Segurança Informática



# Índice

[Índice 2](#_Toc527927645)

[Questão 1 3](#_Toc527927646)

[Questão 2 3](#_Toc527927647)

[Questão 3 4](#_Toc527927648)

[3.1 4](#_Toc527927649)

[3.2 4](#_Toc527927650)

[Questão 4 5](#_Toc527927651)

[4.1 5](#_Toc527927652)

[4.2 5](#_Toc527927653)

[Questão 5 - MD5 Collision Attack Lab 6](#_Toc527927654)

[2.1 6](#_Toc527927655)

[2.2 8](#_Toc527927656)

[2.3 8](#_Toc527927657)

[Questão 6 10](#_Toc527927658)

[Questão 7 10](#_Toc527927659)

## Questão 1

Se o atacante conseguir que o resultado da passagem de duas mensagens distintas (x e x’) na função de hash seja igual, então este estaria na condição de conseguir forjar uma assinatura digital fazendo-se passar por outra pessoa.

## Questão 2

Se T é um esquema MAC logo:

* ***m*** passará por um canal inseguro e será possível interceptar este valor.

Ao aplicar E já sabemos qual é:

* a sua chave: ***T(k1)(m)***
* e a sua mensagem: ***m*** (devido a esta passar em T no seu canal inseguro)

Com estes dois valores conhecendo algoritmo **E** conseguimos decifrar o seu resultado.

Desta forma estando na posse de ***T(k1)(m)*** e ***Es(T(k1)(m)1::L)(m)*** e fazendo a sua concatenação bit a bit obtém-se o criptograma CI(m). Por este meio prova-se que CI não é fiável para ser um criptograma de autenticidade visto que é possível ser quebrado.

## Questão 3

### 3.1



Figura 1 – algoritmo de decifra

### 3.2

1. No modo CBC a existência de blocos de texto em claro afeta a cifra dos blocos seguintes porque o resultado da cifra de um bloco é usado para realizar um XOR com o bloco seguinte e esse resultado é o input do algoritmo de cifra. O mesmo já não acontece no algoritmo acima apresentado, porque mesmo que X0 tenha padrões de texto em claro o resultado da sua cifra não será utilizado para cifrar o bloco seguinte.
2. Neste modo de operação é possível realizar paralelização de trabalho na operação de cifragem visto que para cifrar o bloco seguinte não é necessário utilizar o resultado da cifra do bloco anterior. No extremo oposto a este caso está o modo de operação CBC que para cifrar um bloco *“i”* necessita conhecer o resultado da cifra do bloco *“i-1”* para realizar o *XOR* com o resultado do bloco Ep(k). Caso i=0 então o valor com o qual será realizado o *XOR* será o de um Vetor Inicial (IV).

## Questão 4

### 4.1

As chaves privadas não são utilizadas para validar certificados. Estas chaves apenas são utilizadas para calcular o campo assinatura do emissor de um certificado emitido.

As chaves publicas do emissor, ou a própria no caso de validação de um certificado raiz, sim, são utilizadas para validar um certificado.

Desta forma nenhuma chave privada de um certificado intermédio é utilizada para validar o certificado C, mas sim a chave publica do seu emissor.

### 4.2

O certificado C é uma end-entity, ou seja uma folha, e sendo uma folha a Alice pode emitir novos certificados usando o X.509 com a sua chave privada kd, no entanto não será feita a validação da sequencia. O novo certificado quando tenta validar não será possível, porque detecta que o certificado C de Alice é uma end-entity.

## Questão 5 - MD5 Collision Attack Lab

### 2.1

#### Questão 1 - If the length of your prefix file is not multiple of 64, what is going to happen?

Dimensão prefixo: 44 bytes (ficheiro em anexo “Q1prefix.txt”)

Dimensão de ficheiros de saída: 192 bytes (ficheiros em anexo “Q1out1.txt” e “Q1out2.txt”).

