# Relatório S1

## Docente: José Simão

- LI51_52D – LEIRT51D

- Grupo 6:

Ana Gaspar: 43877
André Martins: 43562
José Pedro Rodrigues: 43866

Síntese: Este primeiro trabalho prático teve como objetivo consolidar a matéria no domínio dos esquemas criptográficos, das primitivas de cifra de bloco e certificados digitais (X.509).

# Índice

# Índice de figuras

# 1) Introdução

Este primeiro trabalho prático aborda os esquemas criptográficos, as primitivas de cifra de bloco e certificados digitais (X.509) sendo, estes temas aplicados com a ajuda do *Java Cryptography Architecture*.

Assim, a fase de execução terá duas partes, sendo que a primeira é a realização das perguntas teóricas e a segunda é a realização das perguntas práticas que incidem na MD5 *Collision*, uma implementação de um esquema de cifra simétrica e autenticidade e, por fim, uma implementação do esquema de assinatura digital com a primitiva RSA.

# 2) Resolução

## Pergunta 1

Para um valor de x e x', o algoritmo obtém as mesmas funções de hash para ambos H(x) = H(x'), desse modo a função SHA1 não irá ser resistente à segunda pré-imagem, ou seja, não irá ser computacionalmente inexecutável achar a segunda pré-imagem (x') que tenha o mesmo *hash* que x.

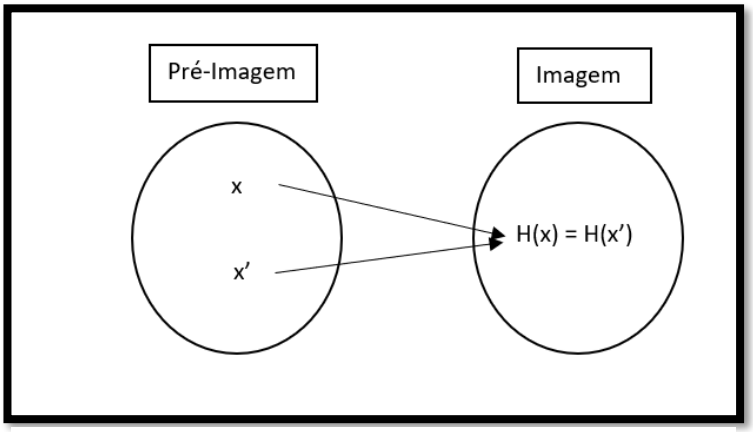


*Figura 1 - Esquema de uma função não resistente à 2ª Pré-Imagem*

Assim, no esquema da assinatura digital (ver figura 2) o atacante poderá gerar x' para falsificar a mensagem original, uma vez que ao fazer a verificação x e x' possuem o mesmo *hash*.

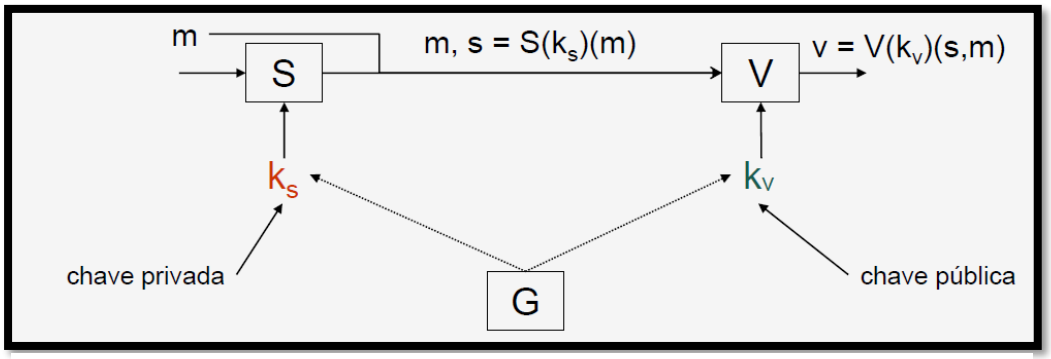


*Figura 2 - Esquema de assinatura digital*

# Pergunta 2

Este esquema, não cumpre os objetivos uma vez que está viola o objetivo de segurança pois a chave ($k_1$) é enviada juntamente com a mensagem. Desta forma, se o atacante conhecer a estrutura da mensagem enviada poderá conseguir decifrar a mesma. Para além disso, viola também o objetivo de autenticidade pois a *Tag* enviada é referente à mensagem original e não a que é enviada (mensagem cifrada), desta forma a validação posterior nunca irá ser feita com sucesso.

