SmartCard por variáveis de ambiente

Em algumas ocasiões pode ser inviável utilizar o Configuration para adicionar um driver diretamente no código. Neste caso, A API do Java permite definir um arquivo de configuração onde pode-se informar o nome do driver e seus parâmetros. O componente permite a definição desse arquivo por meio de váriáveis de ambiente ou variáveis da JVM.

Abaixo temos o exemplo de como declarar essas configurações.

Tabela 3.4. Configurações do PKCS#11

Ambiente	Variável de Ambiente	Variável JVM
Linux	export PKCS11_CONFIG_FILE=/usr/pkcs11/drivers.config	-DPKCS11_CONFIG_FILE=/usr/pkcs11/drivers.config
Windows	set PKCS11_CONFIG_FILE=c:/pkcs11/drivers.config	-DPKCS11_CONFIG_FILE=c:/pkcs11/drivers.config

A estrutura deste arquivo pode ser encontrada *aqui* [http://java.sun.com/j2se/1.5.0/docs/guide/security/p11guide.html] para Java 1.5, *aqui* [http://java.sun.com/javase/6/docs/technotes/guides/security/p11guide.html] para Java 1.6 ou *aqui* [http://docs.oracle.com/javase/7/docs/technotes/guides/security/p11guide.html] para Java 1.7.

Uma alternativa a este arquivo de configuração é informar o driver diretamente. Para isso basta informar na variável, conforme o exemplo abaixo.

Tabela 3.5. Configurações do PKCS#11

Ambiente	Variável de Ambiente	Variável JVM
Linux	export PKCS11_DRIVER=/usr/lib/libepsng_p11.so	-DPKCS11_DRIVER=/usr/lib/libepsng_p11.so

Ambiente	Variável de Ambiente	Variável JVM
Windows	set PKCS11_DRIVER=/WINDOWS/system32/ngp11v211.dll	-DPKCS11_DRIVER=/WINDOWS/system32/ngp11v211.dll
Linux	export PKCS11_DRIVER=Pronova::/usr/lib/libepsng_p11.so	-DPKCS11_DRIVER=Pronova::/usr/lib/libepsng_p11.so
Windows	set PKCS11_DRIVER=Pronova::/WINDOWS/system32/ngp11v211.dll	-DPKCS11_DRIVER=Pronova::/WINDOWS/system32/ngp11v211.dll

Quando a variável for declarada através da JVM, ela deve ser feita diretamente no painel de controle do JAVA. A seguir demonstramos a configuração para o sistema Windows.

Abra o painel de controle e seleciona e abra o aplicativo "Java".

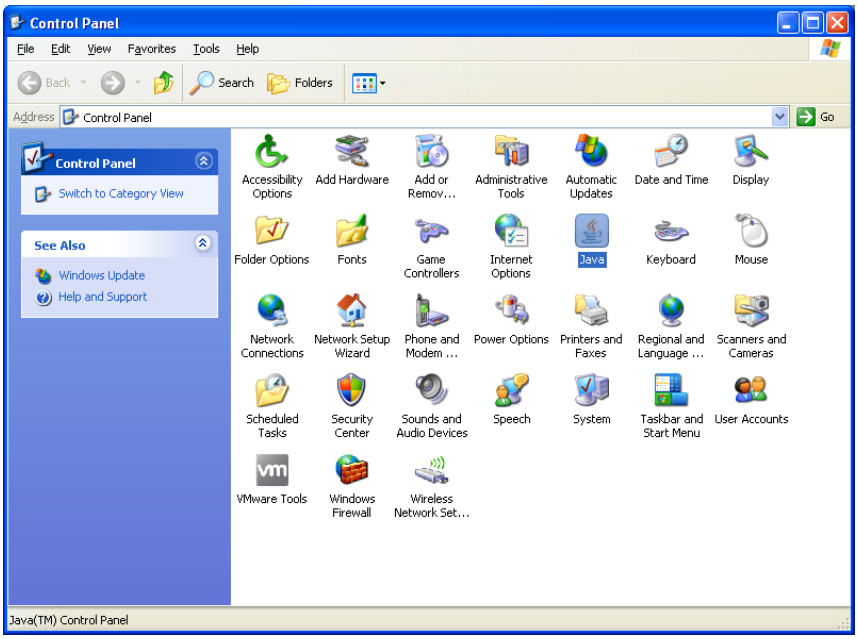

Figura 3.1. Java no Painel de Controle

Selecione a aba "Java" e clique em "View..."

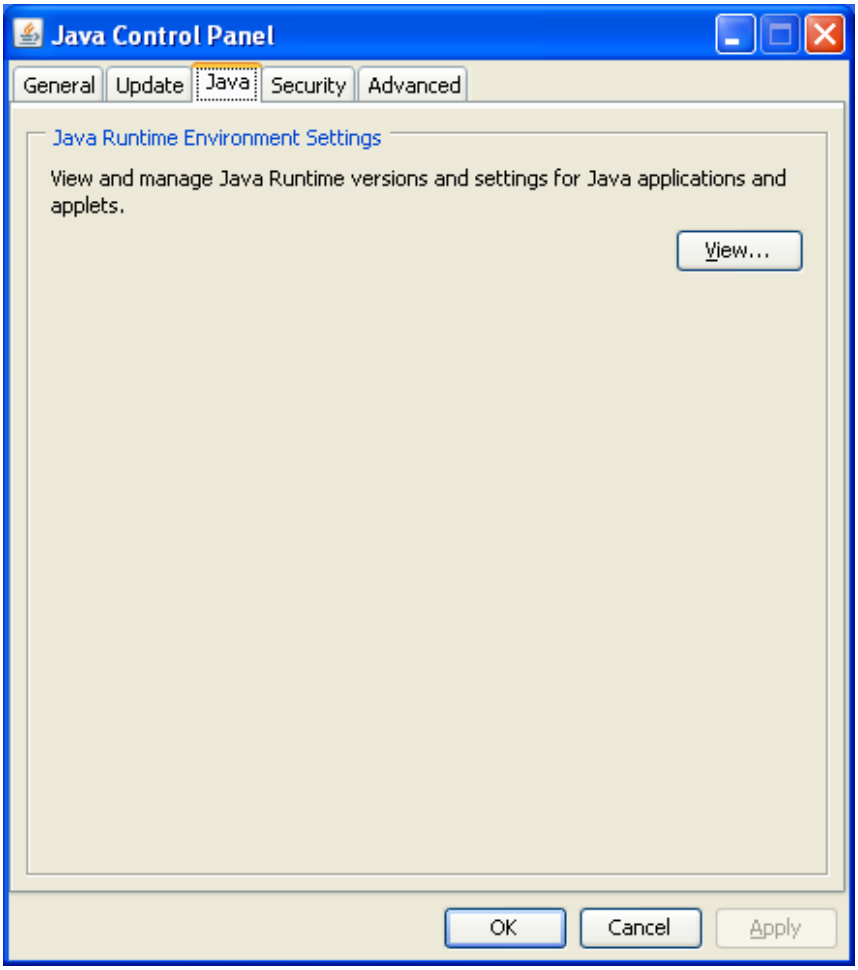

Figura 3.2. Configurações do ambiente Java

Na aba "User", em "Runtime Parameters", coloque a declaração da variável. Em seguida, aplique as alterações.

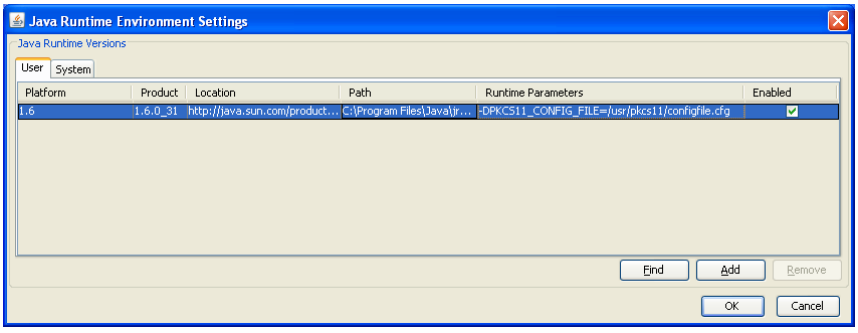

Figura 3.3. Desabilitando a camada MSCAPI

3.8. Configuração de Token / SmartCard por arquivo de configurações

As configurações acima demonstram uma configuração mais refinada para o carregamento de certificados em dispositivos, mas o componente possui um procedimento padrão a ser executado caso se deseje um método mais simplificado. A seguir é explicado como utilizar este mecanismo.

3.8.1. Utilizando certificados armazenados em Disco ou em Token / SmartCard no Windows

O Sistema Operacional Windows fornece uma camada chamada MSCAPI, ou Microsoft CryptoAPI, que facilita o acesso a certificados armazenados em disco ou em dispositivos criptográficos. Neste tipo de acesso, basta que o certificado esteja corretamente instalado e válido, e a própria camada nos fornecerá o driver correto e os meios para acessar os certificados. Até a versão 5 do Java não existia um provedor de acesso para esta camada, mas na versão 6 em diante foi implementado o provedor *SunMSCAPI* para lidar com este tipo de acesso.

3.8.2. Utilizando certificados armazenados em Disco no Linux ou Mac

Ao Contrario do Windows, que utiliza a API da *MS-CAPI* [http://en.wikipedia.org/wiki/Microsoft_CryptoAPI] para abstrair o acesso aos certificados digitais, em outros sistemas operacionais este recurso não existe. Para efetuar o acesso, precisamos criar um arquivo de configuração informando os parâmetros de acesso.

Para viabilizar o acesso em um sistema Não-Windows, deve ser criado um arquivo chamado `drivers.config` dentro do diretório [/home/usuario] com a parametrização mostrada abaixo. Nesta configuração serão carregados todos os certificados A1 que estejam instalados no Firefox.

Para o Ubuntu:

```
name = Provedor
slot = 2
# para 64 bits
library = /usr/lib/x86_64-linux-gnu/nss/libsoftokn3.so
# para 32 bits
# library = /usr/lib/nss/libsoftokn3.so
nssArgs   =   "configdir='/home/<usuario>/.mozilla/firefox/<nnnnnnnn>.default'   certPrefix=''
 keyPrefix='' secmod='secmod.db'  flags='readWrite'"
showInfo=true
```

Para o Mac OS:

```
name = Provedor
slot = 2
library = /Applications/Firefox.app/Contents/MacOS/libsoftokn3.dylib
nssArgs    =    "configdir='/Users/<usuario>/Library/Application    Support/Firefox/Profiles/
<nnnnnnnnn>.default' certPrefix='' keyPrefix='' secmod='secmod.db' flags='readOnly'"
```

Importante

A sequência de caracteres que precede o *.default* , como em *nnnnnnnn.default* é criptografada e, sendo assim, é diferente para cada equipamento e cada usuário.

3.8.3. Utilizando certificados armazenados em Token / SmartCard no Linux ou Mac

Para configurar um token A3, o conteúdo do arquivo `drivers.config` deve ser especificado como mostrado abaixo.

```
name = Provedor
description = Token Pronova ePass2000
library = /usr/local/ngsrv/libepsng_p11.so.1.2.2
```

Importante

Não é possível utilizar certificados A3 e A1 no Linux ou Mac simultaneamente, devendo ser configurado somente UM dos tipos de acesso em um determinado momento.