Hash utilizado obtido através do comando “*md5sum Q1out1.txt”*: 61459646d70cdcb913c9eb4b5a380215

Utilizando um ficheiro não múltiplo de 64 podemos observar, na Figura 2 e Figura 3, que o algoritmo através do mecanismo de *padding* adiciona zeros ao bloco a cifrar.

Na Figura 2 e Figura 3 podemos ver o resultado da comparação dos dois ficheiros resultantes da experiência. Como referido acima é possível visualizar os zeros bem como quais os bytes que são diferentes.

De salientar que os ficheiros de saída dão sempre maiores que o ficheiro de saída.

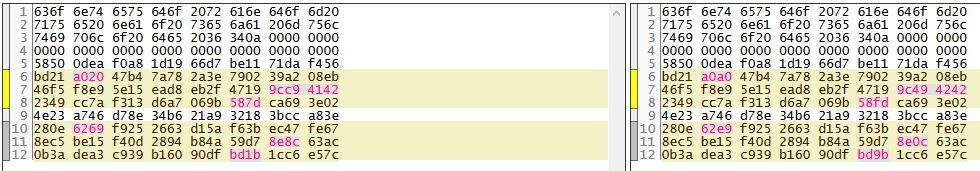


Figura 2 – ex 1 output 1

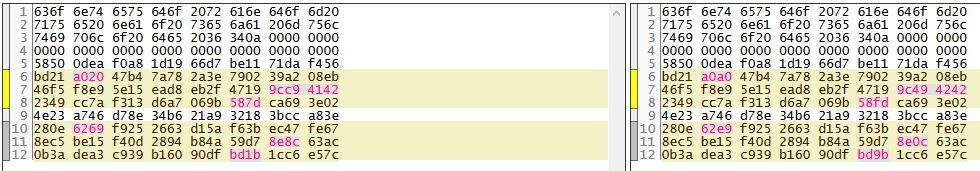


Figura 3 – ex 1 output 2

#### Questão 2 - Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what

#### happens.

Dimensão prefixo: 64 bytes (ficheiro em anexo “Q2prefix64.txt”)

Dimensão de ficheiros de saída: 192 bytes (ficheiros em anexo “Q2out1.txt” e “Q2out2.txt”).

Hash utilizado obtido através do comando “*md5sum out1.txt”*: 8e587cc6f8b6a188ed9535f11bbe829d

Utilizando desta vez um ficheiro diferente, mas com exactamente 64 bytes ao analisar a

Figura 4 e Figura 5 verificamos logo à partida que não existem manchas com bytes a zero, o que significa que não houve necessidade de adicionar *padding* ao output dos ficheiros.