Falácia da autenticidade: $T(k_1)(m) \neq T(k_1)( E_s( T(k_1)(m)_{1..L})(m) )$

# Pergunta 3

## Pergunta 3.1

Algoritmo de decifra: $X_i = D(k)(Y_i \oplus RV)$

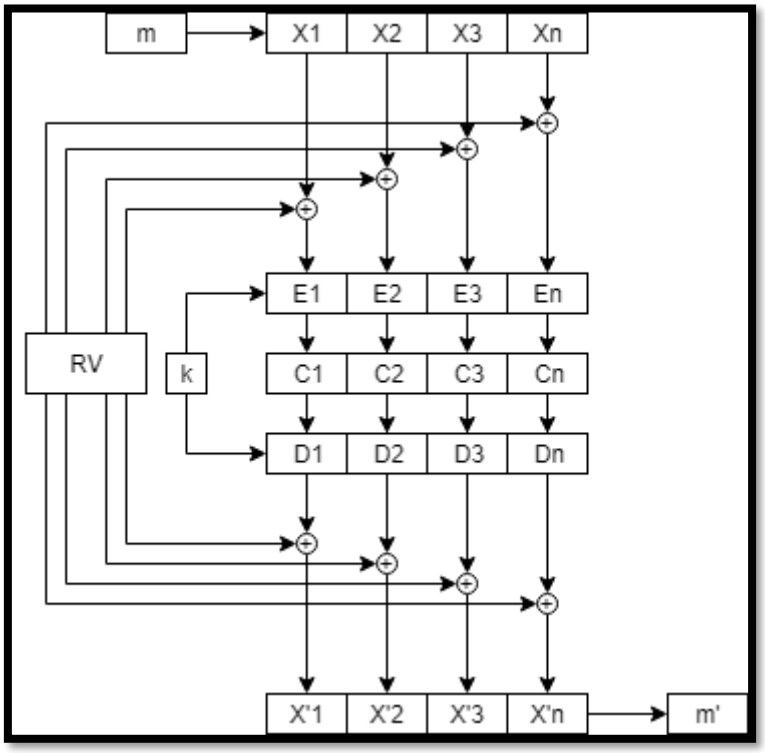


*Figura 3 - Esquema do algoritmo de cifra/decifra pelo novo modo de operação*

## Pergunta 3.2

**a) Padrões do texto em claro passarem para o texto cifrado**

No modo CBC se existirem duas mensagens iguais, os seus criptogramas podem não ser iguais uma vez que a cifra de um bloco depende do resultado da cifra do bloco anterior.

Por outro lado, no novo modo de operação como a cifra de um bloco não depende da cifra do bloco anterior, se duas mensagens forem iguais, os seus criptogramas irão ser iguais.

**b) Capacidade de paralelizar a cifra**

O modo de operação CBC faz XOR com um valor aleatório (IV) e encripta o primeiro bloco, para os restantes blocos irá fazer XOR com o resultado do anterior e depois encripta-os, ou seja, a cifra de um bloco de texto em claro afeta a cifra dos blocos seguintes, sendo que desta forma não é possível a paralelização.

Em relação ao novo modo de operação, a cifra de um bloco não depende da cifra do bloco anterior, logo, será possível realizar a paralelização.

# Pergunta 4

## Pergunta 4.1

Não, as chaves privadas dos certificados intermédios servem apenas para gerar novos certificados e não para fazer validações.

## Pergunta 4.2

Sim, a Alice poderá emitir novos certificados utilizando o certificado C e a sua respetiva chave privada (utilizada para autorização de geração de certificados), porém, os certificados gerados poderão não ter a devida credibilidade perante as empresas de nível superior, sendo comunidades não seguras.

