Demoiselle Signer Versão 4.3.0

Componente para certifi-cação digital ICP-BRASIL

Denis A. Altoé Falqueto
Ednara Oliveira
Emerson Sachio Saito <emerson.saito@gmail.com>
Erick Luiz Flores Guimarães
Fabiano Kuss
Fábio Nogueira de Lucena
Humberto Pacheco
José Rene Campanario
Julian Santos
Thiago Laubstein Ribeiro

Demoiselle Signer Versão 4.3.0: Componente para certificação digital ICP-BRASIL

por Denis A. Altoé Falqueto, Ednara Oliveira, Emerson Sachio Saito, Erick Luiz Flores Guimarães, Fabiano Kuss, Fábio Nogueira de Lucena , Humberto Pacheco, José Rene Campanario, Julian Santos e Thiago Laubstein Ribeiro

Resumo

Índice

Lista de Figuras

Lista de Tabelas

Sobre o Demoiselle Signer

O Demoiselle Signer é um componente para facilitar a geração e a validação de assinaturas digitais. O componente está implementado em conformidade com as políticas da ICP-Brasil. Consulte as resoluções da ICP-Brasil [http://www.iti.gov.br/legislacao/documentos-principais] para detalhes.

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

- **core:** fornece as interfaces básicas 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 ICP-Brasil.
- **policy-impl-CAdES:** permite geração e validação de assinaturas digitais (conforme uma política) no formato CAdES.
- **policy-impl-PAdES:** permite geração e validação de assinaturas digitais (conforme uma política) no formato PBAD-PAdES.
- **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 ICP-Brasil válidas.
- **timestamp:** disponibiliza as funcionalidades para obtenção de carimbos de tempo (fornecidos por uma autoridade de carimbo de tempo).
- **cryptography:** provê funcionalidades de criptografia.
- **Integração WEB:** solução indicada para acesso aos certificados do usuário.

Funcionalidades ainda não implementadas no componente:

- Assinatura no padrão XAdES (assinatura em XML).

Nota

Caso queira baixar uma versão desta documentação em formato PDF clique

aqui: [https://www.frameworkdemoiselle.gov.br/v3/signer/docs/pdf/signer-reference-4.3.0.pdf]

Parte I. Core

Este componente provê uma API (*Application Programming Interface*) para facilitar o tratamento de Certificados Digitais em aplicações Java. Seus objetivos incluem o carregamento, validação, e obtenção de dados para certificados digitais.

Índice

Capítulo 1. Configuração do Signer-Core

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>signer-core</artifactId>
    <version>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:signer-core:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="signer-core" rev="4.3.0" />
```

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

```
@Grapes(@Grab(group='org.demoiselle.signer', module='signer-core', version='4.3.0
```

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

```
<dependency org="org.demoiselle.signer" name="signer-core" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "signer-core" % "4.3.0"
```

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

```
[org.demoiselle.signer/signer-core "4.3.0"]
```

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

https://repo1.maven.org/maven2/org/demoiselle/signer/signer-core/

Capítulo 2. 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, valid
```

ou

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

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);
```

O Certificado Digital

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.NAME)
    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);
```

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

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

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

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().
```

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().
```

Validadores

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.

PeriodValidator

Verifica a data de validade do certificado.

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.

Repositório Online

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

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.

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);
```

Capítulo 3. Funcionalidades relativas ao Keystore

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.factoryKeyStoreLoade
```

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

```
KeyStoreLoader keyStoreLoader = KeyStoreLoaderFactory.factoryKeyStoreLoade
```

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/keyst
```

```
KeyStore keyStore = KeyStoreLoaderFactory.factoryKeyStoreLoader(new Fi
```

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 “Lista de Drivers” .

Abaixo temos exemplos de uso.

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

```
KeyStore keyStore = KeyStoreLoaderFactory.factoryKeyStoreLoader().getK
```

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 NUMB
```

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

Importante

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

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().getK
```

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 “Desabilitar a camada de acesso SunMSCAPI” .

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 predefinidos 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
/usr/lib/libwdpkcs.so
/usr/local/lib64/libwdpkcs.so

Caminho (Path) do Driver
/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/libaetpkss.so.3
/usr/lib/libaetpkss.so.3.0
/usr/lib/opensc-pkcs11.so
/usr/lib/pkcs11/opensc-pkcs11.so
/usr/local/ngsrv/libepsng_p11.so.1.2.2
/usr/local/AWP/lib/libOcsCryptoki.so
/usr/lib/libscmccid.so
/usr/lib64/libeToken.so
/opt/ePass2003-Castle-20141128/i386/redist/libcastle.so.1.0.0
/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
/usr/lib/x86_64-linux-gnu/pkcs11/opensc-pkcs11.so
/usr/lib/libneoidp11.so
/usr/lib/x86_64-linux-gnu/pkcs11/opensc-pkcs11.so
/usr/lib/opensc/openscpkcs11.so

Tabela 3.2. Drivers predefinidos 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

Caminho (Path) do Driver
/system32/SerproPkcs11.dll

Tabela 3.3. Drivers predefinidos 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
//Applications//NeoID Desktop.app//Contents//Java//tools//macos//libneoidp11.dylib

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 Drive
```

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()
```

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
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

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.

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.

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.

Primeiro é preciso importar o certificado A1 (arquivo) no Firefox, conforme as orientações

Figura 3.4. Abrindo menu de configurações do Firefox

Figura 3.5. Abrindo opção "Preferências"

Figura 3.6. Abrindo item "Avançado"

Figura 3.7. Abrindo aba "Certificados"

Figura 3.8. Clicando no botão "Ver Certificados"

Figura 3.9. Selecionando a aba "Seus Certificados"

Figura 3.10. Clicando no botão "Importar..."

Figura 3.11. Selecionando o arquivo de certificado

Após feita a importação no Firefox, para viabilizar o acesso em um sistema tipo LINUX, 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 Linux:

```
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'"
showInfo=true
```

Para o Mac OS, também depois de importar no Firefox, a seguinte configuração:

```
name = Provedor
slot = 2
library = /Applications/Firefox.app/Contents/MacOS/libsoftokn3.dylib
nssArgs = "configdir='/Users/<usuario>/Library/Application Support/Firefox/Profile
```

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.

Caso as configurações não estejam fazendo efeito, um último recurso é fechar o Firefox apagar o arquivo de profile (as configurações serão perdidas, faça backup do que for possível, antes)

Utilizando certificados armazenados em Token / Smart-Card 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.

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.

Também é possível desabilitar o MSCAPI através de uma configuração do SIGNER. Conforme mostrado a seguir:

```
...
            import org.demoiselle.signer.core.keystore.loader.configur
            ...
            Configuration.setMSCAPI_ON(false);
            ...
```

A seguir demonstramos a configuração para o sistema Windows:

Figura 3.12. Desabilitando a camada MSCAPI (-Dmscapi.disabled=true)

Capítulo 4. Configurações de Ambiente

Configurações de Proxy

Para o caso do ambiente de rede que esteja rodando a aplicação esteja sob um Proxy é possível fazer a configuração das seguintes formas:

- *Programaticamente*

```
....
```

```
...
import org.demoiselle.signer.core.util.Proxy;

