Instituto Superior de Engenharia de Lisboa
LEIRT, LEIM, LEIC
Segurança Informática
Primeiro trabalho, Semestre de Inverno de 25/26
Entrega no Moodle até 24 de outubro de 2025

Entregar:

- *documento em PDF com identificação do grupo, respostas às perguntas, comandos OpenSSL, descrição dos ficheiros produzidos a incluir em anexo, e análises e experiências solicitadas;*
- *ficheiros em separado com código fonte nas questões que envolvem implementação (parte 2).*

Parte 1 (30 pontos)

Nos sistemas Linux/macOS, a ferramenta openssl está normalmente disponível na linha de comandos. Poderá verificar a sua disponibilidade e versão com o comando:

```
$ openssl -version
OpenSSL 3.5.2 5 Aug 2025 (Library: OpenSSL 3.5.2 5 Aug 2025)
```

Nos sistemas Windows recomenda-se utilizar a última versão binária disponível num dos sites referidos aqui `https://github.com/openssl/openssl/wiki/Binaries`. Os comandos do enunciado foram testados com a versão `https://download.firedaemon.com/FireDaemon-OpenSSL/openssl-3.5.3.zip`. Neste ZIP o ficheiro `openssl.exe` está na diretoria `openssl-3.5.3\x64\bin`.

```
C:\openssl\x64\bin>openssl -version
OpenSSL 3.5.3 16 Sep 2025 (Library: OpenSSL 3.5.3 16 Sep 2025)
```

1. Considere os seguintes comandos OpenSSL [1] e a utilização de um comando para concatenar os ficheiros `f` e `g` no ficheiro `result` (`cat` em Linux/Unix [2] ou o comando `gc` em Windows [3]).

   ```
   $ openssl enc -e -aes-128-cbc -in f -out g \
   -iv 0ed425637bb85abc9790850627b2355d -K 42c2771f807215802dff1329fa7bf2cf

   $ openssl mac -digest SHA256 -macopt key:changeit -in f -out h HMAC

   $ cat g h > result
   ```

   Para o caso de sistemas Windows via `Powershell` podem considerar, em alternativa ao `cat`, os comandos:

   ```
   $ gc g, h -Encoding byte | sc result -Encoding Byte
   ```

   1.1. Que tipo de proteções estão a ser aplicadas ao ficheiro? Descreva as operações criptográficas realizadas usando a notação apresentada nos slides.

   1.2. Considere os dois parâmetros de cifra/decifra, `-aes-128-cbc` e `-aes-128-ctr`. Indique os dois modos de operação usados e duas diferenças entre eles.

   1.3. Usando a notação dos slides, descreva as operações criptográficas correspondentes para desproteger/-verificar.

2. Considere duas partes num sistema de comunicação, *Alice* e *Bob*, bem como os algoritmos de cifra/decifra assimétrica (E_a e D_a) e função de hash ($SHA256$). Assuma, ainda, que K_eBob e K_dBob são as chaves pública e privada de *Bob*, respetivamente.

Envio. *Alice* envia a *Bob* a mensagem m cifrada e o *hash* da mensagem
$Alice \rightarrow Bob : ch = E_a(K_eBob)(m)||SHA256(m)$

Receção. *Bob* recebe o criptograma, decifra e verifica o *hash* da mensagem
$(c, h) = split(ch)$
$m = D_a(K_dBob)(c)$
$h \stackrel{?}{=} SHA256(m)$

2.1. Com base na documentação OpenSSL ([4, 5]) e nos exemplos presentes no repositório da disciplina [6], indique a sequência de comandos (*e.g.*, openssl, concatenação de ficheiros, ...) para implementar a comunicação.

2.2. Esta abordagem para proteção dos dados garante que os dados são transmitidos com confidencialidade e que *Bob* pode ter a certeza que os dados enviados por *Alice* não foram adulterados (integridade)? Se sim explique porquê, se não apresente uma possível solução.

3. Tenha em conta o site `https://www.example.org`:

3.1. Identifique a cadeia de certificados associada ao certificado do site.

3.2. Tendo por base os comandos OpenSSL, identifique e mostre qual o algoritmo usado para produzir a assinatura do certificado.

Parte 2 (70 pontos)

4. Considere uma nova função de *hash* $H_{16}(m)$ a qual corresponde à aplicação da função $SHA256(m)$ mas aproveitando apenas os primeiros 16 bits do *hash* calculado, ou seja, $H_{16}(m) = SHA256(m)_{0..15}$.