# Pergunta 5

## Pergunta 5.1

a) Existe um preenchimento (*padding*) dos bits em falta (com zeros) para ser múltiplo de 64.
b) Quando o prefixo é múltiplo de 64 não é necessário, logo não existem as sequências de zeros que haviam no caso anterior.
c) Não, apenas algumas partes do ficheiro são diferentes.

```
[09/27/18]seed@VM:~$ hexdump out1.bin
0000000 6f43 6c6c 7369 6f69 206e 6574 7473 7020
0000010 7261 2061 756d 746c 7069 6f6c 6420 2065
0000020 3436 540a 7365 0a74 6554 7473 540a 7365
0000030 0a74 6554 7473 540a 7365 2074 3231 0a33
0000040 f3ef ddcb 32e2 e043 4771 b00c 9806 f202
0000050 f8d8 125b 3021 6296 5e11 6d37 a2bb c790
0000060 d994 05c1 4c6f 9915 e64f f13d 2412 074b
0000070 93d3 8654 816e 126e 8d9c 4058 9076 3243
0000080 6de8 70df 5ebd 63f0 caa1 48e2 7abd 5a13
0000090 e931 968f 9e68 e772 c529 ff00 5f8a d688
00000a0 15d7 3400 c99f fc5e 0839 7b08 4463 5b02
00000b0 fc9d d2d1 1916 ad02 ba7a 8e5a ec2f e083
00000c0
[09/27/18]seed@VM:~$ hexdump out2.bin
0000000 6f43 6c6c 7369 6f69 206e 6574 7473 7020
0000010 7261 2061 756d 746c 7069 6f6c 6420 2065
0000020 3436 540a 7365 0a74 6554 7473 540a 7365
0000030 0a74 6554 7473 540a 7365 2074 3231 0a33
0000040 f3ef ddcb 32e2 e043 4771 b00c 9806 f202
0000050 f8d8 925b 3021 6296 5e11 6d37 a2bb c790
0000060 d994 05c1 4c6f 9915 e64f f13d a412 074b
0000070 93d3 8654 816e 126e 8d9c c058 9076 3243
0000080 6de8 70df 5ebd 63f0 caa1 48e2 7abd 5a13
0000090 e931 168f 9e68 e772 c529 ff00 5f8a d688
00000a0 15d7 3400 c99f fc5e 0839 7b08 c463 5b01
00000b0 fc9d d2d1 1916 ad02 ba7a 0e5a ec2f e083
00000c0
```

Tamanho do prefix.txt: 64bytes

*Figura 4 – Compilação de bin1 e bin2 (múltiplo de 64)*

```
[09/27/18]seed@VM:~$ hexdump out2.bin
0000000 6f43 6c6c 7369 6f69 206e 6574 7473 000a
0000010 0000 0000 0000 0000 0000 0000 0000 0000
*
0000040 b664 d7ab 8c76 4520 a189 efle d43f 15ae
0000050 fad0 2a48 931f 2470 122c ca82 25d0 6f3b
0000060 c10f cde0 65fe 2dce 1bcb fa6a 0a22 8d5c
0000070 0ab9 a329 e593 495d c051 14ba aa3b ce5c
0000080 b9bb 1f44 c700 7da5 8f41 9c03 725b 62af
0000090 72f0 14cc 1932 dbab 790e 96ab 49d0 9f66
00000a0 9f91 b9ba 3c87 27cb 9212 71c5 4085 5ccd
00000b0 d4da 6317 d88f 5a03 f081 f9aa 964a ee0c
00000c0
[09/27/18]seed@VM:~$ hexdump out1.bin
0000000 6f43 6c6c 7369 6f69 206e 6574 7473 000a
0000010 0000 0000 0000 0000 0000 0000 0000 0000
*
0000040 b664 d7ab 8c76 4520 a189 efle d43f 15ae
0000050 fad0 aa48 931f 2470 122c ca82 25d0 6f3b
0000060 c10f cde0 65fe 2dce 1bcb fa6a 8a22 8d5b
0000070 0ab9 a329 e593 495d c051 94ba aa3b ce5c
0000080 b9bb 1f44 c700 7da5 8f41 9c03 725b 62af
0000090 72f0 94cc 1932 dbab 790e 96ab 49d0 9f66
00000a0 9f91 b9ba 3c87 27cb 9212 71c5 c085 5ccd
00000b0 d4da 6317 d88f 5a03 f081 79aa 964a ee0c
00000c0
```