Proxy.setProxyEndereco("endereco_ou_ip");
Proxy.setProxyPorta("numero_da_porta");
Proxy.setProxyUsuario("usuario"); // Caso necessário
Proxy.setProxySenha("senha"); // Caso necessário
Proxy.setProxy();
...
```

- *Setando as variáveis de ambiente*

signer.proxy.host
signer.proxy.port
signer.proxy.user
signer.proxy.password

OU
SIGNER_PROXY_HOST
SIGNER_PROXY_PORT
SIGNER_PROXY_USER
SIGNER_PROXY_PASSWORD

Repositório local de LPA (Lista de Políticas de Assinatura)

As Listas de Política de Assinaturas, são atualizadas trimestralmente pela ICP-BRASIL, o componenente mantem estes arquivos internamente mas haverão momentos que estes arquivos estarão desatualizados. Para evitar que seja necessário atualizar a versão do componente exclusivamente para este propósito, há um mencanismo de recuperação dos arquivos diretamente do site da ICP-BRASIL. E esta funcionalidade também armazena localmente o arquivo para que não seja baixado todas as vezes que o componente for acionado. O diretório padrão do aplicativo é:

/tmp/lpas/

Para alterar o local padrão, existem duas formas de fazê-lo:

- *Programaticamente*

```
...
```

```
...
```

```
org.demoiselle.signer.core.repository.ConfigurationRepo c
configRepo.setLpaPath("/tmp/meudir/");
...
```

- *Setando a variável de ambiente*

  signer.repository.lpa.path

  OU
  SIGNER_REPOSITORY_LPA_PATH

Modo de recuparação das LPAs (Lista de Politicas de Assinatura)

O mecanismo acima permite fazer um armazenamento local das LPAs, na maioria dos casos isso é a melhor estratégia. Mas caso haja algum motivo para que sejam recuparadas cada vez que fizer a Assinatura isso também é possível

Por padrão esta opção vem desativada. Existem duas formas de ativá-la:

- *Programaticamente*

  ```
  ...
  ...
  org.demoiselle.signer.core.repository.ConfigurationRepo c
  configRepo.setOnlineLPA(false);
  ...
  ```

- *Setando variável de ambiente*

  signer.repository.lpa.online

  OU
  SIGNER_REPOSITORY_LPA_ONLINE

Cache no CAManager

Em ambientes onde são executadas muitas assinaturas, algumas verificações repetitivas - que não sofrem alteração entre uma assinatura e outra - podem ser colocadas em cache.

Atualmente as seguintes verificações são armazenadas:

- recuperação da cadeia de certificados associadas a um certificado
- verificação de assinatura de um certificado por um outro certificado

Por padrão esta opção vem desativada. Existem duas formas de ativá-la:

- *Programaticamente*

  ```
  ...
  ...
  org.demoiselle.signer.core.ca.manager.CAManagerConfigurat
  config.setCached(true);
  ```

```
...
```

- *Setando a variável de ambiente*

  signer.camanager.cached

  OU
  SIGNER_CAMANAGER_CACHED

O cache pode ser invalidado a qualquer momento através do método invalidate:

```
...
                        ...
                        org.demoiselle.signer.core.ca.manager.CAManagerCache cache
                        cacheManager.invalidate();
                        ...
```

Timeout para download de LCRs (Lista de Certificados Revogados)

Para definir um limite de tempo para download das LCR , existem duas formas:

- *Programaticamente*

  ```
  ...
                          ...
                          org.demoiselle.signer.core.repository.ConfigurationRepo c
                          configRepo.setCrlTimeOut(10000); // Valor em milisegundos
                          ...
  ```

- *Setando variável de ambiente*

  signer.crl.connection.timeout

  OU
  SIGNER_CRL_CONNECTION_TIMEOUT

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.

Índice

Capítulo 5. Configuração do Policy CAdES

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>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:policy-impl-cades:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="policy-impl-cades" rev="4.3.0" />
```

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

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

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

```
<dependency org="org.demoiselle.signer" name="policy-impl-cades" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "policy-impl-cades" % "4.3.0"
```

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

```
<[org.demoiselle.signer/policy-impl-cades "4.3.0"]
```

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

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

Capítulo 6. 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)

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

/* 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);
```

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 arq
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_3` Refere-se à Assinatura Digital de Referência Básica versão 2.3;
- `AD_RT_CADES_2_3` Refere-se à Assinatura Digital de Referência Temporal (com carimbo de tempo) versão 2.3;

```
byte[] content = readContent("texto.txt"); /* implementar metodo de leitura de arq
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificates(certificateChain);
signer.setPrivateKey(privateKey);
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_3);
byte[] signature = signer.doDetachedSign(this.content);
```

Importante

Caso não seja especificada nenhuma política, o componente assumirá a política padrão AD_RB_CADES_2_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 arq
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificates(certificateChain);
signer.setPrivateKey(privateKey);
byte[] signature = signer.doAttachedSign(fileToSign);
```

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 arq
/* 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_3);
byte[] signature = signer.doHashSign(hash);
```

Importante

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

Co-Assinatura em arquivo único de assinatura .

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 arq
byte[] signatureFile = readContent("fileSignature.p7s"); /* implementar metodo de
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();
signer.setCertificate(certificate);
signer.setPrivateKey(privateKey);
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_3);
byte[] signature = signer.doDetachedSign(content, signatureFile);
```

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 arq
byte[] signatureFile = readContent("fileSignature.p7s"); /* implementar metodo de
/* 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_3);
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.

Validação de assinatura PKCS7 sem o conteúdo anexado (detached)

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

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 a
 private Date signDate;  // data do equipamento no momento da geração das assinatu
    private Timestamp timeStampSigner = null; // Carimbo de tempo da assinatura, q
    private SignaturePolicy signaturePolicy; // Politica ICP-BRASIL usada para ger
    private LinkedList<String> validatorErrors = new LinkedList<String>();  // Lis
```

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
CAdESChecker checker = new CAdESChecker();
List<SignatureInformations> signaturesInfo = checker.checkAttachedSignature(signat
```

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 arq
byte[] signature = readContent("texto.pkcs7"); /* implementar metodo de leitura de
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(SignerAl
```

Tratando os resultados da valiação

Como é possível que um mesmo arquivo possa contar várias assinaturas (PAdES principalmente), só será gerada exceção quando a assinatura estiver comprometida Nos demais casos, o Demoiselle-Signer irá devolver o resultado numa lista de objetos SignatureInformations. Essa classe contém os seguintes atributos:

- chain;

  Lista X509Certificate com a cadeia completa do certificado do Assinante

- signDate

  A data do equipamento onde foi gerada a assinatura, e serve apenas como referência, não tem nenhuma validade legal

- timeStampSigner

  É o carimbo do Tempo (Timestamp) incluído na Assinatura, é a prova legal da data e hora que a Assinatura foi gerada.

- signaturePolicy;

  A política (SignaturePolicy) que foi usada para gerar a Assinatura

- notAfter;

  A data de validade do Certificado do Assinante

- validatorWarnins

  Lista de Avisos. A assinatura pode estar correta mas não foi possível verificar algum atributo exigido por uma política da ICP-Brasil, que serão listados aqui

- validatorErrors

  Lista de Erros. A assinatura pode estar correta mas não foi possível verificar alguma condição de validação exigida pela ICP-Brasil

- invalidSignature

  valor booleano, que indica que Assinatura não está válida

- icpBrasilcertificate

  BasicCertificate do Assinante

Cabe ao sistema com base nos avisos ou erros, aceitar ou não a Assinatura. Apesar de existirem as políticas, qualquer tipo de Assinatura gerada com um certificado ICP-Brasil tem validade legal. A seguir temos um fragmento de código que demonstra esta validação.