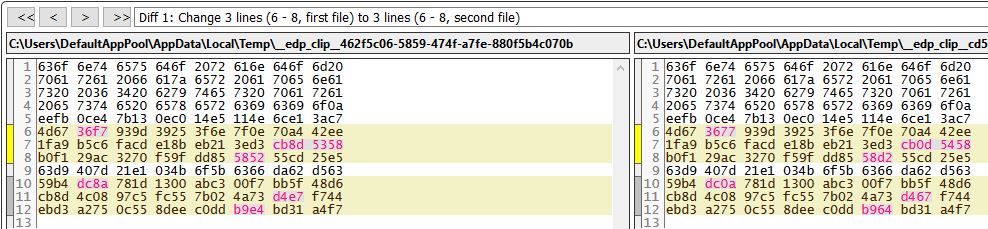


Figura 4 – ex 2 output 1

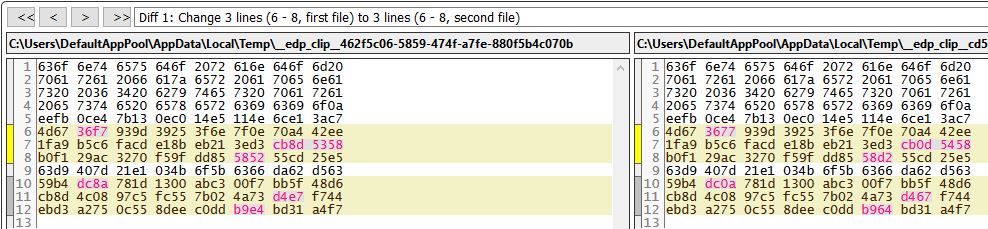


Figura 5 – ex 2 output 1

#### Questão 3 - Are the data (128 bytes) generated by md5collgen completely different for the two

#### output files? Please identify all the bytes that are different.

Analisando as figuras das questões 1 e 2 é percetível identificar as diferenças entre os ficheiros de saída do algoritmo. Podemos observar que só existem 7 diferenças nos ficheiros e estas são sempre no mesmo byte.

### 2.2

Neste exercicio analisamos as propriedades do algoritmo de MD5. Percebemos que o mesmo ficheiro pode ser gerado pelo md5collagen enquanto o processo de hash é feito, resultando em hash identicos. Isto acontece porque varias funçoes de hash como SHA-1 e MD5 estao sujeitas a “extensao de cumprimento”. O processo de hash a mensagem é feita atraves de blocos fixos em que é aplicado algoritmos que compactam a mensagem, observando que tem o hash MD5 identico.

### 2.3

Criamos dois ficheiros com o mesmo MD5 mas com diferentes sufixos, como se ve nas duas figuras, 'one' e 'second'. o output de cada um dos ficheiros é diferente.

