Instituto Superior de Engenharia de Lisboa

Engenharia Informática e de Computadores

SEGURANÇA INFORMÁTICA

1ºSÉRIE

Docente: José Simão

Filipe Fé nº 42141

Inês Gomes nº 42160

José Cunha nº 43526

Índice

Exercício 1 2

Exercício 2 2

Exercício 3

- **3.1)** 3
- **3.2)** 3

Exercício 4

- **4.1** 4
- **4.2** 4

Exercício 5

- **5.2.1** 4
- **5.2.2** 7
- **5.2.3** 7

Exercício 6 9

Exercício 7 9

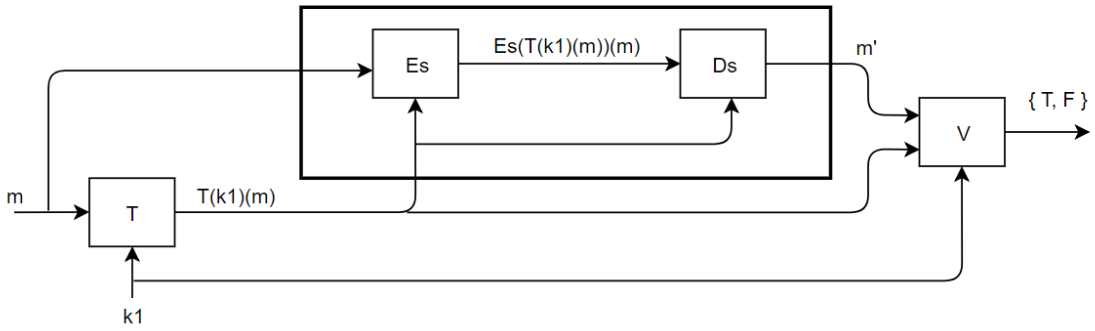
Exercício 1

Dado que num esquema de assinatura digital é realizado um ataque à função de hash SHA1, que permite, dado um x, obter x' diferente de x tal que ambos têm o mesmo hash, é quebrada uma das propriedades de segurança deste esquema: a integridade.

A probabilidade de existirem dois conjuntos de dados iguais com o mesmo hash é teoricamente bastante reduzida. Caso o atacante possua uma mensagem que produza um hash igual ao hash da original, ao invés de falhar na comparação destes (ou seja a mensagem do atacante não ser valida), é dito ao cliente que o conjunto de dados não sofreu alteração.

Exercício 2

No enunciado é referido que o esquema CI, $CI(m) = T(k1)(m) \,||\, Es(T(k1)(m)_{1::L})(m)$, é usado para cifra e autenticidade de mensagens.



Através do esquema descrito podemos constatar que a chave que é usada na cifra da mensagem do esquema simétrico (Es) é a marca de autenticidade gerada pelo esquema MAC com a chave k1, (T(k1) (m)).