```
List<SignatureInformations> signaturesInfo = checker
    .checkDetachedSignature(fileToVerify, signatureFile);

  if (signaturesInfo != null) {
   System.out.println("A assinatura foi validada. e retornou resultados");
   for (SignatureInformations si : signaturesInfo) {
    System.out.println(si.getSignDate());
    if (si.getTimeStampSigner() != null) {
     System.out.println("Serial"
       + si.getTimeStampSigner().toString());
    }
    System.out.println("informações do assinante:");
    BasicCertificate certificate = si.getIcpBrasilcertificate();
    if (!certificate.isCACertificate()) {
     if (certificate.hasCertificatePF()) {
      System.out.println("CPF: "+certificate.getICPBRCertificatePF().getCPF());
      System.out.println("Titulo de Eleitor: "+certificate.getICPBRCertificatePF()
     }
     if (certificate.hasCertificatePJ()) {
      System.out.println("CNPJ: "+certificate.getICPBRCertificatePJ().getCNPJ());
     }
    }
    // Carimbo do tempo
    if(si.getTimeStampSigner()!= null) {

     System.out.println(si.getTimeStampSigner().toString());
    }
    // A assinatura pode estar correta mas não foi possível verificar algum atribu
    for (String valErr : si.getValidatorErrors()) {
     System.err.println("+++++++++++++++ ERROS ++++++++++++++++++");
     System.err.println(valErr);
    }
    //A assinatura pode estar correta mas não foi possível verificar alguma condiç
    for (String valWarn : si.getValidatorWarnins()) {
     System.err.println("+++++++++++++++ AVISOS ++++++++++++++++++");
     System.err.println(valWarn);
    }

   }
```

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 à uma assinatura PKCS7. Essa funcionalidade pode ser útil quando é necessário retirar/extrair 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[] attachedSignature = readContent("texto.pkcs7"); /* implementar metodo de le
CAdESChecker checker = new CAdESChecker();
```

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

/* Para extrair o conteudo original sem validar a assinatura */
byte[] content = checker.getAttached(attachedSignature, 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]

Capítulo 7. Exemplos de Uso

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

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;
```

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();
```

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);
```

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
```

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);
```

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();
```

Carregar um certificado digital de um arquivo

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

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. Implementação das Políticas PAdES

O componente *Policy-Impl-PAdES* foi desenvolvido para atender às necessidades de assinatura digital no âmbito da ICP-Brasil. Conforme as políticas para o padrão PBAD-PAdES [https://www.iti.gov.br/images/repositorio/legislacao/documentos-principais/15.3/DOC-ICP-15.03_-_Versao_7.4_REQ_DAS_POL_DE_ASSIN_DIG_NA_ICP-BRASIL.pdf]. Esse padrão é utilizado para Assinaturas Digitais em Arquivos PDF, conforme define a ICP-Brasil, não é o mesmo padrão que as ferramentas como Adobe Reader geram. A validação dessas assinaturas no Adobe-Reader dependem de um plugin disponibilizado pelo ITI [https://www.iti.gov.br/aplicativos/111-aplicativos/4105-plugin-pades], ou através do Site: https://verificador.iti.gov.br/ ou utilizando o Assinador SERPRO. [http://www.serpro.gov.br/assinador-digital/]

Nota

Este componente não manipula o arquivo PDF, apenas retorna o contéudo da Assinatura em confirmidade com a ICP-Brasil.

Índice

Capítulo 8. Configuração do Policy PAdES

Instalação do componente

Para utilizar o componente *policy-impl-pades* 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-pades</artifactId>
    <version>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:policy-impl-pades:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="policy-impl-pades" rev="4.3.0" />
```

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

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

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

```
<dependency org="org.demoiselle.signer" name="policy-impl-pades" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "policy-impl-pades" % "4.3.0"
```

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

```
<[org.demoiselle.signer/policy-impl-pades "4.3.0"]
```

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

https://repo1.maven.org/maven2/org/demoiselle/signer/policy-impl-pades/

Capítulo 9. Funcionalidades

Este componente provê mecanismos de assinatura digital baseado nas normas ICP-Brasil e implementa mecanismos de assinatura digital para serem incorparadas em arquivo PDF

A interface `org.demoiselle.signer.policy.impl.pades.pkcs7.PCKS7Signer` provê as funcionalidades de Assinatura.

Para as funções de VALIDAÇÃO temos a interface `org.demoiselle.signer.policy.impl.pades.PCKS7Checker`

Este componente, até a presente versão, permite assinar dados representados por um array de bytes. Então se for necessário a assinatura de um arquivo PDF, por exemplo, a aplicação deverá montar um array de bytes com o conteúdo a ser assinado e enviar este para o componente poder assiná-lo. Também é possível enviar apenas o Hash já calculado deste conteúdo. A manipulação do arquivo PDF pode ser feito com ferramentas como o Apache PDFBox® [https://pdfbox.apache.org/]

Para assinar um dado através do componente policy-impl-pades é preciso executar alguns passos.

- Ter um conteúdo (ou hash calculado) a ser assinado
- Escolher qual versão da política ICP-BRASIL
- Caso for assinar apenas o Hash é preciso informar qual o algoritmo usado

Assinatura enviando contédo

O formato PAdES [https://pt.wikipedia.org/wiki/PAdES] define o formato para assinatura. Já a ICP-Brasil define um conjunto próprio de informações básicas para as assinaturas digitais chamado PBAD-PAdES. Esse tipo de Assinatura não é reconhecido automaticamente pelos leitores de PDF como o Adobe Reader [https://acrobat.adobe.com/br/pt/acrobat/pdf-reader.html]. No caso do Adobe há um plugin desenvolvido pelo ITI que permite que esse formato seja reconhecido, o plugin está disponível neste link (clique aqui) [https://www.iti.gov.br/aplicativos/111-aplicativos/4105-plugin-pades]

A seguir temos um fragmento de código que demonstra a utilização do componente

```
byte[] content = // implentar leitura do contéudo do arquivo em PDF
KeyStore ks = getKeyStoreToken();
String alias = getAlias(ks);
PAdESSigner signer = new PAdESSigner();
signer.setCertificates(ks.getCertificateChain(alias));
// para token
signer.setPrivateKey((PrivateKey) ks.getKey(alias, null));
byte [] assinatura =signer.doDetachedSign(content);
```

A seguir temos um fragmento de código que demonstra a utilização do componente 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 PAdES.

- `AD_RB_PADES_1_1` Refere-se à Assinatura Digital de Referência Básica versão 1.1;
- `AD_RT_PADES_1_1` Refere-se à Assinatura Digital de Referência Temporal (com carimbo de tempo) versão 1.1;

```
byte[] content = // implentar leitura do contéudo do arquivo em PDF
KeyStore ks = getKeyStoreToken();
String alias = getAlias(ks);
PAdESSigner signer = new PAdESSigner();
signer.setCertificates(ks.getCertificateChain(alias));
// para token
signer.setPrivateKey((PrivateKey) ks.getKey(alias, null));
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RT_PADES_1_1);
byte [] assinatura =signer.doDetachedSign(content);
```

Importante

Caso não seja especificada nenhuma política, o componente assumirá a política padrão AD_RB_PADES_1_1.

Validação de assinatura com contéudo

Como já foi comentado, o componente até a presente versão não lê o arquivo PDF para validar a Assinatura, por isso é preciso que a aplicação extraia o conteúdo e a Assinatura e envie os dois separados para o componente

```
byte[] content = /* implementar metodo de leitura do PDF para extrair o conteúdo a
byte[] signature = /* implementar metodo de leitura do PDF para extrair a assinatu
PAdESChecker checker = new PAdESChecker();
List<SignatureInformations> signaturesInfo = checker.checkDetachedSignature(conten
```

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 a
 private Date signDate;  // data do equipamento no momento da geração das assinatu
    private Timestamp timeStampSigner = null; // Carimbo de tempo da assinatura, q
    private SignaturePolicy signaturePolicy; // Politica ICP-BRASIL usada para ger
    private LinkedList<String> validatorErrors = new LinkedList<String>();  // Lis
```

Validação de assinatura 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 = /* implementar metodo de leitura do PDF para extrair o conteúdo a
```

