UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Tarefa Prática – OpenSSL + Exercícios de criptografia simétrica, hash, MAC, PBKDF e Criptografia Autenticada em Java

Você irá usar as bibliotecas criptográficas fornecidas pelo provedor Bouncy Castle (https://www.bouncycastle.org/latest_releases.html) e deve incluir os pacotes nas bibliotecas dos projetos. As bibliotecas estão no diretório chamado "bibliotecas" e estão compactadas no arquivo zip da tarefa. Usaremos o provedor BCFIPS que é a versão FIPS da Bouncy Castle e em alguns casos o provedor BC que é o provedor padrão da Bouncy Castle.

O capítulo 3 do livro Beginning Cryptography with Java (arquivo Beginning Cryptography with Java.chm) pode ser lido para entender alguns dos exemplos fornecidos. Este livro está no formato CHM que é o formato Microsoft Compiled HTML Help. Assim, no Windows o arquivo abre normalmente. No Linux e MacOs deve ser obtido um programa para leitura do livro: no Linux pode ser o xCHM, ChmSee ou outro equivalente; no MacOs tem o CHM Reader.

Nas primeiras questões, você irá usar a Kali-Linux e o openssl (www.openssl.org) para executar os comandos. Se você quiser, pode usar o openssl na sua própria máquina Linux.

Depois, na segunda parte da tarefa, estão as questões de implementação em Java.

1ª parte - OpenSSL usando a Kali Linux

https://wiki.openssl.org/index.php/Command_Line_Utilities

CIFRAR E DECIFRAR um arquivo:

1.) (ENTREGAR) Cifrar o arquivo t1.txt com o algoritmo AES no modo CTR. Ver modos de cifragem usando: openssl enc -help

Use o comando: **openssl enc -aes-128-ctr -in t1.txt -out t1.aes -p**

```
root@kali:~/Documents# openssl enc -aes-128-ctr -in t1.txt -out t1.aes -p
enter aes-128-ctr encryption password:
Verifying - enter aes-128-ctr encryption password:
salt=649B18BF3FC1E10E
key=CE6D77CC40FB0D75703B0923144DD977
iv =AD58198FC4EBF993197AD8F21328C1CE
```

a) Qual chave foi usada para cifrar o arquivo?
b) Qual IV foi usado?
c) Como foram gerados a chave e o IV?
d) Onde ficam guardados a chave e o IV?

2.) (ENTREGAR) Agora, decifre o arquivo t1.aes.
Use o comando: **openssl enc -aes-128-ctr -d -in t1.aes -p**
O argumento **-d** significa "decifrar".

```
root@kali:~/Documents# openssl enc -aes-128-ctr -d -in t1.aes -p
enter aes-128-ctr decryption password:
salt=B65964C9D1801540
key=A6BC171B5C2282DC37E78CAEC914A020
iv =286869C26FC9C28AE5EA0581684EB264
Arquivo de teste - Carla.
root@kali:~/Documents#
```

HASH e MAC:

3.) (ENTREGAR) Crie o arquivo t1.txt no *gedit* e escreva algum conteúdo dentro do arquivo. Para gerar o hash deste arquivo usando o algoritmo **sha256**, pode ser usado o seguinte comando:
openssl dgst -sha256 t1.txt

```
Exemplo:
openssl dgst -sha256 t1.txt
SHA256(t1.txt)=
581aaf6196b599ac52fa37cfc0ce1050c399c1f6e71c0fd58a4f594e3f9d0183
```

Execute os seguintes comandos e coloque a saída obtida (screenshot) de cada um deles:

a. Obtenha a saída do comando: openssl dgst -sha256 -c t1.txt

b. Modifique o conteúdo do arquivo t1.txt. Agora, recalcule o valor do hash com o mesmo comando do item a. O valor obtido é igual ao valor do item a ou é diferente? Explique.

c. Encontre um arquivo em alguma página web que tenha o valor do hash SHA-256 listado na página (Sugestão de página: http://httpd.apache.org/download.cgi#apache24). Baixe o arquivo e recalcule o valor do SHA-256 com o openssl para conferir se o valor calculado é igual ao listado na página web. Mostre a tela da execução desse comando e indique o site usado.

4.) (ENTREGAR) Para gerar o MAC do arquivo use o comando abaixo, mostrando a tela de execução do comando. Explique os parâmetros usados no comando (Use o help e digite: **openssl dgst -help**).

a) openssl dgst -sha256 -mac HMAC -macopt hexkey:aabbcc t1.txt

2ª parte – Criptografia em Java

1. (ENTREGAR) (Dupla) Abra o **projeto2CodigoLivro** e teste o seu funcionamento. Responda:

   1.1. Qual algoritmo é usado no código? Em qual modo?

   1.2. Explique o que faz o método generateKey da classe KeyGenerator https://docs.oracle.com/javase/7/docs/api/javax/crypto/KeyGenerator.html.

