Instituto Superior de Engenharia de Lisboa
LEIRT, LEIM, LEIC
**Segurança Informática**
Primeiro trabalho, Semestre de Inverno de 23/24
**Entrega no Moodle até 28 de outubro de 2023**

*Entregar:*

- *documento com identificação do grupo, respostas às perguntas da parte 1, questão 5 (comandos OpenSSL, imagens produzidas e análise solicitada na 5.3) e questões 6.3 e 6.4;*
- *ficheiros em separado com código fonte das questões 6 e 7.*

## Parte 1

1. Considere um novo esquema $CI$ para confidencialidade e autenticidade de mensagens, onde || representa a concatenação de bits e $X_{1..L}$ representa os primeiros $L$ bits de $X$.

$$CI(k_1, m) = E_s(T(k_1)(m)_{1..L})(m) \,||\, T(k_1)(m)$$

   $E_s(k)(m)$ é um esquema simétrico de cifra e $T(k)(m)$ é um esquema de *message authentication code* (MAC). Porque motivo este esquema não cumpre os objectivos?

2. Porque motivo é comum proteger e enviar chaves simétricas com esquemas de cifra assimétrica?

3. Indique duas semelhanças e duas diferenças entre um esquema de assinatura digital e um esquema MAC.

4. No contexto dos certificados X.509 e do perfil PKIX:

   4.1. Porque motivo o certificado $C$ pode ser considerado de confiança pelo sistema $S_a$ e não ser considerado de confiança pelo sistema $S_b$?

   4.2. Que mecanismo está previsto para tornar inválidos certificados assinados com chaves privadas de entidades folha?

## Parte 2

5. Pretende-se neste exercício verificar o impacto da utilização dos modos de operação ECB e CBC quando a mensagem em claro tem repetição de padrões.

   Considere a imagem BMP em anexo (`trab1.bmp`). Os ficheiros com este formato caracterizam-se por terem 2 zonas distintas. Os primeiros 54 bytes são uma zona de meta-informação, designada de cabeçalho, enquanto os restantes bytes (a partir da posição 55) representam os pixeis da imagem:

   5.1. Separe os bytes do cabeçalho e o corpo da imagem com os comandos `head` e `tail`. Por exemplo, os comandos seguintes colocam os primeiros 54 bytes do ficheiro *p1.bmp* no ficheiro *header*, e os restantes 55 bytes no ficheiros *body*:

   Sistemas Linux/macOS:

```
$ head -c 54 p1.bmp > header
$ tail -c +55 p1.bmp > body
```

   Sistemas Windows na *PowerShell*:

```
> gc p1.bmp -Encoding byte -Head 54 | sc header -Encoding byte
> gc p1.bmp -Encoding byte | Select-Object -Skip 54 | sc body -Encoding byte
```

   5.2. Usando a ferramenta de linha de comandos `openssl`, cifre o corpo do BMP com os algoritmos AES e DES e com os modos de operação ECB e CBC.

   Nota: Nos sistemas Linux/macOS a ferramenta openssl está disponível na linha de comandos. Nos sistemas Windows recomenda-se usar a última versão binária disponível aqui: `https://indy.fulgan.com/SSL/`

   5.3. Junte cada um dos resultados anteriores ao cabeçalho, formando 4 novas imagens. Verifique e discuta o resultado obtido em cada caso. Para juntar ficheiros pode usar o comando `cat`. O exemplo seguinte junta o conteúdo dos ficheiros *header* e *body* num novo ficheiro *new.bmp*.

   Sistemas Linux/macOS:

```
$ cat header body > new.bmp
```

Sistemas Windows na *PowerShell*:

```
> cat header > new.bmp
> cat body >> new.bmp
```

6. O objectivo desta questão é implementar uma cadeia de blocos (*blockchain*) simples com a propriedade de verificação de integridade dos seus blocos (validação da cadeia). Considere que um bloco da cadeia é composto por uma transação $t_i$ com os seguintes campos:

   - Origem: número inteiro positivo em decimal;
   - Destino: número inteiro positivo em decimal;
   - Valor da operação: número real (*float*).