```
byte[] signature = /* implementar metodo de leitura do PDF para extrair a assinatu
PAdESChecker checker = new PAdESChecker();
// 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(SignerAl
```

Tratando os resultados da valiação

Como é possível que um mesmo arquivo possa contar várias assinaturas (PAdES principalmente), só será gerada exceção quando a assinatura estiver comprometida Nos demais casos, o Demoiselle-Signer irá devolver o resultado numa lista de objetos SignatureInformations. Essa classe contém os seguintes atributos:

- chain;

  Lista X509Certificate com a cadeia completa do certificado do Assinante

- signDate

  A data do equipamento onde foi gerada a assinatura, e serve apenas como referência, não tem nenhuma validade legal

- timeStampSigner

  É o carimbo do Tempo (Timestamp) incluído na Assinatura, é a prova legal da data e hora que a Assinatura foi gerada.

- signaturePolicy;

  A política (SignaturePolicy) que foi usada para gerar a Assinatura

- notAfter;

  A data de validade do Certificado do Assinante

- validatorWarnins

  Lista de Avisos. A assinatura pode estar correta mas não foi possível verificar algum atributo exigido por uma política da ICP-Brasil, que serão listados aqui

- validatorErrors

  Lista de Erros. A assinatura pode estar correta mas não foi possível verificar alguma condição de validação exigida pela ICP-Brasil

- invalidSignature

  valor booleano, que indica que Assinatura não está válida

- icpBrasilcertificate

  BasicCertificate do Assinante

Cabe ao sistema com base nos avisos ou erros, aceitar ou não a Assinatura. Apesar de existirem as políticas, qualquer tipo de Assinatura gerada com um certificado ICP-Brasil tem validade legal. A seguir temos um fragmento de código que demonstra esta validação.

```
List<SignatureInformations> signaturesInfo = checker
    .checkDetachedSignature(fileToVerify, signatureFile);

  if (signaturesInfo != null) {
   System.out.println("A assinatura foi validada. e retornou resultados");
   for (SignatureInformations si : signaturesInfo) {
    System.out.println(si.getSignDate());
    if (si.getTimeStampSigner() != null) {
     System.out.println("Serial"
       + si.getTimeStampSigner().toString());
    }
    System.out.println("informações do assinante:");
    BasicCertificate certificate = si.getIcpBrasilcertificate();
    if (!certificate.isCACertificate()) {
     if (certificate.hasCertificatePF()) {
      System.out.println("CPF: "+certificate.getICPBRCertificatePF().getCPF());
      System.out.println("Titulo de Eleitor: "+certificate.getICPBRCertificatePF()
     }
     if (certificate.hasCertificatePJ()) {
      System.out.println("CNPJ: "+certificate.getICPBRCertificatePJ().getCNPJ());
     }
    }
    // Carimbo do tempo
    if(si.getTimeStampSigner()!= null) {

     System.out.println(si.getTimeStampSigner().toString());
    }
    // A assinatura pode estar correta mas não foi possível verificar algum atribu
    for (String valErr : si.getValidatorErrors()) {
     System.err.println("+++++++++++++++ ERROS ++++++++++++++++++");
     System.err.println(valErr);
    }
    //A assinatura pode estar correta mas não foi possível verificar alguma condiç
    for (String valWarn : si.getValidatorWarnins()) {
     System.err.println("+++++++++++++++ AVISOS ++++++++++++++++++");
     System.err.println(valWarn);
    }

   }
```

Nota

No repositório do componente no GitHub há códigos de testes unitários para os exemplos acima, Assinar [https://github.com/demoiselle/signer/blob/master/policy-impl-pades/src/test/java/org/demoiselle/signer/policy/impl/pades/pkcs7/impl/PDFSigner.java] e Validar [https://github.com/demoiselle/signer/blob/master/policy-impl-pades/src/test/java/org/demoiselle/signer/policy/impl/pades/pkcs7/impl/PDFVerify.java]

Parte IV. Implementação das Políticas XAdES

O componente *Policy-Impl-XAdES* foi desenvolvido para atender às necessidades de assinatura digital no âmbito da ICP-Brasil. Conforme as políticas para o padrão PBAD-XAdES [http://politicas.icpbrasil.gov.br/LPA_XAdES.xml]. Esse padrão é utilizado para Assinaturas Digitais em Arquivos no formtao XML, conforme define o DOC-ICP-15 [https://www.gov.br/iti/pt-br/assuntos/legislacao/instrucoes-normativas/IN032021_DOC_15.03_assinada.pdf] da ICP-Brasil.

A validação dessas assinaturas pode ser feita através do Site: https://verificador.iti.gov.br/ ou utilizando o Assinador SERPRO. [http://www.serpro.gov.br/assinador-digital/]

Índice

Capítulo 10. Configuração do Policy XAdES

Instalação do componente

Para utilizar o componente *policy-impl-xades* 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-xades</artifactId>
    <version>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:policy-impl-xades:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="policy-impl-xades" rev="4.3.0" />
```

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

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

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

```
<dependency org="org.demoiselle.signer" name="policy-impl-xades" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "policy-impl-xades" % "4.3.0"
```

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

```
<[org.demoiselle.signer/policy-impl-xades "4.3.0"]
```

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

https://repo1.maven.org/maven2/org/demoiselle/signer/policy-impl-xades/

Capítulo 11. Funcionalidades

Este componente provê mecanismos de assinatura digital baseado nas normas ICP-Brasil e implementa mecanismos de assinatura digital no formato XML

O padrão xades [https://www.w3.org/TR/XAdES/] define o formato para assinatura. E a ICP-Brasil define um conjunto próprio de informações básicas para as assinaturas digitais chamado PBAD [https://www.gov.br/iti/pt-br/assuntos/repositorio/artefatos-de-assinatura-digital]

A interface `org.demoiselle.signer.policy.impl.xades.xml.Signer` provê as funcionalidades de Assinatura.

Para as funções de VALIDAÇÃO temos a interface `org.demoiselle.signer.policy.impl.xades.xml.Checker`

Este componente, até a presente versão, permite assinar no formato Enveloped (A Assinatura passa ser parte do documento) e Detached Envloped (Assinatura Desanexada).

Assinatura Enveloped

Para Assinar em XML Enveloped, a entrada deve ser um arquivo em formato XML.

- *Como entrada pode ser informado*

  Local do arquivo no sistema de arquivos
  String que representa o arquivo XML
  Um objeto da classe org.w3c.dom.Document
  InputStream
  ByteArray

A seguir temos um fragmento de código que demonstra a utilização do componente nesta funcionalidade

```
      Keystore ks = getKeyStoreTokenBySigner();
    File newFile = new File("caminho para o arquivo");
String alias = getAlias(ks);
XMLSigner xmlSigner = new XMLSigner();

// para A3
xmlSigner.setPrivateKey((PrivateKey) ks.getKey(alias, null));

// para A1
// quando certificado em arquivo, precisa informar a senha
// char[] senha = "senha".toCharArray();
// xmlSigner.setPrivateKey((PrivateKey) ks.getKey(alias, senha));

xmlSigner.setCertificateChain(ks.getCertificateChain(alias));
// o parâmentro true é para diferenciar do método que recebe uma String contend
Document doc = xmlSigner.signEnveloped(true, newFile.getPath());
```

A seguir temos um fragmento de código que demonstra a utilização do componente 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 XAdES.

