



# **Segurança em Redes de computadores**

## **Trabalho Prático 1**

**Data: 15 abril 2024**

*Docente:*

Eng. Luís Carlos Gonçalves

Luís Fonseca – 45125

*Grupo: 2*

# Índice

Índice de figuras .....	4
Introdução.....	5
Grupo 1 .....	6
Alínea 1.1.....	6
Alínea 1.2.....	6
Pergunta a) .....	6
Pergunta b) .....	7
Alínea 1.3.....	8
Pergunta a) .....	8
Pergunta b) .....	10
Pergunta c) .....	11
Pergunta d) .....	11
Pergunta e) .....	12
Alínea 1.4.....	12
Pergunta a) .....	12
Pergunta b) .....	13
Alínea 1.5.....	13
Pergunta a) .....	13
Pergunta b) .....	13
Alínea 1.6.....	13
Pergunta a) .....	13
Alínea 1.7.....	14
Pergunta a) .....	14
Pergunta b) .....	15
Pergunta c) .....	16
Pergunta d) .....	17
Pergunta e) e f).....	17
Pergunta g) .....	17
Pergunta h) .....	18
Pergunta i) .....	19
Pergunta j) .....	19
Pergunta k) .....	19
Alínea 1.8.....	20

Pergunta a) e b) e c).....	20
Pergunta d) .....	20
Pergunta e) .....	20
Pergunta f).....	22
Pergunta g) .....	22
Pergunta h) .....	23
Pergunta i) .....	23
Pergunta j) .....	23
Pergunta k) .....	24
Alínea 1.9.....	24
Pergunta a) e b) e c).....	24
Pergunta d) .....	25
Pergunta e) .....	25
Pergunta f).....	25
Pergunta g) .....	26
Pergunta h) .....	26
Alínea 1.10.....	27
Pergunta a) .....	27
Pergunta b) .....	28
Pergunta c) .....	28
Pergunta d) .....	30
Pergunta e) .....	30
Pergunta f).....	31
Pergunta g) .....	31
Alínea adicional .....	32
Conclusões .....	34
Bibliografia .....	35
Anexo .....	36

## Índice de figuras

Figura 1 conversão binário para texto.....	7
Figura 2 conversão de base64 para texto.....	9
Figura 3 conversão base 64 para texto (outra mensagem) .....	10
Figura 4 conversão base58 para texto .....	10
Figura 5 conversão hexadecimal.....	12
Figura 6 conversão duma sequência para a mensagem original .....	13
Figura 7 cifra de cipher usando o dCode 1/2.....	14
Figura 8 cifra de cipher usando o dCode 2/2.....	14
Figura 9 algoritmo MD5 .....	15
Figura 10 algoritmo MD5 .....	15
Figura 11 usando algoritmo MD5 pelo dCode .....	16
Figura 12 usando algoritmo MD5 pelo dCode .....	16
Figura 13 usando algoritmo MD4 pelo dCode .....	16
Figura 14 'crack' da mensagem usando o CrackStation.....	18
Figura 15 usando SHA1 cyberchef .....	18
Figura 16 'crack' da mensagem usando o CrackStation.....	18
Figura 17 cyberchef usando o DES como encriptação .....	20
Figura 18 cyberchef usando o DES como encriptação .....	20
Figura 19 DES com CFB.....	21
Figura 20 DES com OFB.....	21
Figura 21 DES com CTR .....	21
Figura 22 DES com ECB.....	22
Figura 23 DES com CBC and CBD padding.....	22
Figura 24 descriptação DES.....	22
Figura 25 erro ao alterar um do bit da key .....	23
Figura 26 alteração de 1byte do parâmetro IV .....	23
Figura 27 chave aleatória quando colocada no cyberchef.....	24
Figura 28 decriptação do triple DES .....	25
Figura 29 alteração de um do bit da key fornecida .....	26
Figura 30 alteração do parâmetro IV.....	26
Figura 31 crackstation para o utilizador root .....	29
Figura 32 crackstation para o utilizador admin.....	29
Figura 33 crackstation para o utilizador joe.....	30
Figura 34 crackstation para o utilizador Security.....	30
Figura 35 cyberchef com uma das passwords modificadas.....	31
Figura 36 crackstation da password modificada .....	31
Figura 37 resultado obtido em python .....	33

## Introdução

O primeiro trabalho prático da unidade curricular de Segurança em Redes de Computadores consiste no estudo de diferentes algoritmos e ferramentas que permitem a codificação e decodificação de diferentes mensagens. Na maior parte dos exercícios, são fornecidos dados para estudar os diferentes algoritmos, estudando os diferentes resultados obtidos. Em alguns casos será necessário ‘inventar’ outros dados, para comparar os diferentes valores que são obtidos, para a comparação de resultados.