3.9. Desabilitar a camada de acesso SunMSCAPI

Quando o componente é utilizado em ambiente Windows, o acesso é feito através de uma camada de abstração chamada MSCAPI, que abstrai informações que são particulares de cada token ou smartcard, como os drivers do dispositivo, por exemplo. Este tipo de recurso facilita o uso do componente com dispositivos de diversos fabricantes. Porém, podem existir casos específicos em que o acesso precisa ser feito diretamente ao driver para utilização de funções específicas, como forçar o logout de um token. Para isso, é necessário informar na JVM um parâmetro chamado `mscapi.disabled` passando o valor `true` . Este parâmetro informa que o acesso será feito via PKCS11, sendo necessário informar o arquivo de configuração do token que se deseja acessar. Caso o parâmetro `mscapi.disabled` esteja ausente, o componente fará uso do MSCAPI normalmente.

A seguir demonstramos a configuração para o sistema Windows.

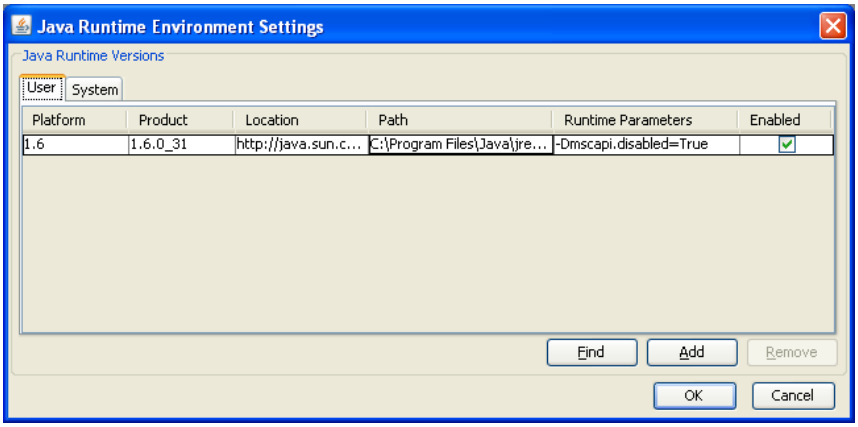

Figura 3.4. Desabilitando a camada MSCAPI

Parte II. Implementação das Políticas CAdES

O componente *Policy-Impl-CAdES* foi desenvolvido para atender às necessidades de assinatura digital no âmbito da ICP-Brasil. Conforme as políticas para o padrão CaDES.

Configuração do Demoiselle Signer

4.1. Instalação do componente

Para utilizar o componente *policy-impl-cades* num projeto Java, basta adicionar a sua dependência no arquivo `pom.xml` , conforme o seu gerenciador de projetos:

- *Apache-Maven* [https://maven.apache.org/]

```
        <dependency>
     <groupId>org.demoiselle.signer</groupId>
     <artifactId>policy-impl-cades</artifactId>
     <version>3.2.6</version>
    </dependency>
```

- *Apache Buildr* [https://buildr.apache.org/]

```
'org.demoiselle.signer:policy-impl-cades:jar:3.2.6'
```

- *Apache Ivy* [http://ant.apache.org/ivy/]

```
<dependency org="org.demoiselle.signer" name="policy-impl-cades" rev="3.2.6" />
```

- *Groovy Grape* [http://docs.groovy-lang.org/latest/html/documentation/grape.html]

```
@Grapes(@Grab(group='org.demoiselle.signer', module='policy-impl-cades', version='3.2.6'))
```

- *Gradle/Grails* [https://github.com/grails/grails-gradle-plugin]

```
<dependency org="org.demoiselle.signer" name="policy-impl-cades" rev="3.2.6" />
```

- *Scala SBT* [http://www.scala-sbt.org/]

```
libraryDependencies += "org.demoiselle.signer" % "policy-impl-cades" % "3.2.6"
```

- *Leiningen* [https://leiningen.org/]

```
<[org.demoiselle.signer/policy-impl-cades "3.2.6"]
```

Caso não esteja utilizando nenhum outro tipo de gerenciador (estava morando numa caverna nos últimos dez anos), pode baixar o .jar do repositório:

http://repo1.maven.org/maven2/org/demoiselle/signer/policy-impl-cades/

Funcionalidades

Este componente provê mecanismos de assinatura digital baseado nas normas ICP-Brasil e implementa mecanismos de assinatura digital em dois formatos: PKCS1 e PKCS7. A maior diferença entre esses dois mecanismos está na forma de envelopamento da assinatura digital, onde o PKCS1 não possui um formato de envelope, sendo o resultado da operação de assinatura a própria assinatura, já o PKCS7 possui um formato de retorno que pode ser binário (especificado na RFC5126) ou XML.

A interface `org.demoiselle.signer.policy.impl.cades.Signer` especifica o comportamento padrão dos mecanismos de GERAÇÃO de assinatura digital. O componente especializa essa interface em mais duas, são elas: `org.demoiselle.signer.policy.impl.cades.pkcs1.PKCS1Signer` para implementações de mecanismos PKCS1 e `org.demoiselle.signer.policy.impl.cades.pkcs7.PKCS7Signer` para implementações de mecanismos de envelopamento PKCS7.

Para as funções de VALIDAÇÃO A interface `org.demoiselle.signer.policy.impl.cades.Checker` especifica as funções de validação de assinatura digital. O componente especializa essa interface em mais duas, são elas: `org.demoiselle.signer.policy.impl.cades.pkcs1.PKCS1Checker` para implementações de mecanismos PKCS1 e `org.demoiselle.signer.policy.impl.cades.pkcs7.PKCS7Checker` para implementações de mecanismos de envelopamento PKCS7.

Este componente, até a presente versão, permite assinar dados representados por um array de bytes. Então se for necessário a assinatura de um arquivo, por exemplo, a aplicação deverá montar um array de bytes com o conteúdo do arquivo e enviar este para o componente poder assiná-lo. Também é possível enviar apenas o Hash já calculado deste arquivo. Veja nas sessões abaixo como fazê-los.

Para assinar um dado através do componente demoiselle-signer é preciso executar alguns passos.

- Ter um conteúdo (ou hash calculado) a ser assinado
- Escolher qual formato de assinatura a ser utilizado PKCS1 ou PKCS7 e qual política ICP-BRASIL
- Fabricar o objeto responsável pela implementação do formato escolhido
- Passar algumas informações para o objeto fabricado como chave criptográfica, algoritmo, política, etc. O formato PKCS7 necessita de mais informações do que o formato PKCS1.
- Assinar o conteúdo (ou hash)

5.1. Assinatura Digital no Formato PKCS1

A seguir temos um fragmento de código que demonstra uma assinatura no formato PKCS1.

```
/* conteudo a ser assinado */
byte[] content = "conteudo a ser assinado".getBytes();

/* chave privada */
PrivateKey chavePrivada = getPrivateKey(); /* implementar metodo para pegar chave privada */

/* construindo um objeto PKCS1Signer atraves da fabrica */
PKCS1Signer signer = PKCS1Factory.getInstance().factory();

/* Configurando o algoritmo */
signer.setAlgorithm(SignerAlgorithmEnum.SHA1withRSA);
```

```
/* Configurando a chave privada */
signer.setPrivateKey(chavePrivada);

/* Assinando um conjunto de bytes */
byte[] signature = signer.doSign(content);
```

5.2. Assinatura Digital no Formato PKCS#7/CAdES sem o conteúdo anexado (detached)

O formato PKCS#7 define o tipo de arquivo para assinatura. Já a ICP-Brasil define um conjunto mínimo de informações básicas para as assinaturas digitais para o padrão CAdES. São elas: Tipo de conteúdo, data da assinatura, algoritmo de resumo aplicado e a política de assinatura. O componente policy-impl-cades já monta o pacote final com três atributos obrigatórios: tipo de conteúdo, data da assinatura e o algoritmo de resumo. Então, para montar um PKCS7 padrão CAdEs ICP-Brasil é necessário informar ao objeto PKCS7Signer qual a política de assinatura a ser aplicada. Uma das formas de gerar a assinatura digital é criar um novo arquivo com a extensão .p7s (PKCS#7) que contém apenas a assinatura, assim independente do arquivo original. Porém para validação da assinatura, será necessário informar tanto o arquivo de assinatura quanto o conteúdo original.

A seguir temos um fragmento de código que demonstra a utilização do pacote PKCS7 padrão.

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificates(certificateChain);
signer.setPrivateKey(privateKey);
byte[] signature = signer.doDetachedSign(this.content);
```

A seguir temos um fragmento de código que demonstra a utilização do pacote PKCS7 padrão com informação da política de assinatura. Neste caso podemos escolher uma das políticas (em vigor) que já acompanham o componente e referem-se à Assinatura Digital padrão CADES.

- AD_RB_CADES_2_2 Refere-se à Assinatura Digital de Referência Básica versão 2.2;
- AD_RT_CADES_2_2 Refere-se à Assinatura Digital de Referência Temporal (com carimbo de tempo) versão 2.2;

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificates(certificateChain);
signer.setPrivateKey(privateKey);
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_2);
byte[] signature = signer.doDetachedSign(this.content);
```

Importante

Caso não seja especificada nenhuma política, o componente assumirá a política padrão AD_RB_CADES_2_2.

5.3. Assinatura Digital no Formato PKCS#7/CAdES com conteúdo anexado (attached)

Identica ao formato apresentado anteriormente, outra das formas de gerar a assinatura digital é incluir todo o conteúdo assinado no novo arquivo com a extensão .p7s (PKCS#7) Desta forma, tanto a assinatura quanto o conteúdo estarão dentro deste arquivo. A sua vantagem é que para validação da assinatura, basta enviar somente este arquivo. Porém, caso o arquivo original seja descartado, para ter acesso ao mesmo, será necessário o uso de um software especializado(como o próprio Demoiselle-Signer)

A seguir temos um fragmento de código que demonstra a utilização do pacote PKCS7 com o conteúdo anexado.

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificates(certificateChain);
signer.setPrivateKey(privateKey);
byte[] signature = signer.doAttachedSign(fileToSign);
```

5.4. Criação de Assinatura Digital enviando apenas o resumo (hash) do conteúdo

Este procedimento visa facilitar a geração de assinaturas digitais em aplicações onde pode haver restrição de trafegar todo o conteúdo do arquivo pela rede, sendo necessário apenas o tráfego dos bytes do resumo do conteúdo original (HASH). Neste caso, é necessário gerar o HASH do conteúdo a ser assinado e passar para o assinador. Ao gerar o HASH, é importante dar atenção ao algoritmo a ser usado, pois na validação da assinatura será considerado o algoritmo da política escolhida. Então, para que esse procedimento funcione corretamente, é necessário escolher o algoritmo do HASH igual ao algoritmo da assinatura digital.

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
/* Gerando o HASH */

java.security.MessageDigest md = java.security.MessageDigest
                  .getInstance(DigestAlgorithmEnum.SHA_256.getAlgorithm());
byte[] hash = md.digest(content);


/* Gerando a assinatura a partir do HASH gerado anteriormente */

PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificate(certificate);
signer.setPrivateKey(privateKey);
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_2);
byte[] signature = signer.doHashSign(hash);
```

Importante

Este procedimento gera o pacote PKCS7 idêntico ao pacote gerado pelo exemplo do tópico 2.2

5.5. Co-Assinatura em arquivo único de assinatura .

5.5.1. Com envio de conteúdo

Por definição, para gerar uma co-assinatura, basta que vários assinantes assinem o mesmo arquivo, gerando assim vários arquivos de assinaturas que estão relacionados ao arquivo original. O componente também oferece uma funcionalidade de co-assinar um arquivo gerando um único arquivo com todas assinaturas. Para isso quando é feita a chamada ao componente é necessário informar tanto o arquivo original (ou seu hash) quanto o arquivo de assinatura que contém a(s) assinatura(s) anterior(es). Conforme exemplificamos abaixo.