- AD_RB_XADES_2_4 Refere-se à Assinatura Digital de Referência Básica versão 2.4;
- AD_RT_XADES_2_4 Refere-se à Assinatura Digital de Referência Temporal (com carimbo de tempo) versão 2.4;

```
      Keystore ks = getKeyStoreTokenBySigner();
    File newFile = new File("caminho para o arquivo");
String alias = getAlias(ks);
XMLSigner xmlSigner = new XMLSigner();

// para A3
xmlSigner.setPrivateKey((PrivateKey) ks.getKey(alias, null));

// para A1
// quando certificado em arquivo, precisa informar a senha
// char[] senha = "senha".toCharArray();
// xmlSigner.setPrivateKey((PrivateKey) ks.getKey(alias, senha));

xmlSigner.setCertificateChain(ks.getCertificateChain(alias));
// para mudar a politica de Assinatura
xmlSigner.setPolicyId(XMLPoliciesOID.AD_RT_XADES_2_4.getOID());
// indicando o local do arquivo XML
Document doc = xmlSigner.signEnveloped(true, newFile.getPath());
```

Importante

Caso não seja especificada nenhuma política, o componente assumirá a política padrão AD_RB_XADES_2_4.

Geração de Assinatura XML do tipo Detached

Podemos gerar uma assinatura em formato XML para qualquer tipo de Arquivo. Da mesmo forma que o padrão CAdES gera um arquivo separado (.p7s), neste caso teremos um arquivo no formato xml desanexado do contéudo assinado.

Abaixo um trecho de código exemplo:

```
KeyStore ks =  getKeyStoreTokenBySigner();
File newFile = new File("caminho para o arquivo");
String alias = getAlias(ks);
XMLSigner xmlSigner = new XMLSigner();

// para A3
xmlSigner.setPrivateKey((PrivateKey) ks.getKey(alias, null));

// para A1
// quando certificado em arquivo, precisa informar a senha
// char[] senha = "senha".toCharArray();
// xmlSigner.setPrivateKey((PrivateKey) ks.getKey(alias, senha));

xmlSigner.setCertificateChain(ks.getCertificateChain(alias));
```

```
Document doc = xmlSigner.signDetachedEnveloped(newFile.getPath());
```

Validação de assinatura XML - Enveloped

A validação de uma Assinatura em XML consiste em enviar ao componente o arquivo XML que contém a Assinatura.

- *Como entrada pode ser informado*

  Local do arquivo no sistema de arquivos
  String que representa o arquivo XML
  Um objeto da classe org.w3c.dom.Document
  InputStream
  ByteArray

A seguir temos um fragmento de código que demonstra a utilização do componente nesta funcionalidade

```
File newFile = new File("caminho para o arquivo");
XMLChecker xadesChecker = new XMLChecker();
xadesChecker.check(true, newFile.getPath());
List<XMLSignatureInformations> results = new ArrayList<XMLSignatureInformations
results = xadesChecker.getSignaturesInfo();
```

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

```
public class SignatureInformations {

 private LinkedList<X509Certificate> chain; // cadeia do certificado que gerou a a
 private Date signDate;  // data do equipamento no momento da geração das assinatu
    private Timestamp timeStampSigner = null; // Carimbo de tempo da assinatura, q
    private XMLSignaturePolicy signaturePolicy; // Politica ICP-BRASIL usada para
    private Date notAfter; // data de vencimento do certificado que produziu a ass
 private LinkedList<String> validatorWarnins = new LinkedList<String>(); // Lista
 private LinkedList<String> validatorErrors = new LinkedList<String>();  // Lista
 private boolean invalidSignature = false; // Se a assinatura é válida
 private BasicCertificate icpBrasilcertificate = null; // Class que representa o c
```

Validação de assinatura para XML Detached

Na validação de uma assinatura XML Detached, temos além do arquivo .xml que contém a Assinautra temos outro arquivo com o conteúdo assinado. A seguir temos um fragmento de código que demonstra esta validação.

```
File newFile = new File("caminho até o contéudo assinado);
File newSignatureFile = new File("caminho até o XML com a Assinatura");
XMLChecker xadesChecker = new XMLChecker();
xadesChecker.check(newFile.getPath(), newSignatureFile.getPath());
```

Tratando os resultados da valiação

Como é possível que um mesmo arquivo possa contar várias assinaturas, só será gerada exceção quando a assinatura ou o arquivo estiverem comprometidos Nos demais casos, o Demoiselle-Signer irá devolver o resultado numa lista de objetos XMLSignatureInformations. Essa classe contém os seguintes atributos:

- chain;

  Lista X509Certificate com a cadeia completa do certificado do Assinante

- signDate

  A data do equipamento onde foi gerada a assinatura, e serve apenas como referência, não tem nenhuma validade legal

- timeStampSigner

  É o carimbo do Tempo (Timestamp) incluído na Assinatura, é a prova legal da data e hora que a Assinatura foi gerada.

- signaturePolicy;

  A política (SignaturePolicy) que foi usada para gerar a Assinatura

- notAfter;

  A data de validade do Certificado do Assinante

- validatorWarnins

  Lista de Avisos. A assinatura pode estar correta mas não foi possível verificar algum atributo exigido por uma política da ICP-Brasil, que serão listados aqui

- validatorErrors

  Lista de Erros. A assinatura pode estar correta mas não foi possível verificar alguma condição de validação exigida pela ICP-Brasil

- invalidSignature

  valor booleano, que indica que Assinatura não está válida

- icpBrasilcertificate

  BasicCertificate do Assinante

Cabe ao sistema com base nos avisos ou erros, aceitar ou não a Assinatura. Apesar de existirem as políticas, qualquer tipo de Assinatura gerada com um certificado ICP-Brasil tem validade legal. A seguir temos um fragmento de código que demonstra esta validação.

```
        List<XMLSignatureInformations> results = new ArrayList<XMLSignatureInforma
        results = xadesChecker.getSignaturesInfo();
if (!results.isEmpty()) {
  for (XMLSignatureInformations sis : results) {
   for (String valErr : sis.getValidatorErrors()) {
```

```
     System.err.println("+++++++++++++++ ERROS ++++++++++++++++++");
     System.err.println(valErr);
    }

    for (String valWarn : sis.getValidatorWarnins()) {
     System.err.println("+++++++++++++++ AVISOS ++++++++++++++++++");
     System.err.println(valWarn);
    }
    if (sis.getSignaturePolicy() != null) {
     System.out.println("------ Politica ---------------- ");
     System.out.println(sis.getSignaturePolicy().toString());
    }
    BasicCertificate bc = sis.getIcpBrasilcertificate();
    System.out.println(bc.toString());
    if (bc.hasCertificatePF()) {
     System.out.println(bc.getICPBRCertificatePF().getCPF());
    }
    if (bc.hasCertificatePJ()) {
     System.out.println(bc.getICPBRCertificatePJ().getCNPJ());
     System.out.println(bc.getICPBRCertificatePJ().getResponsibleCPF());
    }
    if (sis.getTimeStampSigner() != null) {
     System.out.println(sis.getTimeStampSigner().toString());
    }
   }
  }
```

Nota

No repositório do componente no GitHub há códigos de testes unitários para os exemplos acima, Assinar [https://github.com/demoiselle/signer/blob/master/policy-impl-xades/src/test/java/org/demoiselle/signer/policy/impl/xades/xml/XMLSignerTest.java] e Validar [https://github.com/demoiselle/signer/blob/master/policy-impl-xades/src/test/java/org/demoiselle/signer/policy/impl/xades/xml/XMLCheckerTest.java]

Parte V. 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.