Durante a leitura deste relatório, vão ser encontrados todos os resultados obtidos. Como primeira abordagem, vai ser feito um estudo teórico dos diferentes conceitos, nomeadamente comparar diferentes tipos de algoritmos e quais deles representa, em termos de segurança, o melhor. De seguida, executar exercícios práticos que demonstram os resultados obtidos.

Para a realização deste trabalho prático foram usadas diferentes ferramentas que já implementam diversos algoritmos, desde o MD5 até ao SHA1. Irá ser feita a codificação e decodificação de ambos os resultados fornecidos, e como já referido, irá ser feito um estudo e comparação dos diferentes resultados obtidos.

## Grupo 1

### Alínea 1.1

A expressão

```
(?:(?:\d|[01]?\d\d|2[0-4]\d|25[0-5])\.){3}(?:25[0-5]|2[0-4]\d|[01]?\d\d|\d)(?:\V\d{1,2})?
```

Representa a correspondência de endereços IP com uma notação CIDR.

As partes mais importantes são as seguintes:

- `(?:\d|[01]?\d\d|2[0-4]\d|25[0-5])`: corresponde aos três primeiros segmentos do endereço IP, garantindo que cada segmento seja um número válido entre 0 e 255,
- `(?:\V\d{1,2})?`: (opcionalmente) corresponde a uma notação CIDR, onde `\V` representa o caractere de barra seguido por um ou dois dígitos.

Sugestões de melhoria:

- Adicione ‘âncoras’ à expressão regular para garantir que ela corresponda à *string* evitando a correspondência de endereços IP em cadeias maiores.
- Tornar a correspondência da notação CIDR mais restritiva, (exemplo limitar o intervalo do sufixo CIDR).

De seguida, é apresentado um exemplo possível à expressão acima

```
^(?:\d|[01]?\d\d|2[0-4]\d|25[0-5])(?:\.(?:\d|[01]?\d\d|2[0-4]\d|25[0-5])){3}(?:\V\d{1,2})?$$
```

Nesta versão alternativa:

- Adicionado `^` no início e `$` no final para corresponder à *string*,
- Simplifica a expressão, removendo grupos de não captura sempre que possível

### Alínea 1.2

#### Pergunta a)

Usando a aplicação **cyberchef**, passando a seguinte mensagem binária

```
01010100 01110101 01110010 01101101 01100001 00100000 01100100 01100101
00100000 01010011 01010010 01000011 00100000 01100100 01100101 00100000
00110010 00110011 00101111 00110010 00110100
```

Obtemos a seguinte mensagem decodificada

“Turma de SRC de 23/24”

Na imagem abaixo é possível de ver um exemplo do resultado desta aplicação

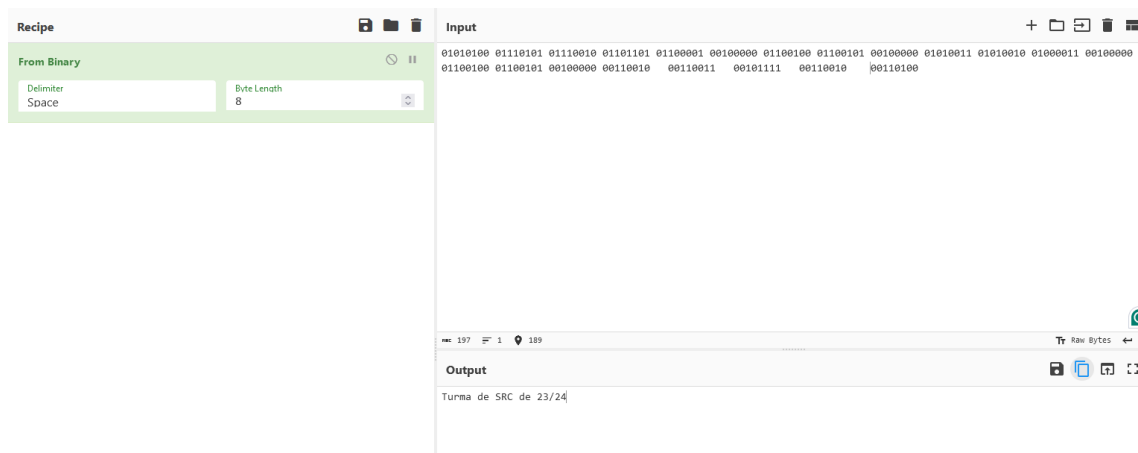


Figura 1 conversão binário para texto

## Pergunta b)

De seguida, são apresentados os conceitos teóricos e as suas diferenças, de cada definição presente

### ASCII

- Padrão de codificação de caracteres que usava 7 bits para codificar caracteres, fornecendo 128 caracteres possíveis.
- Inclui letras do alfabeto latino, dígitos e sinais de.
- Não suporta caracteres de outros idiomas ou símbolos especiais além do conjunto básico.