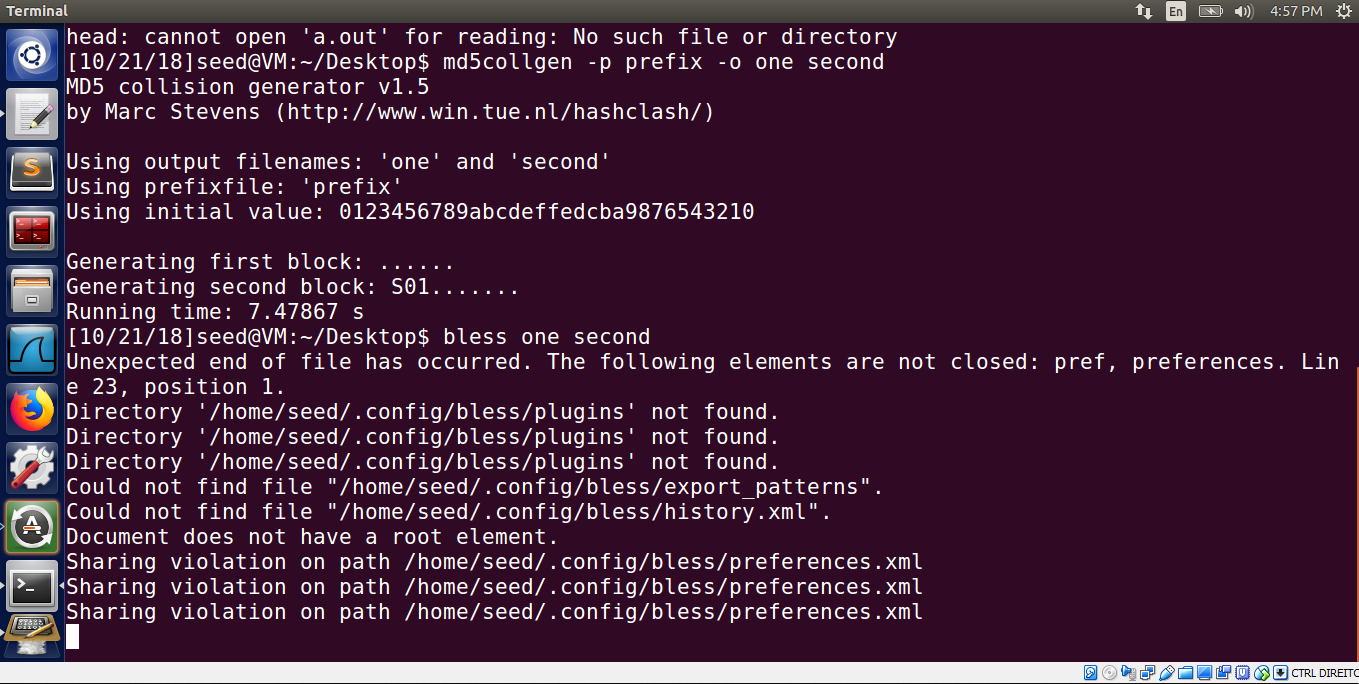


Figura 6 - Criação do prefixo

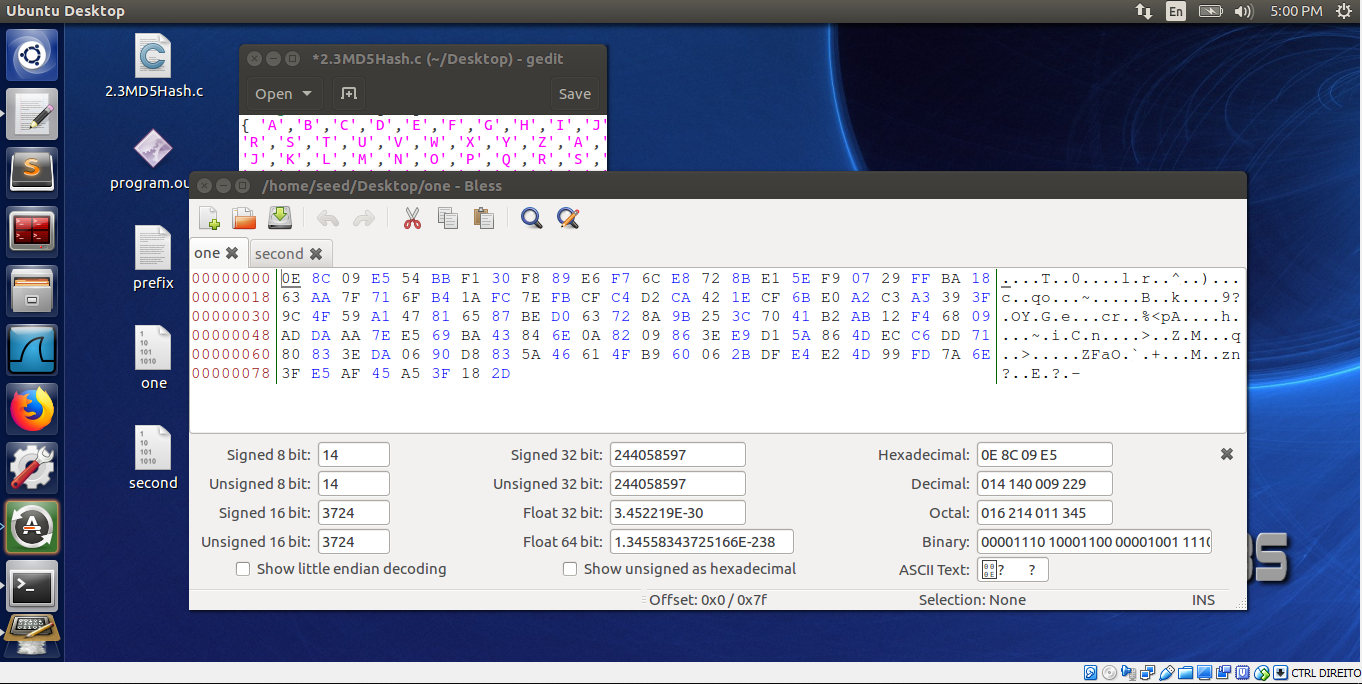


Figura 7 -ficheiro ‘One’