Índice

Capítulo 12. Configuração do Policy Engine

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>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:policy-engine:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="policy-engine" rev="4.3.0" />
```

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

```
@Grapes(@Grab(group='org.demoiselle.signer', module='policy-engine', version='4.3
```

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

```
<dependency org="org.demoiselle.signer" name="policy-engine" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "policy-engine" % "4.3.0"
```

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

```
<[org.demoiselle.signer/policy-engine "4.3.0"]
```

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

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

Capítulo 13. Funcionalidades

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 Referência Básica:
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RB_CADES_2_2);

// Para usar a política com Referência de Tempo
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RT_CADES_2_2);
```

Parte VI. 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* quando a dependência é especificada no arquivo POM.XML

Índice

Capítulo 14. Configuração do Chain-ICP-Brasil

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>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:chain-icp-brasil:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="chain-icp-brasil" rev="4.3.0" />
```

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

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

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

```
<dependency org="org.demoiselle.signer" name="chain-icp-brasil" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "chain-icp-brasil" % "4.3.0"
```

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

```
<[org.demoiselle.signer/chain-icp-brasil "4.3.0"]
```

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

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

Autoridades Certificadoras

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

Tabela 14.1. Lista de Autoridades Certificadoras

Autoridade Certificadora		
AC A DIGIFORTE RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC ACD v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC Certisign ICP-Brasil Code, OU=Autoridade Certificadora Raiz Brasileira v11, O=ICP-Brasil, C=BR		
AC Certisign ICP-Brasil SSL, OU=Autoridade Certificadora Raiz Brasileira v10, O=ICP-Brasil, C=BR		
AC DIGITAL MAIS, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC INFOCO DIGITAL v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC PRIME v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC QUALITYCERT v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC REDE BRASIL v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC SEMPRE RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SERASA SSL EV, OU=Autoridade Certificadora Raiz Brasileira v10, O=ICP-Brasil, C=BR		
Autoridade Certificadora do SERPRO SSLv1, OU=Autoridade Certificadora		

Autoridade Certificadora		
Raiz Brasileira v10, O=ICP-Brasil, C=BR		
AC SOLUTI CS EV, OU=Autoridade Certificadora Raiz Brasileira v11, O=ICP-Brasil, C=BR		
AC SOLUTI SSL EV, OU=Autoridade Certificadora Raiz Brasileira v10, O=ICP-Brasil, C=BR		
AC VALID CODE SIGNING, OU=Autoridade Certificadora Raiz Brasileira v11, O=ICP-Brasil, C=BR		
AC VALID SSL EV, OU=Autoridade Certificadora Raiz Brasileira v10, O=ICP-Brasil, C=BR		
AC BOA VISTA RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC BOA VISTA, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC BOA VISTA CERTIFICADORA, OU=Autoridade Certificadora BOA VISTA - AC BOA VISTA, O=ICP-Brasil, C=BR		
AC BR RFB G4, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC CACB RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC CACB RFB G2, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC CAIXA PF 1v2, OU=Caixa Economica Federal, O=ICP-Brasil, C=BR		
AC CAIXA PJ 1v2, OU=Caixa Economica Federal, O=ICP-Brasil, C=BR		

Autoridade Certificadora		
AC CAIXA SPB, OU=Caixa Economica Federal, OU=CSPB-5, O=ICP-Brasil, C=BR		
AC CAIXA v2, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC CERTBANK, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC CERTDATA BRASIL, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC CERTFACIL v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC CERTIFICA ANAPOLIS v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC CERTIFICA MINAS v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC CERTISIGN-JUS CO-DESIGNING G6, OU=CODESIGNING, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC CERTISIGN-JUS G5, OU=SMIME, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC CERTISIGN-JUS SSL G6, OU=SSL, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC Certisign G6, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC Certisign G7, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC CERTISIGN-JUS G6, OU=SMIME, OU=Autoridade		

Autoridade Certificadora		
Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC Certisign Multipla CodeSigning, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Certisign Multipla G6, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Certisign Multipla G7, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Certisign Multipla SSL, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Certisign RFB G5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC Certisign SPB G6, OU=CSPB-2, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Certisign Tempo G1, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Certisign Tempo G2, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC CNDL RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC CNDL RFB v2, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC CNDL RFB v3, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC CONSULTI BRASIL RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
Autoridade Certificadora de Defesa, OU=Autoridade Certifica-		

Autoridade Certificadora		
dora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC DIGITAL, OU=AC SOLUTI, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC DIGITAL MULTIPLA G1, OU=AC DIGITAL MAIS, O=ICP-Brasil, C=BR		
AC DIGITAL v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC DIGITALSIGN, OU=DigitalSign Certificacao Digital Ltda, O=ICP-Brasil, C=BR		
AC DIGITALSIGN ACP, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC DIGITALSIGN ACP G2, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC DIGITALSIGN G2, OU=DigitalSign Certificacao Digital Ltda, O=ICP-Brasil, C=BR		
AC DIGITALSIGN RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC DIGITALSIGN RFB G2, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC DIGITALSIGN RFB G3, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC DIGITALSIGN SSL, OU=DigitalSign Certificacao Digital Ltda, O=ICP-Brasil, C=BR		
AC DOCCLOUD, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC DOCCLOUD RFB, OU=Secretaria da Receita Fede-		

Autoridade Certificadora		
ral do Brasil - RFB, O=ICP-Brasil, C=BR		
AC DOCCLOUD RFB v2, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC EGBA Multipla, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC EGBA Multipla G2, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC EGBA RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC FENACOR RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC Imprensa Oficial G4, OU=Imprensa Oficial do Estado S A IMESP, O=ICP-Brasil, C=BR		
AC Imprensa Oficial SP G3, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC Imprensa Oficial SP G4, ST=" ", OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC Imprensa Oficial SP RFB G4, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC Imprensa Oficial SP RFB G5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC Imprensa Oficial SSL, OU=Imprensa Oficial do Estado S A IMESP, O=ICP-Brasil, C=BR		
AC Instituto Fenacon G3, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		

Autoridade Certificadora		
AC Instituto Fenacon G4, OU=Certisign Certificadora Digital S.A., O=ICP-Brasil, C=BR		
AC Instituto Fenacon RFB G3, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC INTERCERT v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC LINK CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC LINK RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC LINK RFB v2, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC META CERTIFICADO DIGITAL CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC MULT v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC Notarial RFB G4, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC OAB G3, OU=ORDEM DOS ADVOGADOS DO BRASIL CONSELHO FEDERAL, O=ICP-Brasil, C=BR		
AC ONLINE BRASIL, OU=Autoridade Certificadora VALID - AC VALID, O=ICP-Brasil, C=BR		
AC ONLINE RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC ONLINE RFB v5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC ONLINE BRASIL v5, OU=Autoridade Certificadora		

Autoridade Certificadora		
VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC PETROBRAS G4, OU=PETROLEO BRASILEIRO S A PETROBRAS, O=ICP-Brasil, C=BR		
AC PLANO DIGITAL CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC PREMIUM CERTIFICADORA DIGITAL CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC Prodemge BR, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC Prodemge Codesigning, OU=AC Prodemge BR, O=ICP-Brasil, C=BR		
AC Prodemge Codesigning v2, OU=AC Prodemge BR, O=ICP-Brasil, C=BR		
AC PRODEMGE G4, OU=Companhia de Tecnologia da Informacao do Estado de MG - PRODEMGE, O=ICP-Brasil, C=BR		
AC Prodemge MG, OU=AC Prodemge BR, O=ICP-Brasil, C=BR		
AC Prodemge MG v2, OU=AC Prodemge BR, O=ICP-Brasil, C=BR		
AC Prodemge RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC PRODEMGE RFB G4, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC Prodemge SSL, OU=AC Prodemge BR, O=ICP-Brasil, C=BR		
AC Prodemge SSL v2, OU=AC Prodemge BR, O=ICP-Brasil, C=BR		

Autoridade Certificadora		
AC REDE IDEIA CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC REDE IDEIA RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
Autoridade Certificadora SAFE-ID BRASIL, OU=Servico Federal de Processamento de Dados - SERPRO, O=ICP-Brasil, C=BR		
AC SAFETECH CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC SAFEWEB CD, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC SAFEWEB RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SAFEWEB RFB v5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SAFEWEB TIMESTAMPING, OU=AC SAFEWEB, O=ICP-Brasil, C=BR		
AC SAFEWEB, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC Secretaria da Receita Federal do Brasil v3, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC Secretaria da Receita Federal do Brasil v4, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC SENHA DIGITAL BRASIL, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC SERASA-JUS v5, OU=SMIME, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		

Autoridade Certificadora		
AC SERASA RFB v5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SERPRO-JUS v5, OU=SMIME, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
Autoridade Certificadora SERPRORFBv4, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SIC BRASIL, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC SIC RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SINCOR G4, OU=SINCOR-SP - Sindicato dos Corretores de Seguros no Estado de SP, O=ICP-Brasil, C=BR		
AC SINCOR RFB G5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SINCOR RIO RFB G1, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SINCOR RIO RFB G2, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SOLUTI-JUS CO-DESIGNING v5, OU=CODESIGNING, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC SOLUTI-JUS SSL v5, OU=SSL, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC SOLUTI-JUS v5, OU=SMIME, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		

Autoridade Certificadora		
AC SOLUTI, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC SOLUTI-JUS v1, OU=Autoridade Certificadora da Justica - AC-JUS, O=ICP-Brasil, C=BR		
AC SOLUTI Multipla CO-DESIGNING, OU=AC SOLUTI, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC SOLUTI Multipla SSL, OU=AC SOLUTI, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC SOLUTI Multipla TI-MESTAMPING, OU=AC SOLUTI, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC SOLUTI Multipla, OU=AC SOLUTI, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC SOLUTI Multipla v5, OU=AC SOLUTI v5, O=ICP-Brasil, C=BR		
AC SOLUTI RFB, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SOLUTI RFB V5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
AC SOLUTI v5, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC VALID-JUS CO-DESIGNING v5, OU=CODESIGNING, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC VALID-JUS SSL v5, OU=SSL, OU=Autoridade Certi-		

Autoridade Certificadora		
ficadora da Justica v5, O=ICP-Brasil, C=BR		
AC VALID-JUS v5, OU=SMIME, OU=Autoridade Certificadora da Justica v5, O=ICP-Brasil, C=BR		
AC VALID, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
AC VALID BRASIL CO-DESIGNING, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID BRASIL SSL, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID BRASIL v5, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID-JUS v4, OU=Autoridade Certificadora da Justica - AC-JUS, O=ICP-Brasil, C=BR		
AC VALID PLUS, OU=Autoridade Certificadora VALID - AC VALID, O=ICP-Brasil, C=BR		
AC VALID PLUS CO-DESIGNING, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID PLUS SSL, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID PLUS TIMES-TAMPING, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID PLUS v5, OU=Autoridade Certificadora VALID - AC VALID v5, O=ICP-Brasil, C=BR		
AC VALID RFB v5, OU=Secretaria da Receita Fede-		

Autoridade Certificadora		
ral do Brasil - RFB, O=ICP-Brasil, C=BR		
AC VALID SPB v5, OU=CSPB-6, O=ICP-Brasil, C=BR		
AC VALID v5, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
AC Imprensa Oficial SP RFB SSL, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Casa da Moeda do Brasil v2, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Casa da Moeda do Brasil v3, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Justica v4, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Justica v5, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Presidencia da Republica v3, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Presidencia da Republica v4, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
Autoridade Certificadora da Presidencia da Republica v5, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
Autoridade Certificadora do SERPRO Final SSL, OU=Servico Federal de Processamento de Da-		

Autoridade Certificadora		
dos - SERPRO, OU=CSPB-1, O=ICP-Brasil, C=BR		
Autoridade Certificadora do SERPRO Final v4, OU=Servico Federal de Processamento de Dados - SERPRO, OU=CSPB-1, O=ICP-Brasil, C=BR		
Autoridade Certificadora do SERPRO Final v5, OU=Servico Federal de Processamento de Dados - SERPRO, O=ICP-Brasil, C=BR		
Autoridade Certificadora do SERPRORFB SSL, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
Autoridade Certificadora INFOCOMEX, OU=Servico Federal de Processamento de Dados - SERPRO, O=ICP-Brasil, C=BR		
Autoridade Certificadora PROCERTI, OU=Servico Federal de Processamento de Dados - SERPRO, O=ICP-Brasil, C=BR		
Autoridade Certificadora SEFAZCE, OU=Servico Federal de Processamento de Dados - SERPRO, O=ICP-Brasil, C=BR		
Autoridade Certificadora SERPRO v3, OU=Autoridade Certificadora Raiz Brasileira v2, O=ICP-Brasil, C=BR		
Autoridade Certificadora SERPRO v4, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
Autoridade Certificadora SERPRORFBv5, OU=Secretaria da Receita Federal do Brasil - RFB, O=ICP-Brasil, C=BR		
Autoridade Certificadora Raiz Brasileira v1, OU=Instituto Nacional de Tecnologia da Informacao - ITI, O=ICP-Brasil, C=BR		
Autoridade Certificadora Raiz Brasileira v10, OU=Instituto Na-		

Autoridade Certificadora		
cional de Tecnologia da Informacao - ITI, O=ICP-Brasil, C=BR		
Autoridade Certificadora Raiz Brasileira v11, OU=Instituto Nacional de Tecnologia da Informacao - ITI, O=ICP-Brasil, C=BR		
Autoridade Certificadora Raiz Brasileira v2, OU=Instituto Nacional de Tecnologia da Informacao - ITI, O=ICP-Brasil, C=BR		
Autoridade Certificadora Raiz Brasileira v5, OU=Instituto Nacional de Tecnologia da Informacao - ITI, O=ICP-Brasil, C=BR		
Instituto Nacional de Metrologia Qualidade e Tecnologia INMETRO, OU=Autoridade Certificadora Raiz Brasileira v6, O=ICP-Brasil, C=BR		
SERASA Autoridade Certificadora v5, OU=CSPB-4, O=ICP-Brasil, C=BR		
SERASA Autoridade Certificadora Principal v5, OU=Autoridade Certificadora Raiz Brasileira v5, O=ICP-Brasil, C=BR		
SERASA CD SSL V5, O=ICP-Brasil, C=BR		
SERASA Certificadora Digital v5, O=ICP-Brasil, C=BR		

Parte VII. 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

Índice

Capítulo 15. Configuração do TimeStamp

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>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:timestamp:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="timestamp" rev="4.3.0" />
```