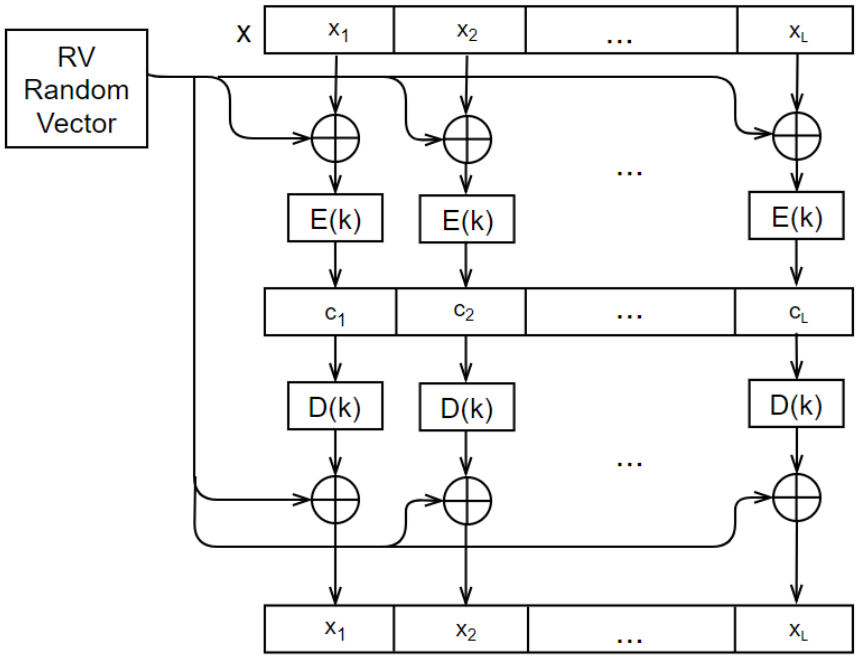
Apesar do esquema MAC garantir a autenticidade da mensagem, não garante a confidencialidade, ou seja, o canal de comunicação entre o gerador de marcas e a função de verificação destas não é seguro. Caso um atacante consiga descobrir a marca gerada, pode usá-la para decifrar a mensagem encriptada pelo esquema simétrico, falhando assim no cumprimento dos requisitos.

Exercício 3

3.1)

O algoritmo de decifra consiste passar à função de decifra o bloco cifrado e de seguida aplicar uma operação XOR com o valor aleatório RV de modo a obter o bloco de texto em claro.

$D(k)(c_i) \oplus RV$



3.2)

a)

O algoritmo de encriptação X do modo de operação definido no enunciado envolve uma operação XOR entre um vetor aleatório, RV, e um bloco de texto em claro, x_i. O resultado dessa operação é passado à primitiva, E. Posto isto, podemos afirmar que o padrão não envolve a encriptação de outros blocos, o que acontece no CBC, onde para cifrar cada bloco de mensagem é necessário realizar uma operação XOR com o bloco cifrado antecedente.

Por esta razão, seria mais fácil ao atacante descobrir o padrão do algoritmo X devido ao seu grau de simplicidade, contrariamente com o que se sucede no CBC.

b)

Como os blocos cifrados resultantes do algoritmo de encriptação X não são dependentes uns dos outros, é possível paralelizar todo o processo de cifra. No caso do modo de operação CBC, a paralelização é mais complicada pelo facto da encriptação de cada bloco depender da encriptação do anterior (à exceção do primeiro bloco, onde a operação lógica XOR é realizada com um valor inicial, IV).

Exercício 4

4.1

Não é possível uma das chaves privadas dos certificados intermédios ser usada na validação do certificado C. As chaves privadas servem para assinar os certificados folha, neste caso C, enquanto que as públicas é que são usadas para validar os certificados.

4.2

A Alice pode gerar novos certificados pois ao possuir a chave privada K_d, consegue assinar um outro certificado. No entanto, essa sua assinatura não é válida pois sendo ela um certificado folha, não esta identificada como uma CA, autoridade de certificação, não tem autoridade para gerar um certificado válido. Esse certificado não estará incluído na cadeia de certificação.

Exercício 5

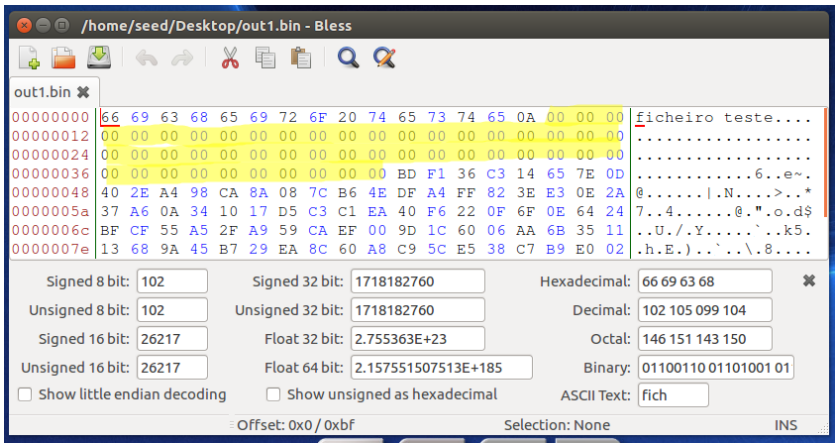
5.2.1
Questão 1

Após gerar os dois ficheiros de output através do comando md5collgen a partir de um ficheiro prefix.txt, ao fazer o comando diff é possível verificar que o seu conteúdo é idêntico, bem como os seus valores de hash, que tal foi possível verificar ao realizar o comando md5sum para cada ficheiro.