   1.3. Explique como são usados os métodos *init*, *update* e *doFinal* para cifrar e para decifrar os dados nesse código. Leia a documentação e entenda bem o funcionamento desses métodos.

2. (ENTREGAR) (Dupla) Nesse projeto você irá programar dois sistemas de decifragem, um usando o AES em **modo CBC** e outro usando o AES no **modo contador** (*counter mode* – **CTR**). Em ambos os casos um IV de 16 bytes é escolhido de forma aleatória e está colocado no início do texto cifrado (precede o texto cifrado). Para o modo CBC use o esquema de padding PKCS5. Para o modo CTR use NoPadding.

   Inicialmente iremos testar apenas a função de decifragem. Use o projeto3Aes para auxiliar a responder as questões. Nas questões seguintes você recebe uma chave AES e um texto cifrado (ambos codificados em hexa) e o seu objetivo é recuperar o texto plano/texto decifrado. A resposta de cada questão é o texto decifrado (frase em português).

a)

- Chave CBC: c38a0d7bdd11e031c24e4895913393f9
- Texto cifrado em modo CBC (IV+texto cifrado):

  b90d84b82b283d5f783b9721f5f8bd1fb170b4319815f1a4fdbaff6f052f6e58a06d0200f28b1d333d8e3
  f11fcafef750122226c1bcea8d69416f5a15e4901b3c2fb5c33507139fe88f18c72fb0c435d

  Resposta-texto decifrado:

b)

- Chave CTR: abd95641ecb005d475496cd0bda4555f
- Texto cifrado em modo CTR (IV+texto cifrado):

  7182eb9d1fd3d9ed3ae1594b3cd3b02bf4667cd27c5e0a01dc2e66f53480e5fa249269e1bd17e7e06682
  4dcab22be4ccff41480a139eae1d390e1dd78548d7bb82841d88ae50fd4ea52727

  Resposta-texto decifrado:

3. No código **testeModifica**, faça:

   3.1. Explique o funcionamento do exemplo;

   3.2. Explique o significado da seguinte linha de código: `cipherText[11] ^= '0' ^ '9';`

4. No código **testeHASH**, faça:

   4.1. Explique como o hash sem chave é usado nesse exemplo;

   4.2. O que significa o valor “false” na verificação do hash?

5. Sobre hash de senhas, leia o material disponível do site https://crackstation.net/hashing-security.htm e escreva um resumo sobre o que é "certo" e o que é "errado" ao criar um arquivo com hashes de senhas. Projete uma ideia de formato seguro de arquivo de senhas fundamentado/baseado nas boas práticas descritas no material. Você deve criar o seu formato e deve descrever razões das escolhas feitas para o seu formato.

6. (ENTREGAR) No código **testeModificaETrocaHash**, faça:

   6.1. Explique como o hash sem chave é usado nesse exemplo;

   6.2. Explique a etapa de troca do hash do código abaixo.

```
// etapa de troca do hash

byte[] originalHash = hash.digest(Utils.toByteArray(input));
byte[] tamperedHash = hash.digest(Utils.toByteArray(
           "Transferir 9000100 para Conta Corrente 1234-5678"));

for (int i = ctLength - hash.getDigestLength(),
     j = 0; i != ctLength;
     i++, j++)
{
    cipherText[i] ^= originalHash[j] ^ tamperedHash[j];
}
```

7. No código **testeHMAC**, faça:

   7.1. Explique como o hash com chave/MAC (HMAC) é usado nesse exemplo;

   7.2. Explique o que acontece no código abaixo:

```
hMac.init(hMacKey);
hMac.update(Utils.toByteArray(input));

ctLength   +=   cipher.doFinal(hMac.doFinal(),   0,   hMac.getMacLength(),
cipherText, ctLength);
```

   7.3. Explique o que acontece no código abaixo:

```
hMac.init(hMacKey);
hMac.update(plainText, 0, messageLength);

byte[] messageHash = new byte[hMac.getMacLength()];
System.arraycopy(plainText, messageLength, messageHash,
                 0, messageHash.length);

System.out.println("Texto decifrado: " +
                   Utils.toString(plainText, messageLength));
System.out.println("Verificado o Mac: " +
                 MessageDigest.isEqual(hMac.doFinal(), messageHash));
```

8. (ENTREGAR) No código **AES-CriptoAutenticada**, faça:

   Obs: para funcionar este código você deve descompactar o arquivo jce_policy-8 que está dentro do diretório deste projeto. LEIA o README para saber para onde você deve copiar os arquivos descompactados. Só fazendo isso você conseguirá criar o AES com tamanhos de chave maiores. Se você tiver o JAVA 7, você deve baixar o arquivo na Internet: procure por Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7.