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
byte[] signatureFile = readContent("fileSignature.p7s"); /* implementar metodo de leitura de
 arquivo */
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificate(certificate);
signer.setPrivateKey(privateKey);
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_2);
byte[] signature = signer.doDetachedSign(content, signatureFile);
```

5.5.2. Enviando apenas o Hash do conteúdo

Assim, como nas outras forma de gerar a assinatura, é possível fazer o envio do Hash do conteúdo já calculado. Veja no exemplo abaixo:

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
byte[] signatureFile = readContent("fileSignature.p7s"); /* implementar metodo de leitura de
 arquivo */
/* Gerando o HASH */

java.security.MessageDigest md = java.security.MessageDigest
                    .getInstance(DigestAlgorithmEnum.SHA_256.getAlgorithm());
byte[] hash = md.digest(content);

PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificate(certificate);
signer.setPrivateKey(privateKey);
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_2);
byte[] signature = signer.doHashCoSign(hash, signatureFile);
```

Nota

Até a presente versão, o algoritmo de resumo (Digest) para geração da assinatura é o SHA_256, portanto somente os resumos com este algoritimo é que serão gerados e validados corretamente.

5.6. Validação de assinatura PKCS7 sem o conteúdo anexado (dettached)

Como foi visto nas seções anteriores, um do modos de gerar a assinatura é criando um arquivo separado do conteúdo original, e a forma de validação está no fragmento de código abaixo.

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
byte[] signature = readContent("texto.pkcs7"); /* implementar metodo de leitura de arquivo */
CAdESChecker checker = new CAdESChecker();
List<SignatureInformations> signaturesInfo = checker.checkDetattachedSignature(content, signature);
```

O retorno é um objeto do tipo org.demoiselle.signer.policy.impl.cades.SignatureInformations que possui os seguintes atributos

```
public class SignatureInformations {

   private LinkedList<X509Certificate> chain; // cadeia do certificado que gerou a assinatura
    private Date signDate;  // data do equipamento no momento da gera##o das assinatura (n#o
tem validade legal)
    private Timestamp timeStampSigner = null; // Carimbo de tempo da assinatura, quando a
politica utilizada permitir
    private SignaturePolicy signaturePolicy; // Politica ICP-BRASIL usada para gera##o da
assinatura
   private LinkedList<String> validatorErrors = new LinkedList<String>();  // Lista de erros
que por ventura possam ter sido encontrados na valida##o da assinatura
```

5.7. Validação de assinatura PKCS7 com o conteúdo anexado (attached)

A seguir temos um fragmento de código que demonstra a validação uma assinatura PKCS7 com o conteúdo anexado.

```
byte[] signature = readContent("texto.pkcs7"); /* implementar metodo de leitura de arquivo */
CAdESChecker checker = new CAdESChecker();
List<SignatureInformations> signaturesInfo = checker.checkAttachedSignature(signature);
```

5.8. Validação de assinatura PKCS7 enviando apenas o resumo (Hash) do conteúdo

Da mesma forma que possibilitamos a criação da assinatura enviando o resumo (hash) calculado do conteúdo, podemos também fazer a validação da mesma forma. Assim como na geração, é preciso saber qual foi o algoritimo de resumo (hash) que foi usado para gerar a assinatura, pois o mesmo deve ser informado para o método de validação. A seguir temos um fragmento de código que demonstra esta validação.

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arquivo */
byte[] signature = readContent("texto.pkcs7"); /* implementar metodo de leitura de arquiv
CAdESChecker checker = new CAdESChecker();
// gera o hash do arquivo que foi assinado
md = java.security.MessageDigest
            .getInstance(DigestAlgorithmEnum.SHA_256.getAlgorithm());
byte[] hash = md.digest(content);
List<SignatureInformations>                          signaturesInfo                          =
   checker.checkSignatureByHash(SignerAlgorithmEnum.SHA256withRSA.getOIDAlgorithmHash(),   hash,
 signature);
```

5.9. Leitura do conteúdo anexado (Attached) a uma assinatura PKCS7

A seguir temos um fragmento de código que demonstra a extração (recuperação) do conteúdo de um arquivo anexado a uma assinatura PKCS7. Essa funcionalidade pode ser útil quando é necessário mostrar o conteúdo assinado, pois no formato anexado (attached) o conteúdo está empacotado na assinatura. Esta funcionalidade também permite que seja feita a validação da assinatura no momento da extração.

```
byte[] signed = readContent("texto.pkcs7"); /* implementar metodo de leitura de arquivo */
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();

/* Para extrair o conteudo original validando a assinatura */
byte[] content = signer.getAttached(signed, true);

/* Para extrair o conteudo original sem validar a assinatura */
byte[] content = signer.getAttached(signed, false);
```

Nota

No repositório do componente no GitHub há um código de testes unitários para os exemplos acima, *neste link* [https://github.com/demoiselle/signer/blob/master/policy-impl-cades/src/test/java/org/demoiselle/signer/policy/impl/cades/pkcs7/impl/CAdESSignerTest.java]

Exemplos de Uso

A seguir temos alguns exemplos de tarefas normalmente necessárias na utilização do componente.

6.1. Carregar um array de bytes de um arquivo

```
byte[] result = null;
File file = new File("documento.odp");
FileInputStream is = new FileInputStream(file);
result = new byte[(int) file.length()];
is.read(result);
is.close();
return result;
```

6.2. Gravar um array de bytes em um arquivo

```
byte[] conteudo = "este eh um conteudo de arquivo texto".getBytes();
FileOutputStream out = new FileOutputStream(new File("arquivo.txt");
out.write(sign);
out.close();
```

6.3. Carregar uma chave privada em arquivo

```
File file = new File("private_rsa_1024.pkcs8");
fileContent = new byte[(int) file.length()];
is = new FileInputStream(file);
is.read(fileContent);
is.close();
PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(fileContent);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PrivateKey chavePrivada = keyFactory.generatePrivate(privateKeySpec);
```

6.4. Carregar uma chave privada de um token

```
KeyStoreLoader keyStoreLoader = KeyStoreLoaderFactory.factoryKeyStoreLoader();
KeyStore keyStore = keyStoreLoader.getKeyStore("pinnumber");
String certificateAlias = keyStore.aliases().nextElement();
PrivateKey chavePrivada = (PrivateKey)keyStore.getKey(certificateAlias, "pinnumber");
```

6.5. Carregar uma chave pública em arquivo

```
File file = new File("public_rsa_1024.pkcs8");
byte[] fileContent = new byte[(int) file.length()];
is = new FileInputStream(file);
is.read(fileContent);
is.close();
X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(fileContent);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey chavePublica = keyFactory.generatePublic(publicKeySpec);
```

6.6. Carregar uma chave pública de um token

```
KeyStoreLoader keyStoreLoader = KeyStoreLoaderFactory.factoryKeyStoreLoader();
KeyStore keyStore = keyStoreLoader.getKeyStore();
CertificateLoader certificateLoader = new CertificateLoader();
certificateLoader.setKeyStore(keyStore);
X509Certificate certificate = certificateLoader.loadFromToken();
PublicKey chavePublica = certificate.getPublicKey();
```

6.7. Carregar um certificado digital de um arquivo

```
CertificateLoader certificateLoader = new CertificateLoaderImpl();
X509Certificate certificate = certificateLoader.load(new File("certificado.cer");
```

6.8. Carregar um certificado digital de um token

```
KeyStoreLoader keyStoreLoader = KeyStoreLoaderFactory.factoryKeyStoreLoader();
KeyStore keyStore = keyStoreLoader.getKeyStore();
CertificateLoader certificateLoader = new CertificateLoaderImpl();
certificateLoader.setKeyStore(keyStore);
X509Certificate certificate = certificateLoader.loadFromToken();
```

Parte III. Policy Engine

O *policy-engine* é um componente acessório que visa atender as politicas de assinaturas publicadas pela ICP-BRASIL, ele prov# uma forma de fabricar as políticas de forma automatizada, através da leitura do arquivo da política

Além disso o componente utiliza algoritmos de hash para criptografia de dados com a finalidade de criar um valor único que identifique um dado original. Este recurso é recomendado para finalidades de autenticação, nas quais deseja-se armazenar as senhas criptografas por meio de um valor hash. Também é possível construir hash de arquivos no intuito de avaliar sua integridade física.

O componente também realiza as funções de cifragem e decifragem por meio de algoritmos de chave-assimétrica. Neste processo é necessário um par de chaves para realizar a cifragem e decifram das mensagens. A primeira chave é denominada chave privada, ela é de posse exclusiva de seu detentor e ninguém mais a conhece. A segunda chave do par e denominada de chave pública e pode ser enviada a qualquer indivíduo.

Configuração do Policy Engine

7.1. Instalação do componente

Para utilizar o componente *policy-engine* num projeto Java, basta adicionar a sua dependência no arquivo `pom.xml` , conforme o seu gerenciador de projetos:

- *Apache-Maven* [https://maven.apache.org/]

```
      <dependency>
    <groupId>org.demoiselle.signer</groupId>
    <artifactId>policy-engine</artifactId>
    <version>3.2.6</version>
   </dependency>
```

- *Apache Buildr* [https://buildr.apache.org/]

```
'org.demoiselle.signer:policy-engine:jar:3.2.6'
```

- *Apache Ivy* [http://ant.apache.org/ivy/]

```
<dependency org="org.demoiselle.signer" name="policy-engine" rev="3.2.6" />
```

- *Groovy Grape* [http://docs.groovy-lang.org/latest/html/documentation/grape.html]

```
@Grapes(@Grab(group='org.demoiselle.signer', module='policy-engine', version='3.2.6'))
```

- *Gradle/Grails* [https://github.com/grails/grails-gradle-plugin]

```
<dependency org="org.demoiselle.signer" name="policy-engine" rev="3.2.6" />
```

- *Scala SBT* [http://www.scala-sbt.org/]

```
libraryDependencies += "org.demoiselle.signer" % "policy-engine" % "3.2.6"
```

- *Leiningen* [https://leiningen.org/]

```
<[org.demoiselle.signer/policy-engine "3.2.6"]
```

Caso não esteja utilizando nenhum outro tipo de gerenciador (estava morando numa caverna nos últimos dez anos), pode baixar o .jar do repositório:

http://repo1.maven.org/maven2/org/demoiselle/signer/policy-engine/

Funcionalidades

8.1. Fabricar políticas

Além de algumas funcionalidade intrinsecas ao uso na validação e geração das assinaturas, a principal funcionalidade do Policy-Engine é fabricar as políticas de assinatura. Atualmente há um série de *políticas* [http://iti.gov.br/repositorio/84-repositorio/133-artefatos-de-assinatura-digital] em vigência na ICP-BRASIL e o componente está preparado para cria a maioria delas, a exceção até a versão 3.1.x do Signer são as políticas para o padrão XAdES (XML).

