Segurança Informática

Fábio Parreirinha Nº39329 Tejal Amratlal Nº40605

Índice

Índice 2
Questão 1 3
Questão 2 3
Questão 3 4
3.1 4
3.2 4
Questão 4 5
4.1 5
4.2 5
Questão 5 - MD5 Collision Attack Lab 6
2.1 6
2.2 8
2.3 8
Questão 6 10
Questão 7 10

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.

Fábio Parreirinha Nº39329 Tejal Amratlal Nº40605

Questão 3

3.1

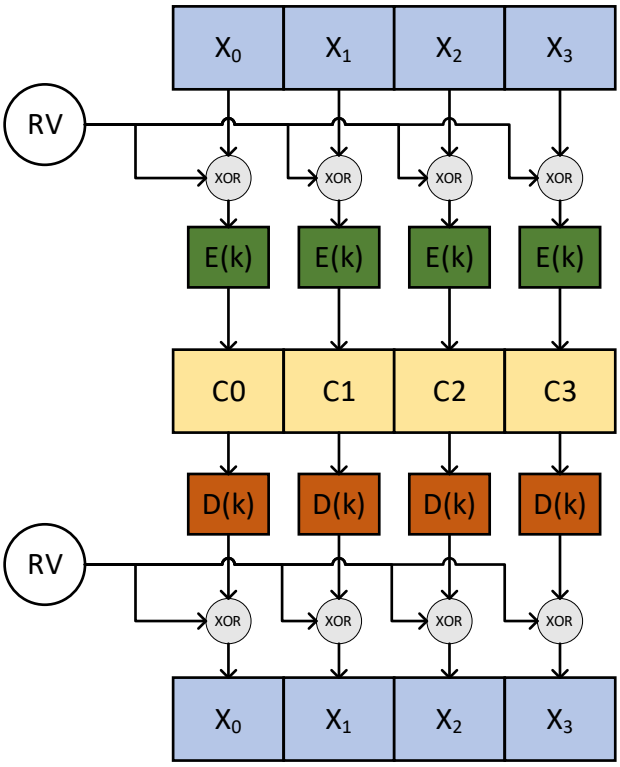

FIGURA 1 – ALGORITMO DE DECIFRA

3.2

a) 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.

b) 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.

```
636f 6e74 6575 646f 2072 616e 646f 6d20
7175 6520 6e61 6f20 7365 6a61 206d 756c
7469 706c 6f20 6465 2036 340a 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
5850 0dea f0a8 1d19 66d7 be11 71da f456
bd21 a020 47b4 7a78 2a3e 7902 39a2 08eb
46f5 f8e9 5e15 ead8 eb2f 4719 9cc9 4142
2349 cc7a f313 d6a7 069b 587d ca69 3e02
4e23 a746 d78e 34b6 21a9 3218 3bcc a83e
280e 6269 f925 2663 d15a f63b ec47 fe67
8ec5 be15 f40d 2894 b84a 59d7 8e8c 63ac
0b3a dea3 c939 b160 90df bd1b 1cc6 e57c
```

FIGURA 2 – EX 1 OUTPUT 1

```
636f 6e74 6575 646f 2072 616e 646f 6d20
7175 6520 6e61 6f20 7365 6a61 206d 756c
7469 706c 6f20 6465 2036 340a 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
5850 0dea f0a8 1d19 66d7 be11 71da f456
bd21 a0a0 47b4 7a78 2a3e 7902 39a2 08eb
46f5 f8e9 5e15 ead8 eb2f 4719 9c49 4242
2349 cc7a f313 d6a7 069b 58fd ca69 3e02
4e23 a746 d78e 34b6 21a9 3218 3bcc a83e
280e 62e9 f925 2663 d15a f63b ec47 fe67
8ec5 be15 f40d 2894 b84a 59d7 8e0c 63ac
0b3a dea3 c939 b160 90df bd9b 1cc6 e57c
```

FIGURA 3 – EX 1 OUTPUT 2

Fábio Parreirinha Nº39329 Tejal Amratlal Nº40605

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.

```
636f 6e74 6575 646f 2072 616e 646f 6d20
7061 7261 2066 617a 6572 2061 7065 6e61
7320 2036 3420 6279 7465 7320 7061 7261
2065 7374 6520 6578 6572 6369 6369 6f0a
eefb 0ce4 7b13 0ec0 14e5 114e 6ce1 3ac7
4d67 36f7 939d 3925 3f6e 7f0e 70a4 42ee
1fa9 b5c6 facd e18b eb21 3ed3 cb8d 5358
b0f1 29ac 3270 f59f dd85 5852 55cd 25e5
63d9 407d 21e1 034b 6f5b 6366 da62 d563
59b4 dc8a 781d 1300 abc3 00f7 bb5f 48d6
cb8d 4c08 97c5 fc55 7b02 4a73 d4e7 f744
ebd3 a275 0c55 8dee c0dd b9e4 bd31 a4f7
```

