Instituto Superior de Engenharia de Lisboa
LEIRT, LEIM, LEIC
Segurança Informática
Primeiro trabalho, Semestre de Inverno de 24/25
Entrega no Moodle até 26 de outubro de 2024

Entregar:

- *documento com identificação do grupo, respostas às perguntas da parte 1, questão 5 (comandos OpenSSL, ficheiros produzidos e análise solicitada na 5.6) e experimentos solicitados questão 6;*
- *ficheiros em separado com código fonte das questões 6 e 7.*

Parte 1

1. Considere o esquema criptográfico CI definido a seguir, no qual || representa a concatenação de bits, $E_s(k)(m)$ é um esquema simétrico de cifra e $T(k)(m)$ é um esquema de *message authentication code* (MAC).

$$CI(k_1, k_2, m) = E_s(k_1)(m) \,||\, T(k_2)(k_1)$$

   Este esquema garante confidencialidade? E autenticidade? Explique.

2. Descreva um protocolo de estabelecimento seguro de chave simétrica entre Alice e Bob usando um esquema de cifra assimétrica. O seu protocolo pode assumir que Bob conhece a chave pública de Alice, mas não o contrário. O protocolo deve garantir a confidencialidade e a integridade da chave simétrica.

3. Suponha que Alice e Bob estabeleceram uma chave simétrica de forma segura e a tenham utilizado para uma comunicação em que as mensagens eram cifradas por uma cifra simétrica e protegidas por um MAC. Defina o conceito de não repúdio e explique se esta comunicação garante esta propriedade.

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

   4.1. É possível que um sistema S_a deixe de confiar num certificado C considerado hoje de confiança? Em quais situações?

   4.2. Considere que a notação $C_a \rightarrow C_b$ denota que o certificado da entidade A é assinado pela entidade B cujo certificado é C_b. Considere, ainda, a seguinte cadeia de certificados: $C_a \rightarrow C_b \rightarrow C_c \rightarrow C_d \rightarrow C_b$. Esta cadeia é válida? Explique.

Parte 2

5. Pretende-se neste exercício verificar o impacto da introdução de erros no criptograma quando utilizados os modos de operação ECB, CBC e CTR.

   Considere o ficheiro TXT em anexo (`mensagem.txt`). Trata-se de uma reprodução da RFC 4772, um documento com aproximadamente 68 kB. Neste exercício, iremos cifrar este ficheiro e, na sequência, alteraremos o valor do byte da posição 1000 do criptograma resultante. Por fim, verificaremos as consequências da introdução deste erro na mensagem decifrada. Realize os passos a seguir:

   5.1. Usando a ferramenta de linha de comandos `openssl`, cifre o ficheiro `mensagem.txt` com o algoritmo AES com chave de 128 bits utilizando os modos de operação ECB, CBC e CTR. Armazene os respetivos criptogramas em três ficheiros distintos.

   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/`

   5.2. Para cada ficheiro cifrado, comece por descobrir o valor atual do byte da posição 1000. Supondo que o ficheiro é chamado `cripto.dat`, o valor deste *byte* em hexadecimal pode ser obtido da seguinte forma:

   Sistemas Linux/macOS:

```
$ tail -c +1001 cripto.dat | head -c 1 | hexdump -C
```

   Sistemas Windows na *PowerShell*:

```
> gc cripto.dat -Encoding byte | Select-Object -Skip 1000 | Select-Object -Index 0
| sc one.dat -Encoding byte
```

5.3. Para cada criptograma, separe o ficheiro num prefixo (*bytes* antes da posição 1000) e um sufixo (*bytes* após a posição 1000).
Sistemas Linux/macOS:

```
$ head -c 1000 cripto.dat > prefixo.dat
$ tail -c +1002 cripto.dat > sufixo.dat
```

Sistemas Windows na *PowerShell*:

```
> gc cripto.dat -Encoding byte -Head 1000 | sc prefixo.dat -Encoding byte
> gc cripto.dat -Encoding byte | Select-Object -Skip 1001 | sc sufixo.dat -Encoding byte
```

5.4. Para cada criptograma, escolha um valor de byte diferente daquele na posição 1000 e crie um ficheiro contendo apenas um byte de valor diferente. Isto pode ser feito escolhendo-se um carácter ASCII e criando um ficheiro TXT chamado `novoByte.txt` com apenas este carácter. Lembre-se de não incluir espaços ou quebras de linha neste ficheiro.