### Unicode:

- Padrão de codificação de caracteres que tem como objetivo abranger todos os caracteres de todos os sistemas de escrita.
- Atribui a cada caractere um ponto de código único, geralmente representado em notação hexadecimal (exemplo, U+0041 para a letra maiúscula "A" latina).
- Pode ser implementado usando diferentes esquemas de codificação, como UTF-8, UTF-16 ou UTF-32.

### UTF-8 (Unicode Transformation Format 8-bit):

- Esquema de codificação de caracteres de largura variável para Unicode.
- Usa unidades de código de 8 bits para codificar caracteres, com o número de bytes por caractere variando de 1 a 4 bytes.
- Utilizado na internet e em sistemas de computação devido à sua compatibilidade com ASCII e eficiência na representação de caracteres.

**UTF-16** (Unicode Transformation Format 16-bit):

- Esquema de codificação de caracteres de largura variável para Unicode.
- Usa unidades de código de 16 bits para codificar caracteres, com o número de unidades de código por caractere variando de 1 a 2.
- Os caracteres do Plano Multilíngue Básico (BMP), que incluem a maioria dos caracteres usados comumente, são codificados com uma única unidade de 16 Bits.
- Caracteres fora do BMP são codificados usando pares de unidades de 16 bits (pares substitutos).
- Usado em plataformas onde caracteres fora do BMP são frequentemente encontrados, como em idiomas asiáticos.

Em suma, ASCII é um padrão de codificação de caracteres básico, estando limitado ao alfabeto inglês e alguns caracteres especiais.

Unicode é um padrão mais amplo que abrange caracteres de todos os sistemas de escrita.

UTF-8 e UTF-16 são esquemas de codificação para Unicode, sendo UTF-8 mais popular na internet devido à sua compatibilidade com ASCII e eficiência, enquanto UTF-16 é frequentemente usado em ambientes que exigem suporte para caracteres fora do Plano Multilíngue Básico.

## Alínea 1.3

### Pergunta a)

#### Alínea i)

Usando a aplicação **cyberchef**, passando a seguinte mensagem em base64(devido ao facto de na última parte da mensagem, existem dois iguais '==')

TmFkYSBkZSBjb3BpYXlgZGEgTmV0IG91IGRIIHVzYXlgbyBjaGF0R1BUIQ==

Obtemos a seguinte mensagem:

“Nada de copiar da Net ou de usar o chatGPT!”

Na figura abaixo é possível de ver um exemplo possível exemplo da execução desta aplicação.