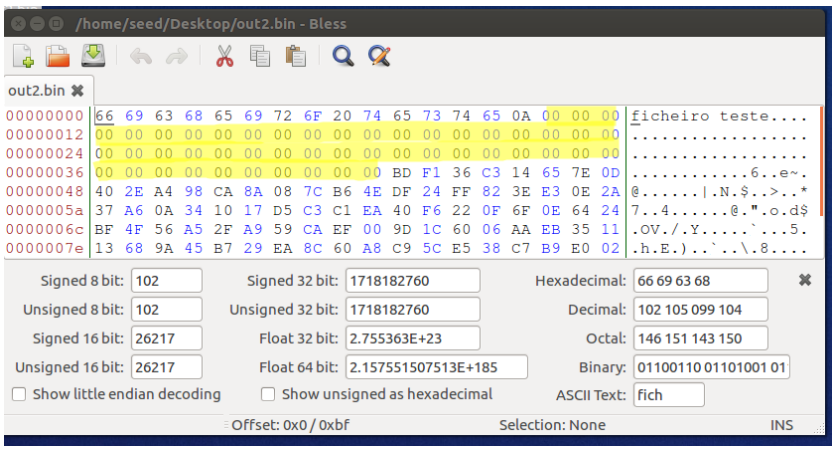
```
[10/03/18]seed@VM:~/Desktop$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 348ebac46732b4993501248967374a61

Generating first block: ................
Generating second block: S01.................
Running time: 40.8253 s
[10/03/18]seed@VM:~/Desktop$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[10/03/18]seed@VM:~/Desktop$ md5sum out1.bin
81bab1b4d4e6cbeea0800ed93d92dc33  out1.bin
[10/03/18]seed@VM:~/Desktop$ md5sum out2.bin
81bab1b4d4e6cbeea0800ed93d92dc33  out2.bin
```

O tamanho do ficheiro de prefixo criado não tem dimensão múltipla de 64, logo é previsto que seja adicionado um certo número de bytes que, em conjunto com os bytes de ficheiros, deem um número múltiplo de 64.

Ao analisar os ficheiros no *bless* é possível verificar que tal acontece.