Atualmennte temos as seguintes:

- *AD_RB_CADES_2_2* [http://politicas.icpbrasil.gov.br/PA_AD_RB_v2_2.der]
- *AD_RT_CADES_2_2* [http://politicas.icpbrasil.gov.br/PA_AD_RT_v2_2.der]
- *AD_RV_CADES_2_2* [http://politicas.icpbrasil.gov.br/PA_AD_RV_v2_2.der]
- *AD_RC_CADES_2_2* [http://politicas.icpbrasil.gov.br/PA_AD_RC_v2_2.der]
- *AD_RA_CADES_2_3* [http://politicas.icpbrasil.gov.br/PA_AD_RA_v2_3.der]
- *AD_RB_PADES_1_0* [http://politicas.icpbrasil.gov.br/PA_PAdES_AD_RB_v1_0.der]
- *AD_RT_PADES_1_0* [http://politicas.icpbrasil.gov.br/PA_PAdES_AD_RT_v1_0.der]
- *AD_RC_PADES_2_3* [http://politicas.icpbrasil.gov.br/PA_PAdES_AD_RC_v1_1.der]
- *AD_RA_PADES_1_1* [http://politicas.icpbrasil.gov.br/PA_PAdES_AD_RA_v1_1.der]

Das políticas acimas o componente atualmente está preparado para gerar assinaturas nos padrões:

- *AD_RB_CADES_2_2* [http://politicas.icpbrasil.gov.br/PA_AD_RB_v2_2.der]
- *AD_RT_CADES_2_2* [http://politicas.icpbrasil.gov.br/PA_AD_RT_v2_2.der]
- *AD_RB_PADES_1_0* [http://politicas.icpbrasil.gov.br/PA_PAdES_AD_RB_v1_0.der]
- *AD_RT_PADES_1_0* [http://politicas.icpbrasil.gov.br/PA_PAdES_AD_RT_v1_0.der]

O componente disponível que faz uso é o policy-impl-cades, conforme o exemplo abaixo:

```
// Para usar a politica de Refer#ncia B#sica:
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_2);

// Para usar a pol#tica com Refer#ncia de Tempo
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RT_CADES_2_2);
```

Parte IV. Cadeias de Autoridades da IPC-BRASIL

O Chain-ICP-Brasil fornece uma implementação para validação do conjunto de Autoridades Certificadoras da cadeia ICP-Brasil. O acionamento das funcionalidades é feito automaticamente pelo componentes de geração e validação de assinaturas como o *policy-impl-cades* O mais importante é mantê-lo sempre atualizado.

Configuração do Chain-ICP-Brasil

9.1. Instalação do componente

Para instalar o componente *Demoiselle CA ICP-Brasil* na aplicação, basta adicionar a sua dependência de acordo com o gerenciador de projetos:

- *Apache-Maven* [https://maven.apache.org/]

```
        <dependency>
     <groupId>org.demoiselle.signer</groupId>
     <artifactId>chain-icp-brasil</artifactId>
     <version>3.2.6</version>
    </dependency>
```

- *Apache Buildr* [https://buildr.apache.org/]

```
'org.demoiselle.signer:chain-icp-brasil:jar:3.2.6'
```

- *Apache Ivy* [http://ant.apache.org/ivy/]

```
<dependency org="org.demoiselle.signer" name="chain-icp-brasil" rev="3.2.6" />
```

- *Groovy Grape* [http://docs.groovy-lang.org/latest/html/documentation/grape.html]

```
@Grapes(@Grab(group='org.demoiselle.signer', module='chain-icp-brasil', version='3.2.6'))
```

- *Gradle/Grails* [https://github.com/grails/grails-gradle-plugin]

```
<dependency org="org.demoiselle.signer" name="chain-icp-brasil" rev="3.2.6" />
```

- *Scala SBT* [http://www.scala-sbt.org/]

```
libraryDependencies += "org.demoiselle.signer" % "chain-icp-brasil" % "3.2.6"
```

- *Leiningen* [https://leiningen.org/]

```
<[org.demoiselle.signer/chain-icp-brasil "3.2.6"]
```

Caso não esteja utilizando nenhum outro tipo de gerenciador (estava morando numa caverna nos últimos dez anos), pode baixar o .jar do repositório:

http://repo1.maven.org/maven2/org/demoiselle/signer/chain-icp-brasil/

9.2. Autoridades Certificadoras

Abaixo mostraremos a lista de autoridades certificadoras atual que está presente no componente:

Tabela 9.1. Lista de Autoridades Certificadoras

Autoridade Certificadora	Arquivo	Validade
AC CAIXA-JUS v2	ac_caixa_jus_v2.crt	Wed Dec 28 11:03:21 BRST 2011 a: Sat Dec 28 11:03:21 BRST 2019
AC VALID RFB v5	ac_valid_rfb_v5.crt	Fri May 05 15:06:38 BRT 2017 a: Tue Feb 20 15:06:38 BRT2029
AC Instituto Fenacon G2	ac_instituto_fenacon_g2.crt	Sun Nov 06 22:00:00 BRST 2011 a: Wed Nov 06 21:59:59BRST 2019
AC Imprensa Oficial SP RFB G4	ac_imprensa_oficial_sp_rfb_g4.crt	Mon Feb 09 15:09:45 BRST 2015 a: Mon Oct 11 14:09:45 BRT 2021
Autoridade Certificadora SERPRORFB v3	ac_serpro_rfb_v3.crt	Tue Dec 13 14:58:21 BRST 2011 a: Fri Dec 13 14:58:21 BRST 2019
AC Certisign-JUS G3	ac_certisign_jus_g3.crt	Fri Dec 23 11:03:23 BRST 2011 a: Mon Dec 23 11:03:23BRST 2019
Autoridade Certificadora SERPRO v3	autoridade_certificadora_serpro_v3.crt	Fri Oct 21 10:02:47 BRST 2011 a: Thu Oct 21 10:02:47 BRST 2021
Autoridade Certificadora da Justica v5	autoridade_certificadora_da_justica_v5.crt	Wed Oct 19 10:18:21 BRST 2016 a: Fri Mar 02 09:00:21 BRT 2029
AC PRODEST RFB v2	ac_prodest_rfb_v2.crt	Tue Dec 20 10:16:22 BRST 2011 a: Fri Dec 20 10:16:22 BRST 2019
AC Certisign Tempo G2	ac_certisign_tempo_g2.crt	Fri Oct 21 00:00:00 BRST 2016 a: Thu Mar 01 00:00:00 BRT 2029
AC VALID RFB	ac_valid_rfb.crt	Mon Jul 23 15:14:06 BRT 2012 a: Thu Jul 23 15:14:06 BRT 2020
AC PRODEMGE G3	ac_prodemge_g3.crt	Wed Nov 23 22:00:00 BRST 2011 a: Sat Nov 23 21:59:59 BRST 2019
Autoridade Certificadora do SERPRO Final SSL	autoridade_certificadora_do_serpro_final_ssl.crt	Mon Jun 09 17:36:52 BRST 2017 a: Thu Feb 15 17:36:52 BRST 2029
AC PRODEMGE RFB G3	ac_prodemge_rfb_g3.crt	Tue Dec 27 11:40:12 BRST 2011 a: Fri Dec 27 11:40:12 BRST 2019
AC BOA VISTA RFB	ac_boa_vista_rfb.crt	Fri Nov 29 15:26:39 BRST 2013 a: Mon Oct 11 14:26:39 BRT 2021

Autoridade Certificadora	Arquivo	Validade
AC CAIXA SPB	ac_caixa_spb.crt	Mon Jan 19 17:57:33 BRST 2015 a: Thu Dec 02 10:16:53 BRST 2021
Autoridade Certificadora da Casa da Moeda do Brasil	autoridade_certificadora_da_casa_da_moeda_do_brasil.crt	Wed Feb 03 16:46:18 BRST 2010 a: Mon Feb 03 16:46:18 BRST 2020
SERASA Autoridade Certificadora v2	serasa_autoridade_certificadora_v2.crt	Wed Oct 05 15:15:06 BRT 2011 a: Sat Oct 05 15:15:06 BRT 2019
Autoridade Certificadora da Presidencia da Republica v4	autoridade_certificadora_da_presidencia_da_republica_v4.crt	Fri Jun 21 10:07:23 BRT 2013 a: Wed Jun 21 09:00:23 BRT 2023
AC SINCOR G3	ac_sincor_g3.crt	Sun Nov 06 22:00:00 BRST 2011 a: Wed Nov 06 21:59:59 BRST 2019
AC FENACON Certisign RFB G3	ac_fenacon_certisign_rfb_g3.crt	Thu Dec 22 11:29:54 BRST 2011 a: Sun Dec 22 11:29:54 BRST 2019
AC DIGITALSIGN ACP	ac_digitalsign_acp.crt	Wed Sep 18 10:36:38 BRT 2013 a: Wed Jun 21 09:00:38 BRT 2023
Autoridade Certificadora do SERPRORFB SSL	autoridade_certificadora_do_serpro_rfb_ssl.crt	Mon Jun 19 16:05:08 BRT 2017 a: Sun Oct 10 16:05:08 BRT 2021
AC Instituto Fenacon RFB G3	ac_instituto_fenacon_rfb_g3.crt	Mon Dec 19 15:29:42 BRST 2016 a: Tue Feb 20 14:29:42 BRT 2029
AC SINCOR RFB G3	ac_sincor_rfb_g3.crt	Mon Dec 26 17:11:16 BRST 2011 a: Thu Dec 26 17:11:16 BRST 2019
AC CAIXA PF v2	ac_caixa_pf_v2.crt	Fri Dec 23 11:52:58 BRST 2011 a: Sat Dec 21 11:52:58 BRST 2019
AC CERTISIGN-JUS SSL G5	ac_certisign-jus_ssl_g5.crt	Thu Nov 24 15:42:23 BRST 2016 a: Tue Feb 20 14:42:23 BRT 2029
AC BR RFB G3	ac_br_rfb_g3.crt	Mon Dec 26 16:08:04 BRST 2011 a: Thu Dec 26 16:08:04 BRST 2019
AC CERTISIGN-JUS G6	ac_certisign_jus_g6.crt	Tue Feb 14 16:43:22 BRST 2017 a: Tue Feb 20 15:43:22 BRT 2029
AC SOLUTI Multipla	ac_soluti_multipla_v1.crt	Wed Dec 05 06:50:39 BRST 2012 a: Tue Jun 20 20:58:59 BRT 2023
AC Imprensa Oficial G4	ac_imprensa_oficial_g4.crt	Fri Jan 16 15:30:48 BRST 2015 a: Mon Jan 16 15:30:48 BRST 2023
SERASA Autoridade Certificadora Principal v2	serasa_autoridade_certificadora_principal_v2.crt	Tue Sep 20 16:08:05 BRT 2011 a: Mon Sep 20 16:08:05 BRT 2021
AC DOCCLOUD RFB	ac_doccloud_rfb.crt	Wed Jan 20 15:22:15 BRST 2016 a: Mon Oct 11 14:22:15 BRT 2021
AC SOLUTI RFB	ac_soluti_rfb.crt	Wed Jun 03 10:34:11 BRT 2015 a: Mon Oct 11 10:34:11 BRT 2021
AC SOLUTI Multipla CODESIGNING	ac_soluti_multipla_codesigning.crt	Tue Dec 13 22:04:22 BRST 2016 a: Tue Jun 20 20:58:59 BRT 2023
AC Secretaria da Receita Federal do Brasil v3	ac_secretaria_da_receita_federal_do_brasil_v3.crt	Fri Oct 21 10:16:29 BRST 2011 a: Thu Oct 21 10:16:29 BRST 2021