FIGURA 4 – EX 2 OUTPUT 1

```
636f 6e74 6575 646f 2072 616e 646f 6d20
7061 7261 2066 617a 6572 2061 7065 6e61
7320 2036 3420 6279 7465 7320 7061 7261
2065 7374 6520 6578 6572 6369 6369 6f0a
eefb 0ce4 7b13 0ec0 14e5 114e 6ce1 3ac7
4d67 3677 939d 3925 3f6e 7f0e 70a4 42ee
1fa9 b5c6 facd e18b eb21 3ed3 cb0d 5458
b0f1 29ac 3270 f59f dd85 58d2 55cd 25e5
63d9 407d 21e1 034b 6f5b 6366 da62 d563
59b4 dc0a 781d 1300 abc3 00f7 bb5f 48d6
cb8d 4c08 97c5 fc55 7b02 4a73 d467 f744
ebd3 a275 0c55 8dee c0dd b964 bd31 a4f7
```

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.

```
[10/21/18]seed@VM:~/Desktop$ md5collgen -p prefix -o one second
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'one' and 'second'
Using prefixfile: 'prefix'
Using initial value: 0123456789abcdeffedcba9876543210

Generating first block: ......
Generating second block: S01.......
Running time: 7.47867 s
[10/21/18]seed@VM:~/Desktop$ bless one second
```

FIGURA 6 - CRIAÇÃO DO PREFIXO

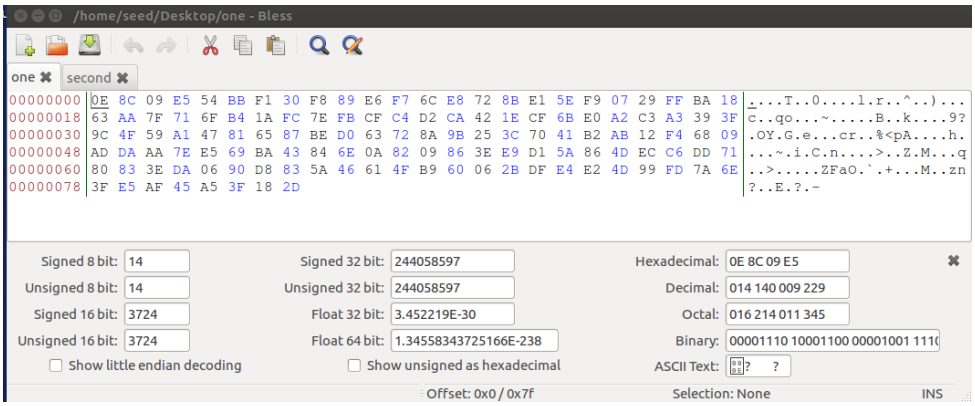

FIGURA 7 -FICHEIRO ‘ONE’

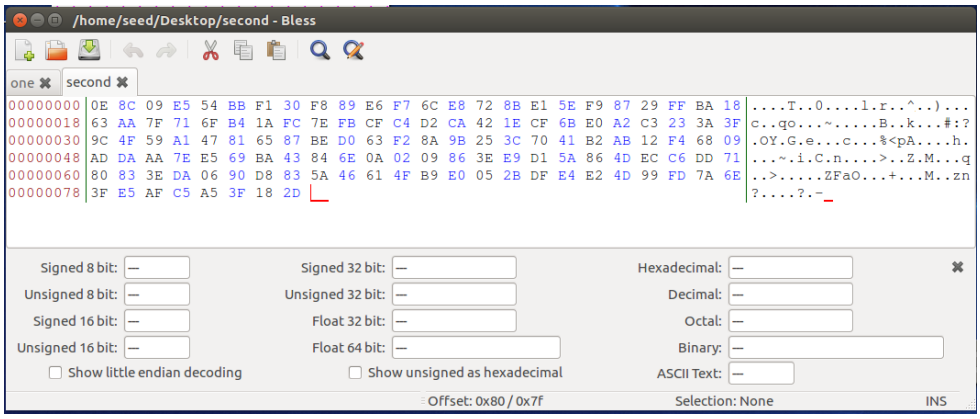

FIGURA 8 - FICHEIRO 'SECOND'

FIGURA 9 - OUTPUT DOS DOIS FICHEIROS

Fábio Parreirinha Nº39329 Tejal Amratlal Nº40605

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.