ou1.bin



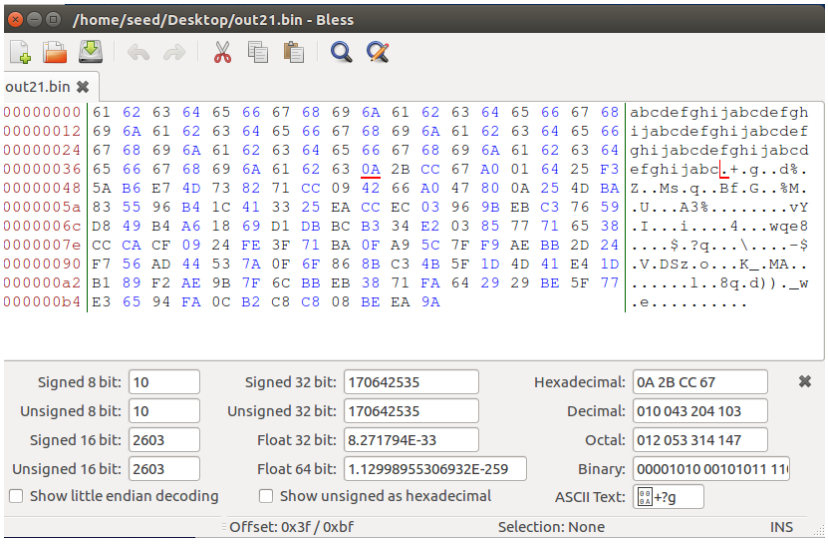
out2.bin

Questões 2 e 3

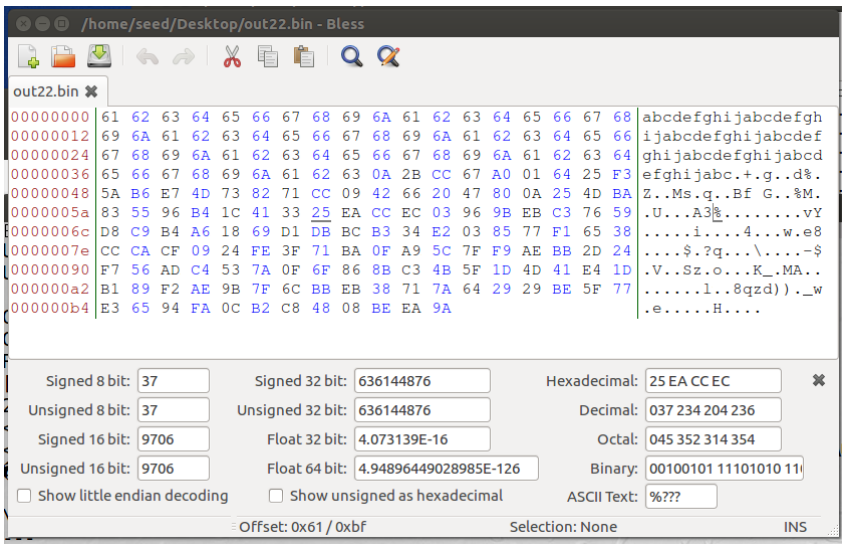
Com um ficheiro de prefixo de dimensão 64 bytes é possível verificar que não realizada padding e que o seu conteúdo binário não é completamente idêntico, nomeadamente nos últimos 128 bytes, mas que os valores de hash são iguais.

```
[10/03/18]seed@VM:~/Desktop$ md5collgen -p prefix2.txt -o out21.bin out22.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out21.bin' and 'out22.bin'
Using prefixfile: 'prefix2.txt'
Error: cannot open inputfile: 'prefix2.txt'
[10/03/18]seed@VM:~/Desktop$ md5collgen -p prefix2.txt -o out21.bin out22.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out21.bin' and 'out22.bin'
Using prefixfile: 'prefix2.txt'
Using initial value: d2d5ccd6677e120ba2a758be761cc278

Generating first block: ..
Generating second block: S11.........
Running time: 5.67447 s
```

```
[10/03/18]seed@VM:~/Desktop$ diff out21.bin out22.bin
2,3c2,3
< +@g@@d%@Z@@Ms@q@       Bf@G@
< %M@@U@@@A3%@@@@@@@@vY@I@@@@@@@@@@wqe8@@@      $@?q@@\@@@@-$@V@DSzo@@@K_@@MA@@@@
@@@@@@@8q@d))@_w@e@@
                  @@@
\ No newline at end of file
---
> +@g@@d%@Z@@Ms@q@       Bf G@
> %M@@U@@@A3%@@@@@@@@vY@n@@@@@@@@@@w@e8@@@      $@?q@@\@@@@-$@V@@Szo@@@K_@@MA@@@@
@@@@@@@8qzd))@_w@e@@
                  @@@
\ No newline at end of file
[10/03/18]seed@VM:~/Desktop$ md5sum out21.bin
93ec442aba65b9429ce04574cc7434d2  out21.bin
[10/03/18]seed@VM:~/Desktop$ md5sum out22.bin
93ec442aba65b9429ce04574cc7434d2  out22.bin
```



ou21.bin



out 22.bin

5.2.2

Neste exercício pretende-se provar a propriedade do algoritmo MD5 que dita que dados dois inputs M e N, se MD5(N) = MD5(M), por ex., os valores de hash dos dois ficheiros forem idênticos, então para qualquer input T, MD5(M ||T) = MD(N || T).