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

```
@Grapes(@Grab(group='org.demoiselle.signer', module='timestamp', version='4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="timestamp" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "timestamp" % "4.3.0"
```

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

```
<[org.demoiselle.signer/timestamp "4.3.0"]
```

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

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

Capítulo 16. Funcionalidades

Carimbo de tempo com componente *policy-im-pl-cades*

O código abaixo demostra como pode ser feita a chamada:

```
PKCS7Signer signer = PKCS7Factory.getInstance().factoryDefault();

Certificate[] certificateToTimeStamp = // Certificado que irá requisitar o cari
signer.setCertificatesForTimeStamp(certificateToTimeStamp);
PrivateKey privateKeyToTimeStamp = // Chave privada que irá requisitar o carimb
signer.setPrivateKeyForTimeStamp(privateKeyToTimeStamp);
Certificate[] certificateToSign = // Certificado que irá gerar a Assinatura
PrivateKey privateKeyToSign = // Chave privada que irá gerar a Assinatura
signer.setCertificates(certificateToSign);
   signer.setPrivateKey(privateKeyToSign);
// usando a politica com carimbo de tempo
signer.setSignaturePolicy(PolicyFactory.Policies.AD_RT_CADES_2_2);
// Assinatura desatachada
byte[] signature = signer.doDetachedSign(fileToSign);
```