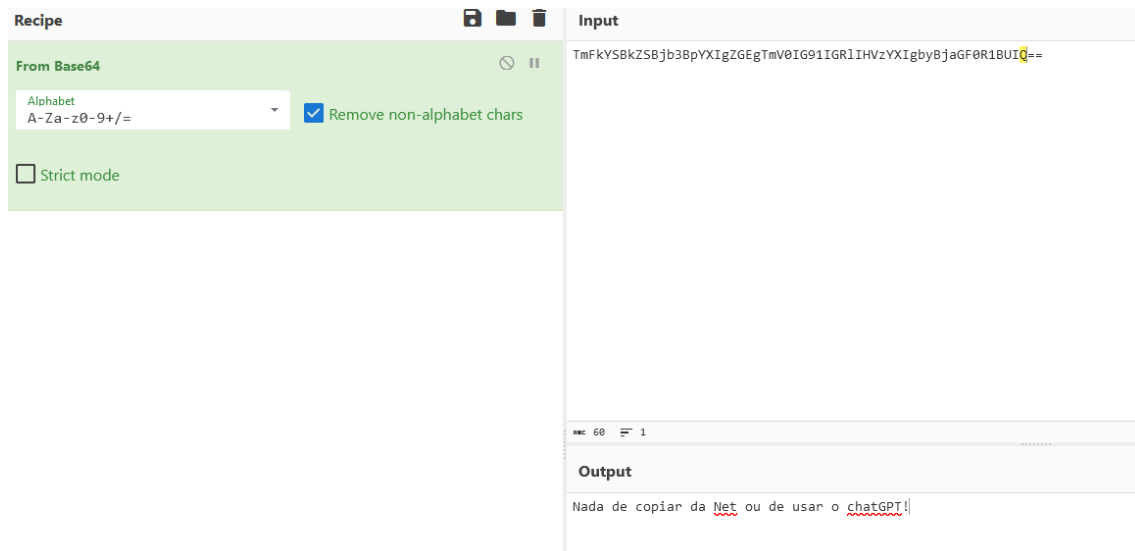


Figura 2 conversão de base64 para texto

### Alínea ii)

O tipo de codificação usado nessa mensagem é Base64.

É usado para codificar dados binários em caracteres ASCII.

É frequentemente utilizada em cenários onde dados binários precisam de ser transmitidos por meios que são projetados para lidar com dados textuais, como sistemas de email ou codificação de dados binários em URLs.

A codificação Base64 alcança isso representando dados binários usando um conjunto de 64 caracteres ASCII que são seguros para transmitir em protocolos baseados em texto.

No cenário para descodificar a mensagem, era necessário um descodificador Base64, onde convertia a *string* para a sua mensagem original.

### Alínea iii)

Usando o mesmo código, obtemos o seguinte exemplo com os membros do grupo

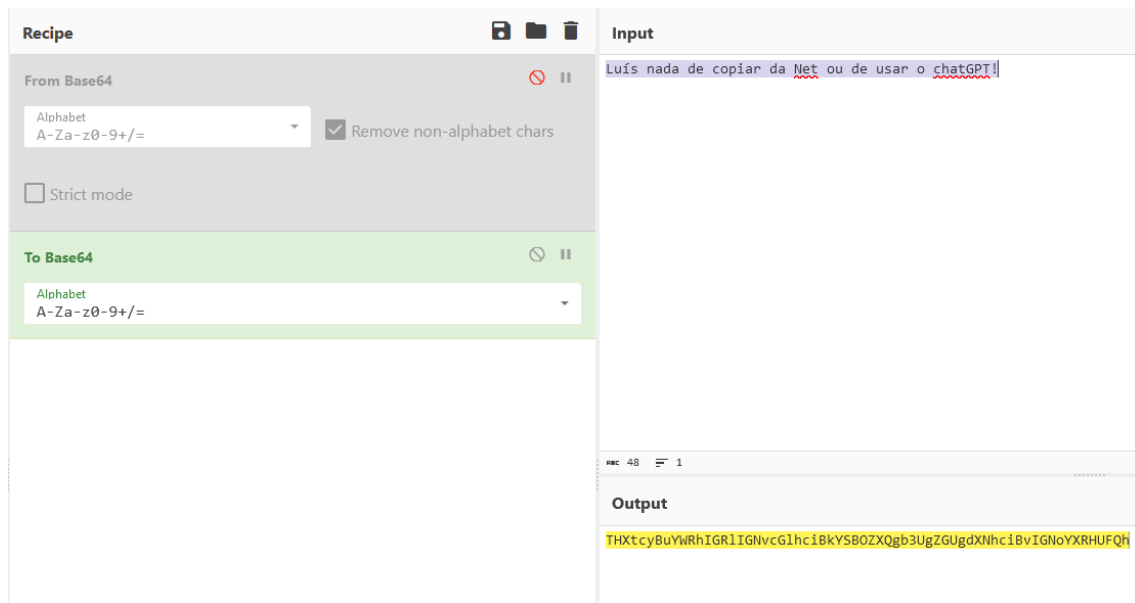


Figura 3 conversão base 64 para texto (outra mensagem)

## Pergunta b)

### Alínea i)

Usando a aplicação **cyberchef**, passando a seguinte mensagem em base58.

wTBQFebwJvCy9Txw4o3NG3FZgPD3i73zpRTynyCBLC96NuQxhbF2K

Obtemos a seguinte mensagem

“Segurança em Redes de Computadores 2324”