Autoridade Certificadora	Arquivo	Validade
AC Imprensa Oficial SP G2	ac_imprensa_oficial_sp_g2.crt	Mon Nov 09 10:24:06 BRST 2009 a: Sat Nov 09 10:24:06 BRST 2019
AC Certisign G7	ac_certisign_g7.crt	Tue Jun 28 10:07:38 BRT 2016 a: Fri Mar 02 09:00:38 BRT 2029
AC BOA VISTA	ac_boavista.crt	Mon Nov 04 16:38:33 BRST 2013 a: Wed Jun 21 09:00:33 BRT 2023
AC Certisign RFB G5	ac_certisign_rfb_g5.crt	Thu Dec 08 15:44:03 BRST 2016 a: Tue Feb 20 14:44:03 BRT 2029
AC VALID PLUS TIMESTAMPING	ac_valid_plus_timestamping.crt	Thu Apr 20 13:48:20 BRT 2017 a: Thu Mar 01 13:48:20 BRT 2029
AC CAIXA-JUS v1	ac_caixa_jus_v1.crt	Fri Jan 14 10:59:34 BRST 2011 a: Mon Jan 14 10:59:34 BRST 2019
Autoridade Certificadora da Casa da Moeda do Brasil v3	autoridade_certificadora_da_casa_da_moeda_do_brasil_v3.crt	Tue Oct 16 16:41:34 BRT 2012 a: Sun Oct 16 16:41:34 BRST 2022
AC Imprensa Oficial SP RFB G3	ac_imprensa_oficial_sp_rfb_g3.crt	Tue Dec 27 15:39:24 BRST 2011 a: Fri Dec 27 15:39:24 BRST 2019
AC Notarial RFB G4	ac_notarial_rfb_g4.crt	Tue Dec 20 15:21:30 BRST 2016 a: Tue Feb 20 14:21:30 BRT 2029
AC OAB G3	ac_oab_g3.crt	Sat Nov 19 00:00:00 BRST 2016 a: Thu Mar 01 00:00:00 BRT 2029
Autoridade Certificadora Raiz Brasileira v1	icp-brasil.crt	Tue Jul 29 16:17:10 BRT 2008 a: Thu Jul 29 16:17:10 BRT 2021
AC VALID SPB v5	ac_valid_spb_v5.crt	Thu May 04 18:56:22 BRT 2017 a: Thu Mar 01 18:56:22 BRT 2029
AC CAIXA PJ v2	ac_caixa_pj_v2.crt	Fri Dec 23 11:55:36 BRST 2011 a: Sat Dec 21 11:55:36 BRST 2019
Autoridade Certificadora SERPRO v2	autoridade_certificadora_serpro_v2.crt	Wed Feb 18 16:03:28 BRT 2009 a: Mon Feb 18 16:03:28 BRT 2019
Autoridade Certificadora do SERPRO Final v5	autoridade_certificadora_do_serpro_final_v5.crt	Mon Feb 06 17:07:54 BRST 2017 a: Thu Feb 15 17:07:54 BRST 2029
Autoridade Certificadora da Justica v4	autoridade_certificadora_da_justica_v4.crt	Tue Nov 22 15:15:01 BRST 2011 a: Mon Nov 22 15:15:01 BRST 2021
AC Certisign Tempo G1	ac_certisign_tempo_g1.crt	Thu Jul 24 21:00:00 BRT 2014 a: Sun Jul 25 20:59:59 BRT 2021
AC Certisign Multipla G7	ac_certisign_multipla_g7.crt	Thu Aug 25 00:00:00 BRT 2016 a: Thu Mar 01 00:00:00 BRT 2029
AC VALID PLUS CODESIGNING	ac_valid_plus_codesigning.crt	Thu Apr 20 13:56:44 BRT 2017 a: Thu Mar 01 13:56:44 BRT 2029
Autoridade Certificadora Raiz Brasileira v2	icp-brasilv2.crt	Mon Jun 21 16:04:57 BRT 2010 a: Wed Jun 21 16:04:57 BRT 2023
AC Secretaria da Receita Federal do Brasil	secretaria_da_receita_federal_do_brasil.crt	Fri Oct 24 10:39:46 BRST 2008 a: Wed Oct 24 10:39:46 BRST 2018

Autoridade Certificadora	Arquivo	Validade
AC CACB RFB	ac_cacb_rfb.crt	Wed Jun 03 11:38:43 BRT 2015 a: Mon Oct 11 11:38:43 BRT 2021
SERASA Autoridade Certificadora Principal v5	serasa_autoridade_certificadora_principal_v5.crt	Tue Jun 07 15:25:00 BRT 2016 a: Fri Mar 02 15:25:00 BRT 2029
AC ONLINE RFB	ac_online_rfb.crt	Fri Sep 25 15:11:38 BRT 2015 a: Mon Oct 11 15:11:38 BRT 2021
AC Certisign G3	ac_certisign_g3.crt	Tue Sep 02 16:56:38 BRT 2008 a: Sun Sep 02 16:56:38 BRT 2018
AC BOA VISTA CERTIFICADORA	ac_boavista_certificadora.crt	Fri Nov 29 20:12:24 BRST 2013 a: Mon Nov 29 20:12:24 BRST 2021
AC Certisign SPB G6	ac_certisign_spb_g6.crt	Fri Jan 27 00:00:00 BRST 2017 a: Thu Mar 01 00:00:00 BRT 2029
AC SOLUTI-JUS CODESIGNING v5	ac_soluti-jus_codesigning_v5.crt	Thu Apr 06 15:51:10 BRT 2017 a: Tue Feb 20 15:51:10 BRT 2029
Autoridade Certificadora da Presidencia da Republica v3	autoridade_certificadora_da_presidencia_da_republica_v3.crt	Tue Nov 22 15:26:47 BRST 2011 a: Mon Nov 22 15:26:47 BRST 2021
AC SOLUTI Multipla TIMESTAMPING	ac_soluti_multipla_timestamping.crt	Tue Dec 13 22:24:40 BRST 2016 a: Tue Jun 20 20:58:59 BRT 2023
AC Certisign Multipla SSL	ac_certisign_multipla_ssl.crt	Tue Dec 13 00:00:00 BRST 2016 a: Thu Mar 01 00:00:00 BRT 2029
AC FENACOR RFB	ac_fenacor_rfb.crt	Wed Feb 22 15:10:15 BRT 2017 a: Tue Feb 20 15:10:15 BRT 2029
AC Instituto Fenacon RFB G2	ac_instituto_fenacon_rfb_g2.crt	Mon Dec 26 10:38:30 BRST 2011 a: Thu Dec 26 10:38:30 BRST 2019
AC SOLUTI-JUS v5	ac_soluti-jus_v5.crt	Thu Apr 06 16:12:05 BRT 2017 a: Tue Feb 20 16:12:05 BRT 2029
AC SOLUTI-JUS v1	ac_soluti_jus_v1.crt	Thu Nov 20 11:28:35 BRST 2014 a: Fri Oct 22 11:28:35 BRST 2021
AC PETROBRAS G4	ac_petrobras_g4.crt	Fri Mar 10 00:00:00 BRT 2017 a: Thu Mar 10 00:00:00 BRT 2022
AC Imprensa Oficial G3	ac_imprensa_oficial_g3.crt	Tue Dec 27 22:00:00 BRST 2011 a: Fri Dec 27 21:59:59 BRST 2019
AC VALID-JUS v4	ac_valid_jus_v4.crt	Fri Nov 28 15:40:23 BRST 2014 a: Fri Oct 22 15:40:23 BRST 2021
AC Imprensa Oficial SSL	ac_imprensa_oficial_ssl.crt	Tue Apr 04 11:23:36 BRT 2017 a: Wed Jun 21 10:01:08 BRT 2023
SERASA CD SSL V5	serasa_cd_ssl_v5.crt	Thu Dec 01 10:35:22 BRST 2016 a: Fri Mar 02 15:25:00 BRT 2029
AC DIGITALSIGN RFB	ac_digitalsign_rfb.crt	Tue Nov 05 14:49:10 BRST 2013 a: Mon Oct 11 13:49:10 BRT 2021
SERASA Autoridade Certificadora Principal v1	serasa_autoridade_certificadora_principal_v1.crt	Thu Jul 31 15:44:55 BRT 2008 a: Tue Jul 31 15:44:55 BRT 2018

Autoridade Certificadora	Arquivo	Validade
Autoridade Certificadora Raiz Brasileira v5	icp-brasilv5.crt	Wed Mar 02 10:01:38 BRT 2016 a: Fri Mar 02 20:59:38 BRT 2029
AC SOLUTI Multipla SSL	ac_soluti_multipla_ssl.crt	Tue Dec 13 22:05:50 BRST 2016 a: Tue Jun 20 20:58:59 BRT 2023
AC Certisign G6	ac_certisign_g6.crt	Tue Sep 20 15:51:40 BRT 2011 a: Mon Sep 20 15:51:40 BRT 2021
AC SINCOR RIO RFB G2	ac_sincor_rio_rfb_g2.crt	Wed Feb 22 15:23:01 BRT 2017 a: Tue Feb 20 15:23:01 BRT 2029
AC Certisign RFB G4	ac_certisign_rfb_g4.crt	Thu Dec 22 10:56:09 BRST 2011 a: Sun Dec 22 10:56:09 BRST 2019
Autoridade Certificadora da Casa da Moeda do Brasil v2	autoridade_certificadora_da_casa_da_moeda_do_brasil_v2.crt	Wed Dec 28 11:24:47 BRST 2011 a: Tue Dec 28 11:24:47 BRST 2021
AC Notarial RFB G3	ac_notarial_rfb_g3.crt	Mon Dec 26 16:37:53 BRST 2011 a: Thu Dec 26 16:37:53 BRST 2019
AC VALID PLUS v5	ac_valid_plus_v5.crt	Thu May 04 18:46:00 BRT 2017 a: Thu Mar 01 18:46:00 BRT 2029
AC OAB G2	ac_oab_g2.crt	Tue Nov 15 22:00:00 BRST 2011 a: Fri Nov 15 21:59:59 BRST 2019
AC SAFEWEB RFB	ac_safeweb_rfb.crt	Thu Feb 12 15:21:01 BRST 2015 a: Mon Oct 11 14:21:01 BRT 2021
AC LINK RFB	ac_link_rfb.crt	Wed Jan 20 15:40:33 BRST 2016 a: Mon Oct 11 14:40:33 BRT 2021
AC CERTISIGN-JUS CODESIGNING G6	ac_certisign-jus_codesigning_g6.crt	Thu Apr 06 16:23:53 BRT 2017 a: Tue Feb 20 16:23:53 BRT 2029
Autoridade Certificadora do SERPRO Final v4	autoridade_certificadora_do_serpro_final_v4.crt	Wed Jan 15 17:34:31 BRST 2014 a: Fri Oct 08 16:34:31 BRT 2021
Autoridade Certificadora da Justica v3	autoridade_certificadora_da_justica_v3.crt	Fri Jun 12 16:20:33 BRT 2009 a: Wed Jun 12 16:20:33 BRT 2019
AC SINCOR RFB G5	ac_sincor_rfb_g5.crt	Tue Dec 20 15:36:17 BRST 2016 a: Tue Feb 20 14:36:17 BRT 2029
AC Certisign Multipla G6	ac_certisign_multipla_g6.crt	Tue Feb 16 22:00:00 BRST 2016 a: Sun Sep 19 20:59:59 BRT 2021
AC VALID PLUS SSL	ac_valid_plus_ssl.crt	Thu Apr 20 13:52:30 BRT 2017 a: Thu Mar 01 13:52:30 BRT 2029
AC SERASA RFB v2	ac_serasa_rfb_v2.crt	Fri Dec 16 11:39:42 BRST 2011 a: Mon Dec 16 11:39:42 BRST 2019
AC VALID BRASIL CODESIGNING	ac_valid_brasil_codesigning.crt	Thu Apr 20 13:24:33 BRT 2017 a: Thu Mar 01 13:24:33 BRT 2029
AC SOLUTI-JUS SSL v5	ac_soluti-jus_ssl_v5.crt	Thu Apr 06 16:04:05 BRT 2017 a: Tue Feb 20 16:04:05 BRT 2029
AC SERPRO-JUS v5	ac_serpro_jus_v5.crt	Thu Mar 16 10:43:37 BRT 2017 a: Tue Feb 20 10:43:37 BRT 2029