   8.1. Explique o funcionamento do método gcmTestWithGCMBlockCipherBC;

   8.2. Como as tags (MACs) foram usadas para identificar a modificação no texto cifrado?

   8.3. A detecção da modificação foi feita de forma manual ou automática no código? Explique.

9. **Compare** os códigos do *testeHMAC* e AES-CriptoAutenticada: a) qual o modo e algoritmo de cifragem usado em cada código? b) qual a diferença no funcionamento destes dois códigos?

10. **Testar** e **explicar** o código no diretório testePBKDF2. Responda também: o que faz o método *generateDerivedKey*?

11. (ENTREGAR) O projeto testeKeyStorev2 tem exemplo de uso do KeyStore do Java. Explique como funciona o código exemplo. A figura 1 mostra exemplo de estrutura do keystore Java. O exemplo de código na classe KeyStrAdap.java demonstra o uso do keystore do tipo BCFKS (keystore da BouncyCastle padrão FIPS). Execute o código da classe KeyStrAdap.java.

    Texto da documentação: "The BCFKS key store is designed to be FIPS compliant. It is available in approved-mode operation and is also capable of storing some secret key types in addition to public/private keys and certificates. The BCFKS key store uses PBKDF2 with HMAC SHA512 for password to key conversion and AES CCM for encryption."

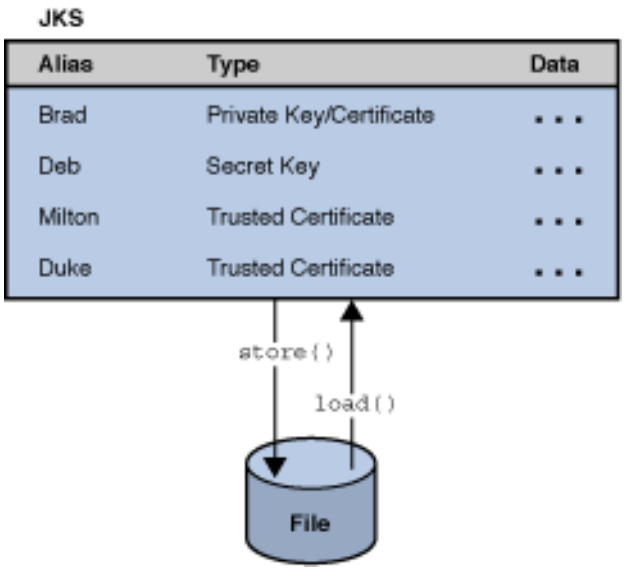

Figura 1 – Conteúdo de um Java KeyStore

Fonte: http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#KeyManagement

12. (ENTREGAR e APRESENTAR - 50 % da nota desta tarefa) Você irá implementar um sistema de comunicação entre funcionários de um ambiente usando a criptografia simétrica autenticada, derivação de chave e um keystore Java.

    Suponha um ambiente de uma empresa que tem os seguintes funcionários: Alice, Bob, Ana e Pedro.

    Alice define que quer conversar com Bob. Para conversar com Bob, Alice deve:

    a) Gerar, de forma segura e correta, uma chave criptográfica e um IV usando mecanismos de derivação de chaves (PBKDF2 ou Argon2). Chave e IV devem ser escritos na tela;

    b) Guardar a chave gerada e outros parâmetros necessários (talvez o IV) em um keystore Java. Esse keystore será o cofre de chaves único da empresa que guardará as chaves e/ou parâmetros criptográficos como o IV usados nas conversas entre os funcionários;

    c) Alice cifra a msg usando a chave e o IV e envia para Bob (via chamada de método). Alice e Bob podem ser objetos funcionários do sistema. Msg original e msg cifrada devem ser escritas na tela.

    Ao receber a msg, Bob deve:

    a) Obter a chave simétrica e/ou parâmetros como o IV usados para criptografar a msg que estão guardados no cofre da empresa (keystore Java). Bob deve escrever na tela a chave e IV obtidos do cofre;

    b) Escrever na tela a msg cifrada recebida;

    c) Decifrar a msg recebida e escrever a msg decifrada na tela.