5.5. Junte cada um dos resultados anteriores (prefixo, novo byte e sufixo), formando 3 novos criptogramas. Para juntar ficheiros, pode usar o comando `cat` (no Linux/macOS) ou os comandos `gc` e `sc` (na *PowerShell* do Windows). O exemplo seguinte junta o conteúdo dos ficheiros *prefixo.dat*, *novoByte.txt* e *sufixo.dat* num novo ficheiro *new.dat*.
Sistemas Linux/macOS:

```
$ cat prefixo.dat novoByte.txt sufixo.dat > new.dat
```

Sistemas Windows na *PowerShell*:

```
> gc prefixo.dat, novoByte.txt, sufixo.dat -Encoding byte | sc new.dat -Encoding Byte
```

5.6. Finalmente, utilize o `openssl` para decifrar cada um dos três criptogramas modificados. Compare o conteúdo da mensagem em texto plano original ao resultado das decifras. Sugestão: utilize um visualizador hexadecimal para isto (no Linux/macOS há o `hexdump`, na *PowerShell* no Windows há o comando `Format-Hex`). As mensagens decifradas contêm erros? Se sim, quantos bytes errados há em cada uma? Explique os resultados.

6. Usando a biblioteca JCA, realize em Java uma aplicação para geração de *hashs* criptográficos de ficheiros. A aplicação recebe na linha de comandos o nome do ficheiro para o qual se quer obter o *hash* e um ou mais nomes de algoritmos de *hash* suportados pela JCA. Os valores de *hash* são apresentados no *standard output*. Considere o seguinte exemplo de utilização:

```
$ filehash <filename> MD5 SHA-256
MD5 = ...
SHA-256 = ...
```

Teste a sua aplicação usando certificados (ficheiros `.cer`) presentes no arquivo `certificates-keys.zip`, em anexo a este enunciado, ou ficheiros cujo *hash* é conhecido (e.g. `https://httpd.apache.org/download.cgi`). Compare o resultado com os valores de *hash* apresentados pelo visualizador de certificados do sistema operativo (ou outro da sua confiança).

7. Usando a biblioteca JCA, realize em Java uma aplicação para cifrar ficheiros com um esquema híbrido, ou seja, usando cifra simétrica e assimétrica. O conteúdo do ficheiro é cifrado com uma chave simétrica, a qual é cifrada com a chave pública do destinatário do ficheiro. A aplicação recebe na linha de comandos a opção para cifrar (`-enc`) ou decifrar (`-dec`) e o ficheiro para cifrar/decifrar.

   No modo de cifra, a aplicação também recebe o certificado com a chave pública do destinatário e produz dois ficheiros, um com o conteúdo original cifrado e outro com a chave simétrica cifrada. Ambos os ficheiros devem ser codificados em Base64.

   No modo de decifra, a aplicação recebe também i) ficheiro com conteúdo original cifrado; ii) ficheiro com chave simétrica cifrada; iii) *keystore* com a chave privada do destinatário; e produz um novo ficheiro com o conteúdo original decifrado.

Valorizam-se soluções que: i) Validem o certificado antes de ser usada a chave pública e que não imponham limites à dimensão do ficheiro a cifrar/decifrar; e ii) Os algoritmos usados na cifra (da chave simétrica e da mensagem) sejam um parâmetro da aplicação.

Para a codificação e descodificação em *stream* de bytes em Base64 deve usar a biblioteca Apache Commons Codec [1].

Considere os ficheiros `.cer` e `.pfx` em anexo ao enunciado onde estão chaves públicas e privadas necessárias para testar a aplicação. A *password* dos *keystores* `.pfx` é *changeit*.

20 de setembro de 2024

Referências

[1] `https://commons.apache.org/proper/commons-codec/`