Autoridade Certificadora	Arquivo	Validade
AC DIGITALSIGN	ac_digitalsign.crt	Mon Oct 21 13:23:02 BRST 2013 a: Sun Oct 17 13:23:02 BRST 2021
AC Imprensa Oficial SP G4	ac_imprensa_oficial_sp_g4.crt	Fri Dec 19 11:01:08 BRST 2014 a: Wed Jun 21 10:01:08 BRT 2023
AC EGBA Multipla	ac_egba_multipla.crt	Tue Feb 16 22:00:00 BRST 2016 a: Sun Sep 19 20:59:59 BRT 2021
AC Certisign SPB G5	ac_certisign_spb_g5.crt	Thu Sep 22 21:00:00 BRT 2011 a: Sun Sep 22 20:59:59 BRT 2019
AC VALID SPB	ac_valid_spb.crt	Thu Apr 18 22:28:10 BRT 2013 a: Sun Apr 18 22:28:10 BRT 2021
AC CERTISIGN-JUS G5	ac_certisign-jus_g5.crt	Thu Nov 24 15:31:24 BRST 2016 a: Tue Feb 20 14:31:24 BRT 2029
AC VALID	ac_valid.crt	Mon Jun 04 09:43:46 BRT 2012 a: Sat Jun 04 09:43:46 BRT 2022
AC VALID BRASIL	ac_valid_brasil.crt	Thu Jul 12 12:34:31 BRT 2012 a: Sun Jul 12 12:34:31 BRT 2020
AC CNDL RFB	ac_cndl_rfb.crt	Wed Jan 27 15:08:20 BRST 2016 a: Mon Oct 11 14:08:20 BRT 2021
Autoridade Certificadora da Presidencia da Republica v2	autoridade_certificadora_da_presidencia_da_republica_v2.crt	Mon Oct 13 15:48:17 BRT 2008 a: Sat Oct 13 15:48:17 BRT 2018
AC Instituto Fenacon G3	ac_instituto_fenacon_g3.crt	Wed Dec 14 00:00:00 BRST 2016 a: Thu Mar 01 00:00:00 BRT 2029
Autoridade Certificadora SERPRORFBv4	ac_serpro_rfb_v4.crt	Mon Aug 04 15:38:36 BRT 2014 a: Mon Oct 11 15:38:36 BRT 2021
AC Certisign Multipla CodeSigning	ac_certisign_multipla_codesigning.crt	Tue Dec 13 00:00:00 BRST 2016 a: Thu Mar 01 00:00:00 BRT 2029
Autoridade Certificadora SERPRORFBv5	autoridade_certificadora_serprorfbv5.crt	Tue Mar 28 16:07:19 BRT 2017 a: Tue Feb 20 16:07:19 BRT 2029
AC PETROBRAS G3	ac_petrobras_g3.crt	Sun Nov 27 22:00:00 BRST 2011 a: Wed Nov 27 21:59:59 BRST 2019
Autoridade Certificadora SERPRO v4	autoridade_certificadora_serpro_v4.crt	Wed Sep 14 10:10:42 BRT 2016 a: Fri Mar 02 09:00:42 BRT 2029
AC CNDL RFB v2	ac_cndl_rfb_v2.crt	Mon Feb 09 16:36:22 BRST 2015 a: Mon Oct 11 15:36:22 BRT 2021
AC Imprensa Oficial G2	ac_imprensa_oficial_g2.crt	Sun May 15 21:00:00 BRT 2011 a: Wed May 15 20:59:59 BRT 2019
AC ONLINE BRASIL	ac_online_brasil.crt	Fri Nov 28 10:01:00 BRST 2014 a: Tue May 31 09:01:00 BRT 2022
AC SERASA RFB v5	ac_serasa_rfb_v5.crt	Wed Oct 19 16:11:56 BRST 2016 a: Fri Feb 02 16:11:56 BRST 2029
SERASA Certificadora Digital v2	serasa_certificadora_digital_v2.crt	Thu Oct 13 12:31:15 BRT 2011 a: Sun Oct 13 12:31:15 BRT 2019

Autoridade Certificadora	Arquivo	Validade
AC PRODEMGE G4	ac_prodemge_g4.crt	Thu Jan 12 00:00:00 BRST 2017 a: Thu Mar 01 00:00:00 BRT 2029
AC CAIXA v2	ac_caixa_v2.crt	Fri Dec 02 10:16:53 BRST 2011 a: Thu Dec 02 10:16:53 BRST 2021
AC VALID v5	ac_valid_v5.crt	Fri Sep 30 09:58:13 BRT 2016 a: Fri Mar 02 09:00:13 BRT 2029
AC PRODEMGE RFB G4	ac_prodemge_rfb_g4.crt	Mon Dec 19 15:19:03 BRST 2016 a: Tue Feb 20 14:19:03 BRT 2029
AC SINCOR RIO RFB G1	ac_sincor_rio_rfb_g1.crt	Thu May 08 15:19:59 BRT 2014 a: Mon Oct 11 15:19:59 BRT 2021
AC SERASA-JUS v2	ac_serasa_jus_v2.crt	Wed Dec 28 11:39:38 BRST 2011 a: Sat Dec 28 11:39:38 BRST 2019
AC DIGITAL	ac_digital.crt	Thu Feb 26 15:11:52 BRT 2015 a: Tue Jun 20 20:58:59 BRT 2023
AC VALID BRASIL SSL	ac_valid_brasil_ssl.crt	Thu Apr 20 13:30:14 BRT 2017 a: Thu Mar 01 13:30:14 BRT 2029
AC VALID PLUS	ac_valid_plus.crt	Fri Nov 28 10:45:43 BRST 2014 a: Tue May 31 09:45:43 BRT 2022
AC CONSULTI BRASIL RFB	ac_consulti_brasil_rfb.crt	Wed May 31 15:52:04 BRT 2017 a: Tue Feb 20 15:52:04 BRT 2029
AC SINCOR G4	ac_sincor_g4.crt	Thu Jan 12 00:00:00 BRST 2017 a: Thu Mar 01 00:00:00 BRT 2029
AC CERTISIGN-JUS CODESIGNING G5	ac_certisign-jus_codesigning_g5.crt	Thu Nov 24 15:47:59 BRST 2016 a: Tue Feb 20 14:47:59 BRT 2029
Autoridade Certificadora do SERPRO Final v3	autoridade_certificadora_do_serpro_final_v3.crt	Wed Nov 16 11:37:27 BRST 2011 a: Sat Nov 16 11:37:27 BRST 2019
AC SINCOR RFB G4	ac_sincor_rfb_g4.crt	Thu Jan 26 10:10:17 BRST 2012 a: Sun Jan 26 10:10:17 BRST 2020
AC CERTISIGN-JUS SSL G6	ac_certisign-jus_ssl_g6.crt	Thu Apr 06 16:18:05 BRT 2017 a: Tue Feb 20 16:18:05 BRT 2029
AC BR RFB G4	ac_br_rfb_g4.crt	Tue Dec 20 15:29:07 BRST 2016 a: Tue Feb 20 14:29:07 BRT 2029
Autoridade Certificadora do PRODERJ v2	autoridade_certificadora_do_proderj_v2.crt	Thu Dec 22 12:18:56 BRST 2011 a: Sun Dec 22 12:18:56 BRST 2019
AC Certisign Multipla G5	ac_certisign_multipla_g5.crt	Thu Sep 22 21:00:00 BRT 2011 a: Sun Sep 22 20:59:59 BRT 2019
AC SOLUTI	ac_soluti.crt	Mon Dec 03 10:39:13 BRST 2012 a: Tue Jun 20 20:59:59 BRT 2023
AC SERPRO-JUS v4	ac_serpro_jus_v4.crt	Fri Dec 23 12:08:40 BRST 2011 a: Mon Dec 23 12:08:40 BRST 2019
AC Secretaria da Receita Federal do Brasil v4	ac_secretaria_da_receita_federal_do_brasil_v4.crt	Wed Jun 20 10:32:04 BRT 2016 a: Fri Mar 02 09:00:04 BRT 2029

Autoridade Certificadora	Arquivo	Validade
SERASA Certificadora Digital v5	serasa_cd_v5.crt	Thu Sep 22 11:39:57 BRT 2016 a: Fri Mar 02 15:25:00 BRT 2029
AC Imprensa Oficial SP G3	ac_imprensa_oficial_sp_g3.crt	Wed Dec 21 10:54:11 BRST 2011 a: Tue Dec 21 10:54:11 BRST 2021
AC DIGITALSIGN SSL	ac_digitalsign_ssl.crt	Mon Mar 06 12:31:35 BRT 2017 a: Wed Jun 21 09:00:38 BRT 2023
AC CACB RFB G2	ac_cacb_rfb_g2.crt	Wed Apr 05 15:43:23 BRT 2017 a: Tue Feb 20 15:43:23 BRT 2029
AC VALID BRASIL v5	ac_valid_brasil_v5.crt	Thu May 04 18:28:32 BRT 2017 a: Thu Mar 01 18:28:32 BRT 2029
AC SERASA-JUS v5	ac_serasa_jus_v5.crt	Wed Nov 23 15:59:39 BRST 2016 a: Tue Feb 20 14:59:39 BRT 2029
SERASA Autoridade Certificadora v5	serasa_ac_v5.crt	Thu Sep 22 11:26:36 BRT 2016 a: Fri Mar 02 15:25:00 BRT 2029

Parte V. TimeStamp - Carimbo de tempo

O *signer-timestamp* é o componente que provê as funcionalidades para obtenção de carimbos de tempo de uma *Autoridade de Carimbo de Tempo* [https://en.wikipedia.org/wiki/Trusted_timestamping] credenciada à ICP-BRASIL .

- *Lista das autoridades credenciadas* [http://www.iti.gov.br/icp-brasil/57-icp-brasil/134-autoridades-certificadoras-do-tempo].