Carimbo de tempo com componente *policy-im-pl-pades*

O código abaixo demostra como pode ser feita a chamada neste caso, lembrando que o componente não anexa a assinatura no documento PDF:

```
         PAdESSigner signerPades = new PAdESSigner();
         Certificate[] certificateToTimeStamp = // Certificado que irá requisit
signerPades.setCertificatesForTimeStamp(certificateToTimeStamp);
PrivateKey privateKeyToTimeStamp = // Chave privada que irá requisitar o carimb
signerPades.setPrivateKeyForTimeStamp(privateKeyToTimeStamp);
Certificate[] certificateToSign = // Certificado que irá gerar a Assinatura
PrivateKey privateKeyToSign = // Chave privada que irá gerar a Assinatura
signerPades.setCertificates(certificateToSign);
   signerPades.setPrivateKey(privateKeyToSign);
// usando a politica com carimbo de tempo
signerPades.setSignaturePolicy(PolicyFactory.Policies.AD_RT_PADES_1_1);
// Assinatura desatachada
byte[] signature = signerPades.doDetachedSign(fileToSign);
```

Requisições de carimbo de tempo

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:

```
byte[] signatureFile = // array de bytes do conteúdo a ser assinado e carimbado
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
varCAdESTimeStampSigner.setCertificates(CertificateChain);
varCAdESTimeStampSigner.setPrivateKey(PrivateKey);
byte[] signatureWithTimeStamp = varCAdESTimeStampSigner
  .doTimeStampForSignature(signatureFile);
```

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]

```
byte[] content = // array de bytes do conteúdo
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
varCAdESTimeStampSigner.setCertificates(CertificateChain);
varCAdESTimeStampSigner.setPrivateKey(PrivateKey);
byte[] timeStampForContent = varCAdESTimeStampSigner.doTimeStampForContent(cont
```

Para o resumo (hash) de um conteúdo

A outra funcionalidade disponível permite enviar o resumo já calculado.

```
byte[] hash = // array de bytes do hash
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
varCAdESTimeStampSigner.setCertificates(CertificateChain);
varCAdESTimeStampSigner.setPrivateKey(PrivateKey);
byte[] timeStampForContent = varCAdESTimeStampSigner.doTimeStampFromHashContent(
```

Validdação do Carimbo de tempo com componente *policy-impl-cades e policy-impl-pades*

Veja a sessão de Validação

Validações de carimbo de tempo

Para uma assinatura padrão CAdES

```
byte[] signatureFile = // array de bytes da Assinatura
PAdESTimeStampSigner varPAdESTimeStampSigner = new PAdESTimeStampSigner();
List<Timestamp> listTimeStamp = varPAdESTimeStampSigner.checkTimeStampOnSignatu
if (!listTimeStamp.isEmpty()){
 for (Timestamp ts : listTimeStamp){
  System.out.println(ts.toString());
 }
}
```

Para uma assinatura padrão PAdES

```
byte[] signatureFile = // array de bytes da Assinatura
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
List<Timestamp> listTimeStamp = varCAdESTimeStampSigner.checkTimeStampOnSignatu
if (!listTimeStamp.isEmpty()){
 for (Timestamp ts : listTimeStamp){
  System.out.println(ts.toString());
 }
}
```

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:

```
byte[] timeStampFile = // array de bytes da Assinatura
byte[] content = // array de bytes do conteúdo assinado
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
Timestamp varTimeStamp = varCAdESTimeStampSigner.checkTimeStampWithContent(time
```

Para o resumo (hash) de um conteúdo

```
byte[] timeStampFile = // array de bytes da Assinatura
byte[] hash = // array de bytes do hash do conteúdo
CAdESTimeStampSigner varCAdESTimeStampSigner = new CAdESTimeStampSigner();
Timestamp varTimeStamp = varCAdESTimeStampSigner.checkTimeStampWithHash(timeStam
```

Definir timeout e tentativas de conexão

```
TimeStampConfig tsConfig = TimeStampConfig.getInstance();
tsConfig.setTimeOut(3000); // valor em milisegundos
tsConfig.setConnectReplay(3);
```

Parte VIII. 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.

Índice

Capítulo 17. Configuração do Cryptography

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>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:cryptography:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="cryptography" rev="4.3.0" />
```

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

```
@Grapes(@Grab(group='org.demoiselle.signer', module='cryptography', version='4.3
```

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

```
<dependency org="org.demoiselle.signer" name="cryptography" rev="4.3.0" />
```

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

```
libraryDependencies += "org.demoiselle.signer" % "cryptography" % "4.3.0"
```

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

```
<[org.demoiselle.signer/cryptography "4.3.0"]
```

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

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

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 17.1. 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 17.2. 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 17.3. 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 17.4. Exemplo com Variável JVM

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

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

Capítulo 18. Funcionalidades

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().fact

                /* 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_
                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().fact

                /* 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_
                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().fact

                /* 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_
                System.out.println(new String(conteudo_descriptografado));
        }
```

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.

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(fil
			inputStreamPublicKey.read(encodedPublicKey);
			inputStreamPublicKey.close();
			X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(
			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(fi
			inputStreamPrivateKey.read(encodedPrivateKey);
			inputStreamPrivateKey.close();
			PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySp
			kf = KeyFactory.getInstance("RSA");
			PrivateKey privateKey = kf.generatePrivate(privateKeySpec)

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

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

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

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

```
}
```

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.

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.factoryKeyStoreL
			String alias = (String) keyStore.aliases().nextElement();
			PrivateKey privateKey = (PrivateKey) keyStore.getKey(alias

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

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

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

			/* descriptografando com a chave publica */
			crypto.setKey(publicKey);
			byte[] conteudoAberto = crypto.decipher(conteudoCriptograf
			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.

Geração de Hash

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);
      }
```

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}/rel
              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.

Parte IX. Cadeias de homologação do SERPRO

O chain-icp-brasil-homolog fornece uma implementação para validação do conjunto de Autoridades de homologação emitidas pelo SERPRO. O acionamento das funcionalidades é feito pelo componentes de geração e validação de assinaturas como o *policy-impl-cades*, *policy-impl-pades* e *policy-impl-xades* quando a dependência for incluída no arquivo POM.XML da aplicação. Conforme descrito abaixo:

Índice

Capítulo 19. Configuração do Chain-ICP-Brasil-Homolog

Instalação do componente

Para instalar o componente *Demoiselle CA ICP-Brasil-HOMOLOG* 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-homolog</artifactId>
    <version>4.3.0</version>
   </dependency>
```

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

```
'org.demoiselle.signer:chain-icp-brasil-homolog:jar:4.3.0'
```

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

```
<dependency org="org.demoiselle.signer" name="chain-icp-brasil-homolog" rev="4.3
```

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

```
@Grapes(@Grab(group='org.demoiselle.signer', module='chain-icp-brasil-homolog', v
```

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

```
<dependency org="org.demoiselle.signer" name="chain-icp-brasil-homolog" rev="4.3
```

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

```
libraryDependencies += "org.demoiselle.signer" % "chain-icp-brasil-homolog" % "4
```

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

```
<[org.demoiselle.signer/chain-icp-brasil-homolog "4.3.0"]
```

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

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

Parte X. Integração com Sistemas Web para geração de Assinaturas

No antecessor do Demoiselle-Signer, existia o componente chamado *demoiselle-certificate-desktop* que fornecia algumas interfaces para que o desenvolvedor implementasse os métodos para comunicação com os certificados digitais instalados na máquina do usuário usando a tecnologia chamada Java Web Start, que utiliza JNLP (Java Network Launch Protocol) mas que foi depreciado pela Oracle no Java SE 9 e removido no Java SE 11.

Para resolver esse problema e possibilitar a integração com sistemas web, o SERPRO criou e disponibilizou a solução chamada Assinador SERPRO (http://www.serpro.gov.br/assinador-digital/)