Na nossa prova, M e N correspondem aos ficheiros m1.out e m2.out obtidos através da utilização de md5collgen sobre um ficheiro m.txt, o que significa que têm o mesmo valor de hash.

```
[10/03/18]seed@VM:~/Desktop$ md5collgen -p m.txt -o m1.out m2.out
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'm1.out' and 'm2.out'
Using prefixfile: 'm.txt'
Using initial value: 67f00ae97b6f07ffaa4abed506874d96

Generating first block: ..............................
Generating second block: S10.....................................
.................................
Running time: 29.9613 s
```

T corresponde ao nosso ficheiro sufixo.txt.

```
[10/03/18]seed@VM:~/Desktop$ cat m1.out sufixo.txt > m1suf.txt
[10/03/18]seed@VM:~/Desktop$ cat m2.out sufixo.txt > m2suf.txt
```

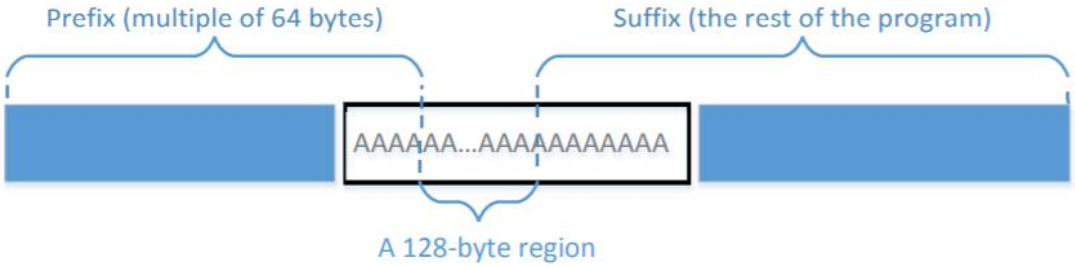
```
[10/03/18]seed@VM:~/Desktop$ md5sum m1suf.txt
6dda8032b926e4d5ed1c4a378ba15fad  m1suf.txt
[10/03/18]seed@VM:~/Desktop$ md5sum m2suf.txt
6dda8032b926e4d5ed1c4a378ba15fad  m2suf.txt
```

Como é possível verificar os ficheiros concatenados com o ficheiro de sufixo possuem o mesmo valor hash.

Nota: Os ficheiros utilizados e gerados encontram-se na diretoria Exc5 – 2.2.

5.2.3

O objetivo deste exercício foi criar duas versões diferentes do programa proposto, onde o conteúdo do seu array xyz é diferente, mas os valores hash dos executáveis são o mesmo.



O prefixo necessita de ser múltiplo de 64 para não adicionar padding. O primeiro endereço múltiplo de 64 correspondente ao conteúdo do array é 4224. Após a obtenção do prefixo a partir desse valor, é necessário correr o comando md5collgen para serem gerados dois ficheiros diferentes com o mesmo hash. Esses dois ficheiros correspondem ao prefixo mais a região de 128 bytes. Esses 128 são os bytes gerados pelo comando e correspondem ao diferente conteúdo do array.

```
[10/05/18]seed@VM:~/exc4$ gcc a.c
[10/05/18]seed@VM:~/exc4$ head -c 4224 a.out > prefixo.out
[10/05/18]seed@VM:~/exc4$ md5collgen prefixo.out -o p1.out p2.out
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'p1.out' and 'p2.out'
Using prefixfile: 'prefixo.out'
Using initial value: b31d1bf3bfb3399430240c08df34d362

Generating first block: ......
Generating second block: S00.....
Running time: 5.17809 s
```

Para a obtenção do sufixo somou-se aos 4224 do prefixo, os 128 bytes da região mais 1.

```
[10/05/18]seed@VM:~/exc4$ tail -c 4353 a.out > sufixo.out
```