A autoridade padrão do componente é a *Autoridade de Carimbo do SERPRO.* [https://act.serpro.gov.br/]

Importante

Para emissão de um carimbo de tempo é preciso que o assinante possua um certificado ICP-BRASIL e este certificado esteja cadastrado na Autoridade de Carimbo de tempo. O custo de emissão depende de cada Autoridade

Configuração do TimeStamp

10.1. Instalação do componente

Para instalar o componente *Demoiselle Signer TimeStamp* na aplicação, basta adicionar a sua dependência de acordo com o gerenciador de projetos:

- *Apache-Maven* [https://maven.apache.org/]

```
      <dependency>
    <groupId>org.demoiselle.signer</groupId>
    <artifactId>timestamp</artifactId>
    <version>3.2.6</version>
   </dependency>
```

- *Apache Buildr* [https://buildr.apache.org/]

```
'org.demoiselle.signer:timestamp:jar:3.2.6'
```

- *Apache Ivy* [http://ant.apache.org/ivy/]

```
<dependency org="org.demoiselle.signer" name="timestamp" rev="3.2.6" />
```

- *Groovy Grape* [http://docs.groovy-lang.org/latest/html/documentation/grape.html]

```
@Grapes(@Grab(group='org.demoiselle.signer', module='timestamp', version='3.2.6'))
```

- *Gradle/Grails* [https://github.com/grails/grails-gradle-plugin]

```
<dependency org="org.demoiselle.signer" name="timestamp" rev="3.2.6" />
```

- *Scala SBT* [http://www.scala-sbt.org/]

```
libraryDependencies += "org.demoiselle.signer" % "timestamp" % "3.2.6"
```

- *Leiningen* [https://leiningen.org/]

```
<[org.demoiselle.signer/timestamp "3.2.6"]
```

Caso não esteja utilizando nenhum outro tipo de gerenciador (estava morando numa caverna nos últimos dez anos), pode baixar o .jar do repositório:

http://repo1.maven.org/maven2/org/demoiselle/signer/timestamp/

Funcionalidades

11.1. Carimbo de tempo com componente *policy-impl-cades*

O código abaixo demostra como pode ser feita a chamada:

```
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificates(CertificateChain);
 signer.setPrivateKey(PrivateKey);
// com carimbo de tempo
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RT_CADES_2_2);
// Assinatura desatachada
byte[] signature = signer.doDetachedSign(fileToSign);
```

11.2. Requisições de carimbo de tempo

11.2.1. Para uma assinatura padrão CADES

Para obter um carimbo de tempo para uma assinatura CADeS, basta enviar o conteúdo da assinatura. O retorno será a assinatura com o carimbo embutido, veja no exemplo abaixo:

```
String fileSignatureDirName = "local_e_nome_do_arquivo_da_assinatura";
 byte[] signatureFile = readContent(fileSignatureDirName); // implementar o m#todo
de leitura do arquivo
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
varCAdESTimeStampSigner.setCertificates(CertificateChain);
varCAdESTimeStampSigner.setPrivateKey(PrivateKey);
byte[] signatureWithTimeStamp = varCAdESTimeStampSigner
        .doTimeStampForSignature(signatureFile);
```

11.2.2. Para um conteúdo

É possível também obter o carimbo para o conteúdo de uma informação. Neste caso o carimbo não estará associado à assinatura

Nota

A ICP-Brasil não traz nenhuma norma relativa a este tipo de carimbo, o que existe são as política para assinatura.

O código abaixo demosntra como é feita a requisição. O retorno é o arquivo do tipo TimeStampToken descrito na *RFC 3161* [https://www.ietf.org/rfc/rfc3161.txt]

```
            String fileDirName = "local_e_nome_do_arquivo_para_carimbar";
               byte[] content = readContent(fileDirName); // implementar m#todo para leitura
do conte#do
            CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
            varCAdESTimeStampSigner.setCertificates(CertificateChain);
            varCAdESTimeStampSigner.setPrivateKey(PrivateKey);
          byte[] timeStampForContent = varCAdESTimeStampSigner.doTimeStampForContent(content);
```

11.2.3. Para o resumo (hash) de um conteúdo

A outra funcionalidade disponível permite enviar o resumo já calculado.

```
        String fileDirName = "local_e_nome_do_arquivo";
       byte[] content = readContent(fileDirName); // implementar metodo para leitura do arquivo
        // gera o hash do conteudo
        java.security.MessageDigest md = java.security.MessageDigest
                    .getInstance(DigestAlgorithmEnum.SHA_256.getAlgorithm());
        byte[] hash = md.digest(content);
        CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
        varCAdESTimeStampSigner.setCertificates(CertificateChain);
        varCAdESTimeStampSigner.setPrivateKey(PrivateKey);
       byte[] timeStampForContent = varCAdESTimeStampSigner.doTimeStampFromHashContent(hash);
```

11.3. Validdação do Carimbo de tempo com componente *policy-impl-cades*

Veja a sessão de *Validação*

11.4. Validações de carimbo de tempo

11.4.1. Para uma assinatura padrão CADES

```
            String fileSignatureDirName = "local_e_nome_do_arquivo_da_assinatura";
            byte[] signatureFile = readContent(fileSignatureDirName);
            CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
            List<Timestamp> listTimeStamp = varCAdESTimeStampSigner.checkTimeStampOnSignature(signatureFile);
            if (!listTimeStamp.isEmpty()){
                for (Timestamp ts : listTimeStamp){
                    System.out.println(ts.toString());
                }
            }
```

11.4.2. Para um conteúdo

Para validar o carimbo associado a um contéudo, é preciso enviar ao componente, o contéudo e a assinatura, conforme o código abaixo:

```
        String fileTimeStampDirName = "local_e_nome_do_arquivo_da_assinatura";
        String fileContentDirName = "local_e_nome_do_arquivo_assinado";
        byte[] timeStampFile = readContent(fileTimeStampDirName);
        byte[] content = readContent(fileContentDirName);
        CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
        Timestamp varTimeStamp = varCAdESTimeStampSigner.checkTimeStampWithContent(timeStampFile, content);
```

11.4.3. Para o resumo (hash) de um conteúdo

```
String fileTimeStampDirName = "local_e_nome_do_arquivo_da_assinatura";
String fileContentDirName = "local_e_nome_do_arquivo_assinado";
byte[] timeStampFile = readContent(fileTimeStampDirName);
byte[] content = readContent(fileContentDirName);
// gera o hash do conteudo
java.security.MessageDigest md = java.security.MessageDigest
        .getInstance(DigestAlgorithmEnum.SHA_256.getAlgorithm());
byte[] hash = md.digest(content);
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
Timestamp varTimeStamp = varCAdESTimeStampSigner.checkTimeStampWithHash(timeStampFile, hash);
```

Parte VI. Demoiselle Signer Cryptography

O *cryptography* é um componente corporativo que provê um mecanismo simplificado de criptografia de dados. Neste contexto o componente atua nos dois principais tipos de algoritmos, os algoritmos de chave simétrica e algoritmos de chave assimétrica.

O componente provê as funções de cifragem e decifragem utilizando algoritmos simétricos ou também chamados de algoritmos de chave-simétrica, ou seja, os que utilizam uma mesma chave para cifrar e decifrar as mensagens.

Além disso o componente utiliza algoritmos de hash para criptografia de dados com a finalidade de criar um valor único que identifique um dado original. Este recurso é recomendado para finalidades de autenticação, nas quais deseja-se armazenar as senhas criptografas por meio de um valor hash. Também é possível construir hash de arquivos no intuito de avaliar sua integridade física.

O componente também realiza as funções de cifragem e decifragem por meio de algoritmos de chave-assimétrica. Neste processo é necessário um par de chaves para realizar a cifragem e decifram das mensagens. A primeira chave é denominada chave privada, ela é de posse exclusiva de seu detentor e ninguém mais a conhece. A segunda chave do par e denominada de chave pública e pode ser enviada a qualquer indivíduo.

Configuração do Cryptography

12.1. Instalação do componente

Para instalar o componente *Demoiselle Signer Cryptography* na aplicação, basta adicionar a sua dependência de acordo com o gerenciador de projetos:

- *Apache-Maven* [https://maven.apache.org/]

```
      <dependency>
   <groupId>org.demoiselle.signer</groupId>
   <artifactId>cryptography</artifactId>
   <version>3.2.6</version>
  </dependency>
```

- *Apache Buildr* [https://buildr.apache.org/]

```
'org.demoiselle.signer:cryptography:jar:3.2.6'
```

- *Apache Ivy* [http://ant.apache.org/ivy/]

```
<dependency org="org.demoiselle.signer" name="cryptography" rev="3.2.6" />
```

- *Groovy Grape* [http://docs.groovy-lang.org/latest/html/documentation/grape.html]

```
@Grapes(@Grab(group='org.demoiselle.signer', module='cryptography', version='3.2.6'))
```

- *Gradle/Grails* [https://github.com/grails/grails-gradle-plugin]

```
<dependency org="org.demoiselle.signer" name="cryptography" rev="3.2.6" />
```

- *Scala SBT* [http://www.scala-sbt.org/]

```
libraryDependencies += "org.demoiselle.signer" % "cryptography" % "3.2.6"
```

- *Leiningen* [https://leiningen.org/]

```
<[org.demoiselle.signer/cryptography "3.2.6"]
```

Caso não esteja utilizando nenhum outro tipo de gerenciador (estava morando numa caverna nos últimos dez anos), pode baixar o .jar do repositório:

http://repo1.maven.org/maven2/org/demoiselle/signer/cryptography/

12.2. Customização das implementações

O componente Demoiselle Signer Cryptography possui implementações padrões às funcionalidades de criptografia, entretanto é possível definir outras implementações. Neste caso é necessário informar, ou como variável de ambiente, ou com variável da JVM, qual a implementação das interfaces: `org.demoiselle.signer.cryptography.Cryptography` e `org.demoiselle.signer.cryptography.Digest`.

Por padrão, as respectivas implementações são: `org.demoiselle.signer.cryptography.implementation.CriyptographyImpl` e `org.demoiselle.signer.cryptography.implementation.DigestImpl`.

Veja a configuração por meio de variável de ambiente para definição da implementação da interface `org.demoiselle.signer.cryptography.Cryptography`

Tabela 12.1. Exemplo com Variável de Ambiente

Ambiente	Variável de ambiente
Linux	export cryptography.implementation = org.demoiselle.signer.cryptography.implementation.CriyptographyImpl
Windows	set cryptography.implementation = org.demoiselle.signer.cryptography.implementation.CriyptographyImpl

Tabela 12.2. Exemplo com Variável JVM

Ambiente	Variável JVM
Linux	-cryptography.implementation= org.demoiselle.signer.cryptography.implementation.MyCriptographyImpl
Windows	-cryptography.implementation= org.demoiselle.signer.cryptography.implementation.MyCriptographyImpl

Veja a configuração por meio de variável de ambiente para definição da implementação da interface `org.demoiselle.signer.cryptography.Digest`.