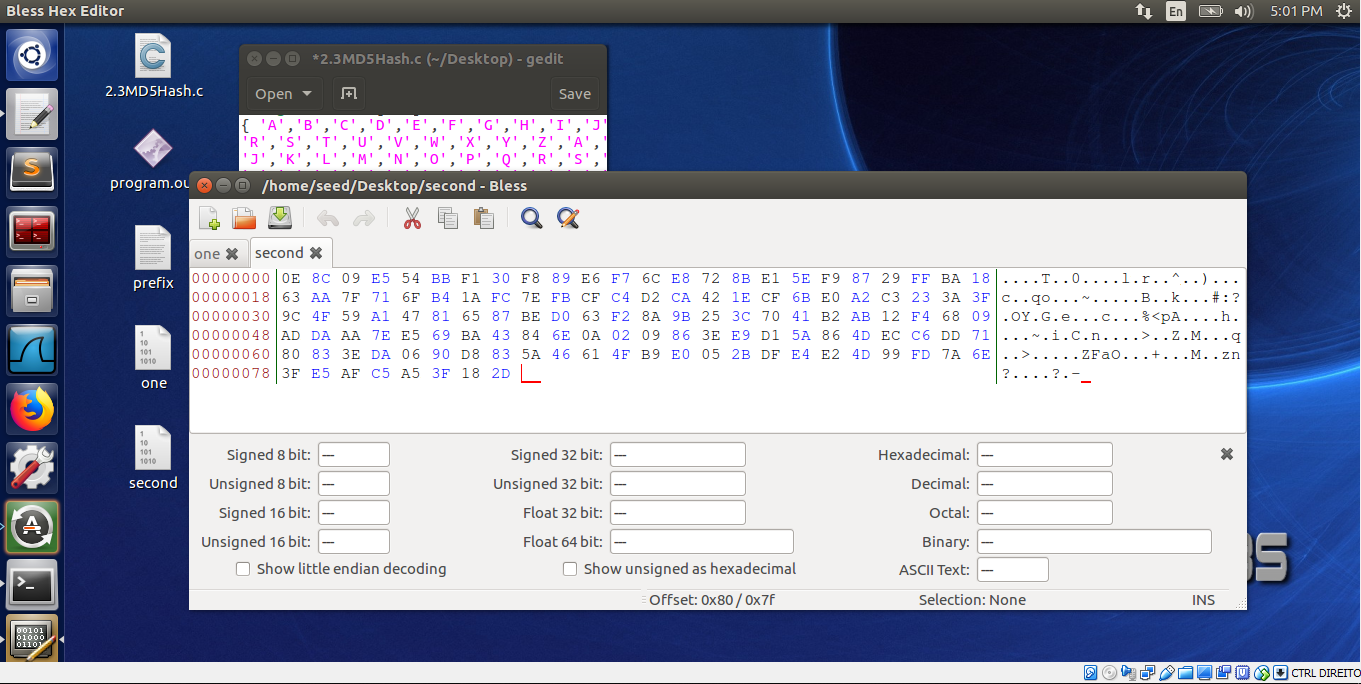


Figura 8 - ficheiro ‘second’

