Seleção PBAD/LabSEC - Etapa final

Brendon Vicente Rocha Silva
Graduando em Ciências da Computação
Universidade Federal de Santa Catarina - UFSC
Florianópolis, SC, Brasil
<brendon.vicente@grad.ufsc.br>

Foi proposto, nessa etapa do processo de seleção, a implementação de uma aplicação capaz de realizar operações relacionadas à segurança da computação.

O projeto foi desenvolvido em linguagem de programação *Java*, utilizando o *Development Kit 18.0.1.1* e, em grande parte, fazendo-se uso da biblioteca de algoritmos criptográficos *Bouncy Castle*.

1 PRIMEIRA ETAPA - RESUMO CRIPTOGRÁFICO

Para resumir o texto escolhido, foi necessário o uso da classe *MessageDigest*, capaz de resumir criptograficamente *arrays* de *bytes*, gerando um novo conjunto de *bytes*, de tamanho fixo, de acordo com um algoritmo de *hash*, previamente especificado (no caso do desafio, *SHA-256*).

Para salvar o arquivo, em formato hexadecimal, o método *format*, da classe *String*, foi utilizado, juntamente com o método *write*, da classe *Files*.

2 SEGUNDA ETAPA - CHAVES ASSIMÉTRICAS

A geração de chaves assimétricas foi feita através da classe *KeyPairGenerator*, com o algoritmo *ECDSA*. As chaves foram geradas com uma fonte de aleatoriedade, *SecureRandom*.

A classe *JcaPEMWriter* foi utilizada para escrever, em disco, os objetos, em formato *PEM*. As chaves privadas foram salvas em padrão *PKCS #8*, enquanto as chaves públicas foram salvas em padrão *X.509*.

As chaves são lidas, do disco, através das classes *PEMParser* e *JcaPEMKeyConverter*.

3 TERCEIRA ETAPA - CERTIFICADOS DIGITAIS

A estrutura primordial dos certificados é gerada através da classe *V3TBSCertificateGenerator*, do pacote *Bouncy Castle*. Os valores inseridos nos

certificados seguem as instruções do desafio e as chaves utilizadas foram lidas do disco, como produto dos resultados da etapa anterior.

O valor da assinatura do certificado foi gerado por meio da classe *Signature* e é convertido em uma instância da classe *DERBitString*.

Por fim, o certificado é gerado com o auxílio de funções contidas na classe *CertificateFactory*.

Para escrever os certificados em disco, utilizou-se um processo análogo à armazenar as chaves: através da classe *JcaPEMWriter*, os certificados foram salvos, em arquivos *.pem*.

O processo de leitura de certificados, do disco, também é similar às técnicas utilizadas anteriormente: uma instância de *PEMParser* e uma instância de *JcaX509CertificateConverter* foram utilizadas.

4 QUARTA ETAPA - REPOSITÓRIO DE CHAVES

Dois repositórios de chaves, em formato *PKCS #12*, foram criados, nessa etapa. Para tal, utilizou-se da classe *KeyStore*.

As chaves privadas e os certificados criados foram salvos nesses repositórios, guardados por senha.

O procedimento para ler os arquivos, do repositório em disco, é simples e assemelha-se ao processo de escrita dos arquivos: a própria classe *KeyStore* implementa métodos para recuperar os certificados e chaves do disco.

5 QUINTA ETAPA - ASSINATURA DIGITAL

A assinatura digital, em formato *CMS*, com algoritmo de resumo criptográfico *SHA-256* e algoritmo de criptografia assimétrica *ECDSA*, foi criada com a ajuda dos métodos da classe *CMSSignedDataGenerator*.

Para assinar o documento foram necessários os seguintes passos: preparar os dados para serem assinados, através da classe *CMSProcessableByteArray*; estruturar as informações do assinante, a partir de seu certificado digital e sua chave privada; adicionar o certificado à assinatura, através da classe *JcaCertStore*.

Escrever a assinatura em disco foi um processo relativamente simples, visto que a classe *CMSSignedData* provê métodos para recuperar os dados em formato binário.

6 SEXTA ETAPA - VERIFICAR ASSINATURA DIGITAL

O processo de verificação da assinatura digital foi feito com auxílio das classes SignerInformation e *SignerInformationVerifier*, que encarregam-se de verificar a validade de assinaturas do tipo anexadas, de forma facilitada.