NÃO É PERMITIDO: ter chaves e IV fixos e escritos no próprio código. Parâmetros devem ser guardados cifrados em arquivo (não podem ser guardados em texto plano). Não é permitido ter “passwords” no código-fonte.

O arquivo keystore Java é cifrado por padrão. A “senha” usada no keystore deve ser gerada com método de derivação.

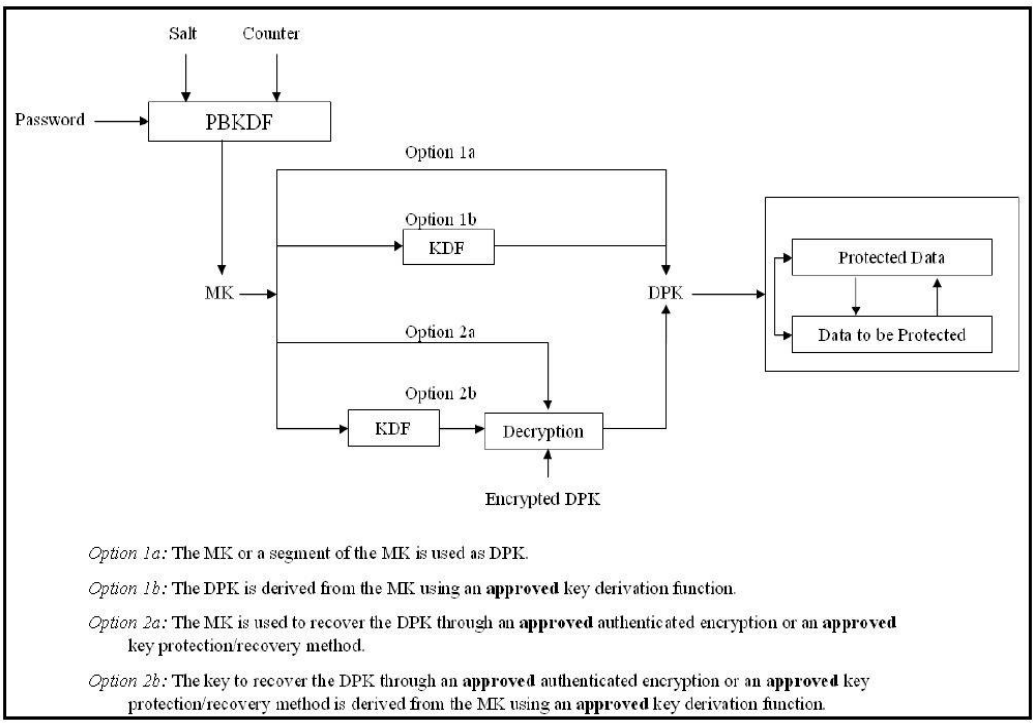

Figura 2 – SP 800-132 - Recommendation for Password-Based Key Derivation(Part 1: Storage Applications)

13. (Questão desafio, não é obrigatória a entrega) Um atacante intercepta o seguinte texto cifrado (codificado em hexa):

    20814804c1767293b99f1d9cab3bc3e7 ac1e37bfb15599e5f40eef805488281d

Ele sabe que o texto plano (mensagem original) é a seguinte mensagem ASCII: "Pay Bob 100$" (excluindo as aspas). Ele sabe que o texto cifrado usa a cifragem em modo CBC com um IV aleatório usando o AES como cifrador de bloco. Mostre que o atacante pode mudar o texto cifrado de forma que a decifragem resulte no seguinte texto: "Pay Bob 500$". **Responda: qual é o texto cifrado resultante (codificado em hexa)?** Isso mostra que o CBC não fornece integridade. Dica: Você pode ler o início do capítulo 3 do livro Beginning Cryptography with Java e ver os exemplos.

Referências:

1. Livro e códigos exemplo do livro Beginning Cryptography with Java: diretório Example (compactado junto no zip da tarefa): Disponível em http://www.wrox.com/WileyCDA/WroxTitle/Beginning-Cryptography-with-Java.productCd-0764596330.html
2. Bouncy Castle últimas versões: https://www.bouncycastle.org/latest_releases.html
3. Documentação classes BouncyCastle: https://www.bouncycastle.org/docs/docs1.5on/index.html
4. Lista das especificações dos algoritmos criptográficos da BouncyCastle: http://www.bouncycastle.org/specifications.html
5. Tutorial: http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html
6. Java Cryptography: http://www.cogentlogic.com/jc/JavaCryptographyPresentation.pdf
7. Documentação Java Cryptography Architecture Java 8 – https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html
5. Password Storage Cheat Sheet - https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md