Conteudo prefix.txt: "Collision test"
Tamanho: 15bytes

*Figura 5 - Compilação de bin1 e bin2 (não múltiplo de 64)*

## Pergunta 5.2

Para a resolução deste exercício primeiro criámos um ficheiro para servir de prefixo, de seguida executámos o comando *$md5collgen -p prefixo -o M N* de forma a gerar os dois ficheiros M e N com o mesmo *hash*, por fim concatenámos no final de cada um dos ficheiros um ficheiro T (aleatório) através dos comandos *$cat M T > MT* e *$cat N T > NT* e verificámos que os ficheiros finais MT e NT tinham também o mesmo *hash* com o comando *$md5sum file.* Desta forma, confirma-se a seguinte propriedade do MD5:

MD5 (M) = MD5(N) → MD5(M||T) = MD5(N||T)

```
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ md5collgen -p hash.txt -o M.txt N.txt
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'M.txt' and 'N.txt'
Using prefixfile: 'hash.txt'
Using initial value: 7bada30a76e59b3d11fabd793cc2c322

Generating first block: ........
Generating second block: W..
Running time: 20.9099 s
[09/29/18]seed@VM:~/.../Ex5 - 2_2$
```

*Figura 6 – Geração da colisão*

```
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ openssl md5 M.txt
MD5(M.txt)= d4ab622d55eed1491608f6bfe03e83f4
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ openssl md5 N.txt
MD5(N.txt)= d4ab622d55eed1491608f6bfe03e83f4
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ diff M.txt N.txt
Binary files M.txt and N.txt differ
```

*Figura 7 – Verificação da colisão*

```
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ cat N.txt T.txt > NT.txt
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ cat M.txt T.txt > MT.txt
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ openssl md5 NT.txt
MD5(NT.txt)= bde7461a362df5134995ecd4db0ad03f
[09/29/18]seed@VM:~/.../Ex5 - 2_2$ openssl md5 MT.txt
MD5(MT.txt)= bde7461a362df5134995ecd4db0ad03f
```

*Figura 8 - Concatenação e verificação da Hash final*

## Pergunta 5.3

Para a resolução deste exercício primeiro realizámos a compilação do código em C fornecido através de *$gcc src.c*. De seguida, com o auxílio de um programa para analisar o binário dos ficheiros verificámos onde começava o respectivo array no executável e selecionámos um ponto (no início do array, múltiplo de 64, evitando *padding*) para que esta parte do executável seja o nosso prefixo. Depois, gerámos dois ficheiros através do *md5collgen* com o prefixo antes obtido, ficando assim com 2 ficheiros com 128 bytes do array já gerados. Por fim, apenas concatenámos o fim do primeiro execútavel em ambos os ficheiros gerados (restantes 72 bytes do array e fim do executável), tendo ainda que modificar as permissões destes para que pudessemos executar, através do comando *$chmod*.

Confirmámos que os *hashes* dos dois executáveis gerados eram iguais (através do *md5sum*) e que o seu output era diferente (através da simples execução).

# Pergunta 6

Resolução em anexo.

# Pergunta 7

Resolução em anexo.

# 3) Conclusão

Este trabalho permitiu melhorar a interpretação da utilização dos esquemas criptográficos e dos certificados, bem como as suas respetivas importâncias.

Assim, foi possível consolidar a arquitetura interna dos esquemas criptográficos com as primitivas de cifra em bloco e, também, com os certificados. Aplicando as respetivas técnicas de forma consciente.

Por fim, com a realização deste trabalho foram consolidados conhecimentos sobre os temas abordados nas aulas.