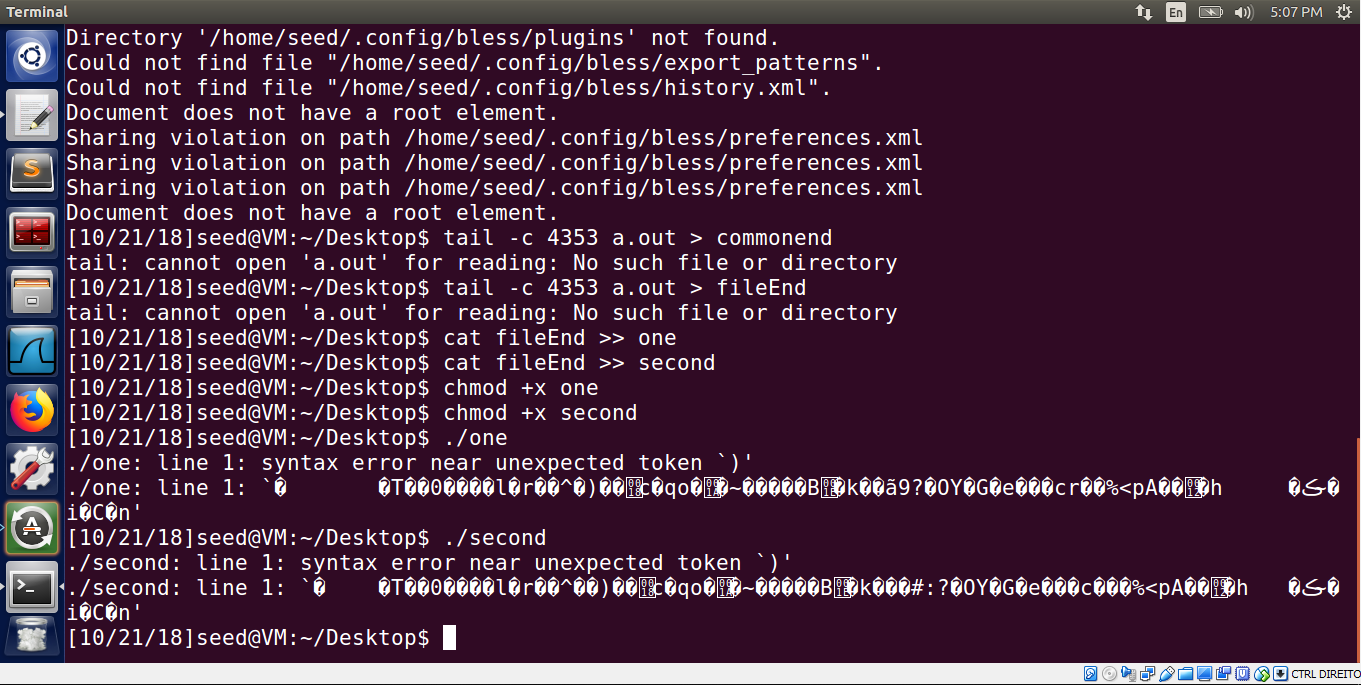


Figura 9 - Output dos dois ficheiros

## Questão 6

**Cifra**

Na implementação deste exercício optou-se por ler e gravar os ficheiros com uma dimensão definida pela variável readDimensionBlock que é parametrizável. Na cifra, a cada leitura de um bloco destas dimensões é feito o **update** da cifra e do MAC. Quando for efetuada a última leitura é chamado o método **doFinal** da cifra e do MAC.

Após esta operação o ficheiro resultante é gravado com o nome “CIPER\_RESULT.txt” que se encontra em anexo. Este ficheiro resulta da concatenação da cifra com o MAC como é pedido no enunciado

**Decifra**

A decifra tem por base o mesmo método de leitura e execução das tarefas e posterior gravação de resultados.

Por fim para realizar a verificação de autenticidade usou-se a classe **MessageDigest** a fim de obter a confirmação que o MAC obtido do ficheiro de cifra é o mesmo que o da cifra. Neste ponto não foi possível obter um resultado positivo visto que os arrays devolvidos pelo método **digest** no **modo SHA-1** não são iguais provavelmente devido a um erro não identificado.

## Questão 7

Objetivo deste exercicio é assinar ou verificar um documento passado como parametro de argumentos.

Assinar🡪 para poder assinar um documento é necessario ler o ficheiro .pfx. obtemos a instancia de uma keystore com “JKS”, de seguida lemos esse ficheiro que se encontra na pasta certs, como resource, com load, em que passamos a palavra passe “changeit”. Usamos a interface ProtectionParameter para proteger o conteudo da keystore. Depois de obter a chave privada, PrivateKey, inicianos a assinatura com a instancia signatureSHA1 ou signatureSHA256. Finalizamos o metodo ao criar um ficheiro novo com a assinatura.

Verificar 🡪 ao verificar a assinatura, temos a certeza que é um documento valido e que nao foi altereado por ninguem. Neste metodo recebemos um ficheiro, criamos um Certificate para puder obter a chave publica da mesma. E é comparado com a chave publica do certificado de quem assinou. Por fim este metodo retorna true caso a assinatura seja valida ou false caso a assinatura nao seje igual.