4.1. Sejam m_1 e m_2 respetivamente os códigos-fonte dos programas Java definidos nos ficheiros `BadApp.java` e `GoodApp.java` (presentes em anexo ao enunciado). Dois programas com códigos-fonte (possivelmente diferentes) m e m' dizem-se equivalentes ($m \equiv m'$) se a sua execução produz o mesmo resultado observável (*e.g.*, mesma saída).

Usando a biblioteca JCA realize uma aplicação para encontrar o código-fonte m' de um programa tal que $H_{16}(m') = H_{16}(m_2)$ e $m' \equiv m_1$.

4.2. Para os diferentes elementos do grupo indique o tempo médio (de 3 execuções) que o programa demora a encontrar m' e uma breve descrição do ambiente de teste (recursos de hardware, sistema operativo).

4.3. Se este ataque fosse possível sobre a totalidade dos 256 bits do *hash* SHA256, quais as consequências de usar a função SHA256 num esquema de assinatura digital?

5. Considere as seguintes questões sobre cifra autenticada e estabelecimento de chaves.

5.1. Usando a biblioteca JCA [7], implemente o esquema simétrico de cifra autenticada, cujo processo de proteção é descrito em seguida:

$$AE(k)(m) = E(k)(m)||T(k)(E(k)(m)) \quad (1)$$

onde m é a mensagem, E é uma função de cifra simétrica, k é a chave e T é uma função de MAC (*Message Authentication Code*).

- Para garantir confidencialidade use o algoritmo AES em modo *CBC* com padding *PKCS#5*, e o HMAC-SHA256 para autenticidade. A aplicação recebe na linha de comandos a opção para cifrar (`-cipher`) ou decifrar (`-decipher`), o ficheiro a proteger/desproteger (m), e um ficheiro com a chave (k). O ficheiro protegido deve ser codificado em base64.
- A aplicação produz o ficheiro protegido/desprotegido, e no caso da decifra deve também indicar se a mensagem é autêntica ou não. Na entrega inclua o PDF deste enunciado cifrado e a chave utilizada.
- Valorizam-se soluções que (i) funcionem para ficheiros de qualquer dimensão; (ii) tenham forma de geração aleatória da chave simétrica.

5.2. Na alínea anterior a chave simétrica era indicada num ficheiro o qual teria de ser enviado por um canal seguro. Descreva uma solução que não precise de um canal seguro para troca de chaves, propondo as modificações necessárias à definição (1).

6. Realize em Java uma implementação de um esquema de assinatura digital com a primitiva RSA. A aplicação recebe na linha de comandos a opção para assinar (`-sign`) ou verificar (`-verify`), a função de *hash* (`-sha1` ou `-sha256`) e o ficheiro para assinar/verificar.

No modo de assinatura recebe também:

i) *keystore* com a chave privada;

ii) `password` para aceder ao *keystore*;

e produz um ficheiro só com a assinatura.

No modo de verificação recebe também:

i) ficheiro com a assinatura;

ii) certificado de quem assinou;

e indica se a assinatura é válida ou não.

Valorizam-se soluções que:

i) possam assinar/verificar ficheiros de qualquer dimensão;

ii) validem o certificado e a sua cadeia antes de ser usado para validar a assinatura.

Para assinar e verificar use os certificados e chaves privadas disponíveis em anexo. Na entrega inclua a assinatura do PDF deste enunciado usando as chaves privadas de $Alice_1$ e Bob_2.

19 de setembro de 2025

Referências

[1] OpenSSL Docs, `https://docs.openssl.org/master/man1/` (18-set-2025)

[2] Comando cat, `https://man7.org/linux/man-pages/man1/cat.1.html` (18-set-2025)

[3] Comando gc, `https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-content` (18-set-2025)

[4] OpenSSL command rsa, `https://docs.openssl.org/3.3/man1/openssl-rsa/` (18-set-2025)

[5] OpenSSL command dgst, `https://docs.openssl.org/3.0/man1/openssl-dgst` (18-set-2025)

[6] SegInf Github, `https://github.com/isel-deetc-computersecurity/seginf-public/` (18-set-2025)

[7] Java Cryptography Architecture, `https://docs.oracle.com/en/java/javase/21/security/java-cryptography-architecture-jca-reference-guide.html` (18-set-2025)