Tabela 12.3. Exemplo com Variável de Ambiente

Ambiente	Variável de ambiente
Linux	export digest.implementation= org.demoiselle.signer.cryptography.implementation.MyDigestImpl
Windows	set digest.implementation= org.demoiselle.signer.cryptography.implementation.MyDigestImpl

Tabela 12.4. Exemplo com Variável JVM

Ambiente	Variável JVM
Linux	-messageDigest.implementation= org.demoiselle.signer.cryptography.implementation.MyDigestImpl
Windows	-messageDigest.implementation= org.demoiselle.signer.cryptography.implementation.MyDigestImpl

Funcionalidades

13.1. A Criptografia Simétrica

A cifragem e decifragem de dados são providas pela interface `Cryptography` e o componente se utiliza de uma fábrica dessa interface. Segue abaixo um exemplo ilustrativo:

```
public class App {

        public static void main(String[] args) {
                String frase = "conteudo original";

               Cryptography cryptography = CryptographyFactory.getInstance().factoryDefault();

                /* Geracao da chave unica */
                Key key = cryptography.generateKey();
                cryptography.setKey(key);

                /* Cifragem */
                byte[] conteudo_criptografado = cryptography.cipher(frase.getBytes());
                System.out.println(conteudo_criptografado);

                /* Decifragem */
             byte[] conteudo_descriptografado = cryptography.decipher(conteudo_criptografado);
                System.out.println(new String(conteudo_descriptografado));
        }
}
```

Os métodos `cipher` e `decipher` recebem como entrada um array de bytes e retornam o array de bytes processado.

Para que a criptografia simétrica seja realizada é necessário o uso de uma única chave, para criptografar e descriptografar. Neste caso, é necessário gerar a chave através do método `generateKey`.

Caso não seja informado o algoritmo de criptografia o componente utilizará como padrão o algoritmo *AES (Advanced Encryption Standard)*. Caso necessite utilizar outro algoritmo invoque o método `setAlgorithm` informando um `SymmetricAlgorithmEnum` ou um `AsymmetricAlgorithmEnum.`

```
public class App {

        public static void main(String[] args) {
                String frase = "conteudo original";

               Cryptography cryptography = CryptographyFactory.getInstance().factoryDefault();

                /* Alterando algoritmo */
                cryptography.setAlgorithm(SymmetricAlgorithmEnum.TRI_DES);

                /* Geracao da chave unica */
                Key key = cryptography.generateKey();
```

```
                cryptography.setKey(key);

                byte[] conteudo_criptografado = cryptography.cipher(frase.getBytes());
                System.out.println(conteudo_criptografado);

             byte[] conteudo_descriptografado = cryptography.decipher(conteudo_criptografado);
                System.out.println(new String(conteudo_descriptografado));
        }
}
```

Caso as opções de criptografia definidas pelo `SymmetricAlgorithmEnum` não atendam é possível customizar os parâmetros de criptografia através dos métodos `setAlgorithm`, `setKeyAlgorithm` e `setSize`.

```
public static void main(String[] args) {
                String frase = "conteudo original";

               Cryptography cryptography = CryptographyFactory.getInstance().factoryDefault();

                /* Customizacao de parametros */
                cryptography.setAlgorithm("AES/ECB/PKCS5Padding");
                cryptography.setKeyAlgorithm("AES");
                cryptography.setSize(128);

                /* Cifragem */
                byte[] conteudo_criptografado = cryptography.cipher(frase.getBytes());
                System.out.println(conteudo_criptografado);

                /* Decifragem */
             byte[] conteudo_descriptografado = cryptography.decipher(conteudo_criptografado);
                System.out.println(new String(conteudo_descriptografado));
        }
```

13.2. A Criptografica Assimétrica

Na criptografia assimétrica é necessário um par de chaves para realizar a cifragem e decifram das mensagens. A primeira chave é denominada chave privada e é de posse exclusiva de seu detentor. A segunda chave do par é denominada de chave pública e pode ser enviada a qualquer indivíduo.

```
        /* Cifragem */
        Cryptography crypto = CryptographyFactory.getInstance().factoryDefault();
        crypto.setKey(privateKey);
        byte[] conteudoCriptografado = crypto.cipher("SERPRO".getBytes());
        System.out.println(conteudoCriptografado);

        /* Decifragem */
        Cryptography crypto2 = CryptographyFactory.getInstance().factoryDefault();
        crypto2.setKey(publicKey);

        byte[] conteudoDescriptografado = crypto2.decipher(conteudoCriptografado);
        System.out.println(new String(conteudoDescriptografado));
```

Perceba a utilização do método `setKey` para informar qual chave será utilizada no processo de cifragem e decifragem. Vale lembrar que na criptografia assimétrica a cifragem realizada com a chave privada só poderá ser decifrada com a chave pública e vice-versa.

Na sequência, demonstramos a utilização do componente com certificados digitais do tipo A1 e A3.

13.2.1. Certificados A1

O certificado A1 é aquele que encontra-se armazenado no sistema de arquivo do sistema operacional. Para exemplificar sua manipulação, segue o código abaixo:

```
	public static void main(String[] args) {

		try {
			/* Obtendo a chave publica */
			File file = new File("/home/{usuario}/public.der");
			byte[] encodedPublicKey = new byte[(int) file.length()];
			InputStream inputStreamPublicKey = new FileInputStream(file);
			inputStreamPublicKey.read(encodedPublicKey);
			inputStreamPublicKey.close();
		   X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(encodedPublicKey);
			KeyFactory kf = KeyFactory.getInstance("RSA");
			PublicKey publicKey = kf.generatePublic(publicKeySpec);

			/* Obtendo a chave privada */
			file = new File("/home/{usuario}/private.pk8");
			byte[] encodedPrivateKey = new byte[(int) file.length()];
			InputStream inputStreamPrivateKey = new FileInputStream(file);
			inputStreamPrivateKey.read(encodedPrivateKey);
			inputStreamPrivateKey.close();
			PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(encodedPrivateKey);
			kf = KeyFactory.getInstance("RSA");
			PrivateKey privateKey = kf.generatePrivate(privateKeySpec);

			/* Cifragem */
		    Cryptography cripto = CryptographyFactory.getInstance().factoryDefault();
			crypto.setAlgorithm(AsymmetricAlgorithmEnum.RSA);
			crypto.setKey(privateKey);
			byte[] conteudoCriptografado = crypto.cipher("SERPRO".getBytes());
			System.out.println(conteudoCriptografado);

			/* Decifragem */
		    Cryptography cripto2 = CryptographyFactory.getInstance().factoryDefault();
			crypto2.setAlgorithm(AsymmetricAlgorithmEnum.RSA);
			crypto2.setKey(publicKey);

		    byte[] conteudoDescriptografado = crypto2.decipher(conteudoCriptografado);
			System.out.println(new String(conteudoDescriptografado));

		} catch (Exception e) {
			e.printStackTrace();
		    Assert.assertTrue("Configuracao nao carregada: " + e.getMessage(), false);
		}
	}
```

Neste exemplo é demonstrada a obtenção das chaves pública e privada do certificado A1. Note que, apesar do código para manipulação dos certificados, a forma de uso do componente *Demoiselle Cryptography* para cifragem e decifragem de mensagens é a mesma.

13.2.2. Certificados A3

O certificado A3 é armazenado em dispositivos eletrônicos como smart card ou tokens usb que criptografam o certificado provendo maior segurança. No exemplo abaixo utilizamos o componente *Demoiselle Core* para obtenção do keyStore a partir de um token usb. Você pode ver mais sobre esse componente em *Capítulo 1, Configuração do Signer-Core*

```
	public static void main(String[] args) {

		/* Senha do dispositivo */
		String PIN = "senha_do_token";
		try {

			/* Obtendo a chave privada */
			KeyStore keyStore = KeyStoreLoaderFactory.factoryKeyStoreLoader().getKeyStore(PIN);
			String alias = (String) keyStore.aliases().nextElement();
		 PrivateKey privateKey = (PrivateKey) keyStore.getKey(alias, PIN.toCharArray());

			/* Obtendo a chave publica */
			CertificateLoader cl = new CertificateLoaderImpl();
			X509Certificate x509 = cl.loadFromToken(PIN);
			PublicKey publicKey = x509.getPublicKey();

			/*Configurando o Criptography */
		    Cryptography crypto = CryptographyFactory.getInstance().factoryDefault();
			crypto.setAlgorithm(AsymmetricAlgorithmEnum.RSA);
			crypto.setProvider(keyStore.getProvider());

			/* criptografando com a chave privada */
			crypto.setKey(privateKey);
			byte[] conteudoCriptografado = crypto.cipher("SERPRO".getBytes());
			System.out.println(conteudoCriptografado);

			/* descriptografando com a chave publica */
			crypto.setKey(publicKey);
			byte[] conteudoAberto = crypto.decipher(conteudoCriptografado);
			System.out.println(new String(conteudoAberto));

		} catch (UnrecoverableKeyException e) {
			e.printStackTrace();
		} catch (KeyStoreException e) {
			e.printStackTrace();
		} catch (NoSuchAlgorithmException e) {
			e.printStackTrace();
		}
	}
```

O componente utiliza como padrão o provider SUN JCE, mas caso necessite de outro provider utilize o método `setProvider` da classe `Cryptography`.

No exemplo acima foi utilizado o método `setProvider` para informar o provedor, ou seja, quem executará os algoritmos de criptografia. Até então, nos exemplos anteriores, o provedor era a biblioteca SUN JCE contida na própria JVM. Como o token é o único a ter acesso a chave privada do certificado ele também será o único capaz de executar os processos de cifragem e decifragem.

Desta forma foi utilizado o objeto KeyStore do próprio token usb para informar o novo provider ao Cryptography.

13.3. Geração de Hash

13.3.1. Hash simples

A criptografia de hash tem a finalidade de criar um valor único que identifique um dado original. Este recurso pode ser utilizado por exemplo para finalidades de autenticação nas quais deseja-se armazenar as senhas cifradas por meio de um valor hash.

A fábrica `DigestFactory` do componente constrói um objeto padrão do tipo Digest que calcula o valor hash de um array de bytes retornando outro array de bytes.

```
public static void main(String[] args) {
        Digest digest = DigestFactory.getInstance().factoryDefault();
        byte[] resumo = digest.digest("SERPRO".getBytes());
        System.out.println(resumo);
}
```

Caso queira obter o valor hash no formato caractere hexadecimal utilize o método `digestHex` . Este formato é bastante utilizado para representar o hash de arquivos.

```
public static void main(String[] args) {
        Digest digest = DigestFactory.getInstance().factoryDefault();
        String resumo = digest.digestHex("SERPRO".getBytes());
        System.out.println(resumo);
}
```

13.3.2. Hash de arquivo

O hash de arquivo pode ser utilizado quando se deseja verificar a integridade física de um arquivo. No caso de ferramentas de download é possível ao final do processo de transferência de dados, verificar se o arquivo obtido apresenta o mesmo hash do arquivo original.

A interface `Digest` possui os métodos `digestFile` e `digestFileHex` para retornar respectivamente o valor hash em array de bytes ou caracter hexadecimal:

```
public static void main(String[] args) {
        Digest digest = DigestFactory.getInstance().factoryDefault();
        digest.setAlgorithm(DigestAlgorithmEnum.SHA_256);
      String resumo = digest.digestFileHex(new File("/home/{usuario}/relatorio.pdf"));
        System.out.println(resumo);
```

```
        }
```

Os algoritmos de hash recomendados pelo componente são definidos pelo enum `DigestAlgorithmEnum`. Caso não seja informado o componente utilizará o "SHA-1" por ser considerado mais seguro quanto a quebras em relação ao MD5.