Ao concatenar os ficheiros p1 e p2 com o sufixo e correr o comando md5sum é possível verificar que, apesar do conteúdo do array ser diferente, os valores de hash são iguais.

```
[10/05/18]seed@VM:~/exc4$ cat sufixo.out >> p1.out
[10/05/18]seed@VM:~/exc4$ cat sufixo.out >> p2.out
[10/05/18]seed@VM:~/exc4$
[10/05/18]seed@VM:~/exc4$ ./p1.out
bash: ./p1.out: Permission denied
[10/05/18]seed@VM:~/exc4$
[10/05/18]seed@VM:~/exc4$ chmod +x p1.out
[10/05/18]seed@VM:~/exc4$ chmod +x p2.out
[10/05/18]seed@VM:~/exc4$
[10/05/18]seed@VM:~/exc4$ ./p1.out
41414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141de7d2e0b03a59
51dd2423551078a924e44520d4c2c66692bf639d6d0711d4e3df8ecbd8adbb66becdb1a229da1aebb2df3563194cd531d0c683a7ead99952a7a8efd4f5767d4daeeab5afdc10f
2ba875ea163f4122c93bf8c4d95959b2478197da1b5a6e909b6dcc8f8bf27a88d647a787234094915bcdb5caedaac8500000000
[10/05/18]seed@VM:~/exc4$
[10/05/18]seed@VM:~/exc4$ ./p2.out
41414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141de7d2e0b03a59
51dd2423551078a924e4452054c2c66692bf639d6d0711d4e3df8ecbd8adbb66becdb1a2a9da1aebb2df3563194cd531d0c63a7ead99952a7a8efd4f5767d4daeeab5afdc10f2
ba8756a163f4122c93bf8c4d95959b2478197da1b5a6e909b6dcc8f8b727a88d647a787234094915bcdbdcaedaac8500000000
[10/05/18]seed@VM:~/exc4$
[10/05/18]seed@VM:~/exc4$ diff p1.out p2.out
Binary files p1.out and p2.out differ
[10/05/18]seed@VM:~/exc4$
[10/05/18]seed@VM:~/exc4$ md5sum p1.out p2.out
edbcccd078434acac72317e6fbb42bab  p1.out
edbcccd078434acac72317e6fbb42bab  p2.out
[10/05/18]seed@VM:~/exc4$
```

Nota: Os ficheiros utilizados e gerados encontram-se na diretoria Exc5 – 2.3.

Exercício 6

O processo para a implementação do esquema simétrico de cifra autenticada,

AE(k, m) = E(k)(m) || T(k)(E(k)(m))

envolveu trabalhar com objetos Cipher, Mac e SecretKey.

No processo de cifra foi, não apenas necessário encriptar o ficheiro dado, bem como gerar a marca de verificação e obter o vetor inicial, IV, do modo de operação, pois estes dois últimos dados são necessários para o processo de decifra, logo têm de estar contido no ficheiro encriptado.

No processo de decifra, foi necessário gerar a marca através do ficheiro encriptado (sem contar com os 20 bytes da tag do ficheiro cifrado e dos 8 bytes do IV) para se comparar o resultado com a tag presente no ficheiro recebido e validar a sua autenticidade. Caso o ficheiro passe na verificação, é decifrado.

Nota: Solução no ficheiro AuthCipherSym.java

Exercício 7

Visto que para realizar uma operação de assinature sobre um ficheiro é necessário ter posse da chave privada, foi necessário utilizar um objeto de tipo Signature que, em conjunto com a função de hash especificada e a essa chave privada, contida no keystore dado, serviu para alcançar esse fim.

No processo de verificação, de modo a obter a chave pública necessária do certificado dado, recorreu-se às classes CertificateFactory e X509Certificate. Para a realização da operação de verificação em si, recorreu-se de novo a Signature.

Nota: Solução no ficheiro AssinDigitalRSAg.java