   O encadeamento de blocos $b_i$ da *blockchain* em questão é definido por:

   $$b_i = t_i \mid\mid hash(b_{i-1}),$$

   para todo $1 \leq i \leq n$. Isto é, um bloco $b_i$ é o resultado da concatenação de uma transação $t_i$ e o *hash* do bloco anterior. Esta utilização do valor do hash na cadeia permite validar a integridade da cadeia de blocos.

   O bloco $b_0$, também chamado de bloco *genesis*, contém valores dos campos de transação nulos: origem com valor `-1`, destino com valor `-1`, valor de operação igual a `-1.0` e *hash* igual a 0x0. A cadeia de blocos deve ser armazenada e recuperada de um ficheiro CSV (*Comma-separate values*). Cada bloco corresponde a uma linha deste ficheiro com o seguinte formato:

   ```
   <origem>,<destino>,<valor>,<hash>
   ```

   O valor do *hash* de um bloco, calculado sobre a representação como *string* do bloco anterior, deve ser representado como uma sequência de algarismos em hexadecimal (*digest message*).

   6.1. Realize uma aplicação em Java e JCA que receba dados de uma transação na linha de comandos e esta transação na cadeia de um dado ficheiro. Considere o seguinte exemplo de utilização:

   ```
   $ addblock <origin> <destiny> <value> <filename>
   ```

   6.2. Realize uma aplicação em Java e JCA que obtenha de um ficheiro uma cadeia de blocos, valide a integridade da cadeia e apresente-a ao utilizador no caso da validação ser bem sucedida. Considere o seguinte exemplo de utilização:

   ```
   $ verifychain <filename>
   ```

   6.3. Suponha uma cadeia de 100 blocos conforme a descrição do enunciado. Suponha, ainda, que um atacante altera o valor da transação do bloco 10. Explique como esta alteração seria detetada pelo processo de validação.

   6.4. Considere agora a mesma cadeia de 100 blocos da alínea anterior. Porém, assuma que gostaríamos de realizar uma alteração legítima do valor da transação do bloco 10. Qual ou quais informações da cadeia precisariam ser alteradas?

7. Realize uma aplicação de consola para proteger texto (*strings*) usando a biblioteca JCA [1], representando o resultado no formato *JSON Web Encryption* (JWE) [2]. A proteção consiste no uso de cifra/decifra autenticada, com transporte de chaves usando criptografia assimétrica, tal como descrito na secção 3.3 do RFC 7516 [2]. Considere os seguintes exemplos de utilização para proteção e desproteção:

   ```
   $ jwe enc <some string> <recipient certificate>
   JWE token = ...

   $ jwe dec <jwe string> <recipient pfx>
   Decrypted text = ...
   ```

   Sugere-se a seguinte abordagem:

a. Realize e teste um componente para cifrar e decifrar *strings* usando o algoritmo AES em modo GCM [1]. Este componente deve receber a *string* a cifrar/decifrar, a *string* com dados adicionais, a chave simétrica, o IV e a dimensão da marca de autenticidade. Deve ser possível distinguir a parte do criptograma que corresponde à cifra da *string* e a parte que corresponde à marcada de autenticidade.

b. Realize e teste um componente para cifrar e decifrar uma chave simétrica AES usando o algoritmo RSA.

c. Realize e teste um componente para obter a chave pública de um certificado X.509 validado e uma chave privada de um *keystore*.

d. Use os componentes anteriores para realizar e testar o sistema pretendido.

Use o material criptográficos presente nos certificados e *keystores* do anexo `certificates-keys.zip`.

20 de setembro de 2023

## Referências


[1] `https://docs.oracle.com/en/java/javase/17/security/java-cryptography-architecture-jca-reference-guide.html`

[2] `https://datatracker.ietf.org/doc/html/rfc7516`