Na imagem abaixo é possível de ver um exemplo feito

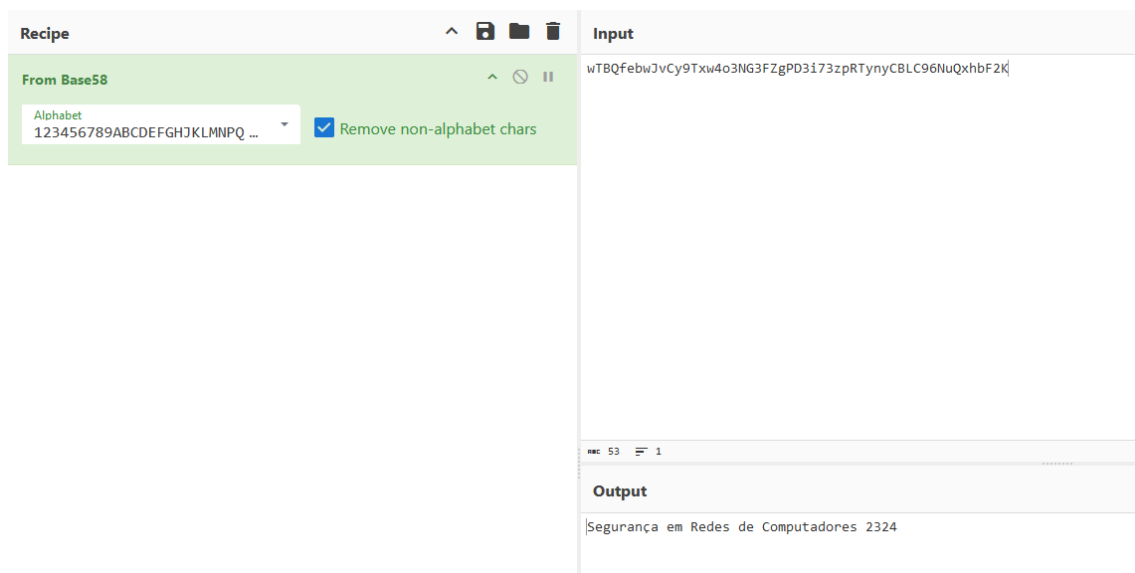


Figura 4 conversão base58 para texto

## Alínea ii)

O tipo de codificação usado é a base58.

## Pergunta c)

Base64 e Base58 são dois métodos de codificação usados para representar dados binários em texto legível.

### **Tamanho do alfabeto:**

Base64 utiliza um alfabeto de 64 caracteres, composto por letras maiúsculas e minúsculas, dígitos numéricos e caracteres especiais.

Base58 utiliza um alfabeto de 58 caracteres, excluindo alguns caracteres que podem ser confundidos, como '0' (zero) e 'O' (letra O), '1' (um) e 'l' (letra L).

### **Caracteres excluídos:**

Base58 exclui certos caracteres do alfabeto Base64 que podem ser confundidos entre si ou que são considerados problemáticos em certos contextos. Por exemplo, os caracteres '+' e '/' são usados em URLs e sistemas de arquivos, sendo excluídos na Base58.

### **Tamanho da saída:**

Devido ao tamanho do alfabeto, a codificação Base64 geralmente produz uma saída mais compacta do que Base58 para a mesma entrada. Isso significa que Base64 é mais eficiente em termos de espaço quando a compactação é importante.

## Pergunta d)

### **Uso:**

Base64 é amplamente usado em uma variedade de contextos, incluindo comunicações na web, armazenamento de dados binários em bases de dados.

Base58 é comumente usado em cripto moedas(Bitcoin), para representar endereços públicos de forma legível e compacta.

## Pergunta e)

Não. A criptografia envolve transformar dados em um formato que seja ininteligível sem a chave de criptografia apropriada, enquanto a codificação simplesmente representa dados em um formato diferente sem necessariamente ocultar seu significado.

Ambas as sequências estão codificadas com diferentes codificações, uma de base64 e outra de base58.

A criptografia, por outro lado, envolve a conversão de texto simples em texto cifrado usando algoritmos criptográficos e chaves. Sem a chave de criptografia, o texto original não pode ser facilmente recuperado.

A criptografia é usada para garantir a segurança dos dados enquanto são armazenados ou transmitidos, enquanto a codificação é frequentemente usada para fins de representação e transformação de dados.

## Alínea 1.4

### Pergunta a)

Usando a aplicação **cyberchef**, passando a seguinte mensagem em hexadecimal.

```
37 55 71 6e 78 43 4a 4d 54 7a 4a 66 68 47 6d 4d 62 37 44 32 50 75 55 50 61 33 71 77 33 31
34 6d 7a 70 38 6a 71 41 56 38 6d 4a 44 76 5a 76 64 6a 33 62 47 66 53 79 34 68 4c 59 68 6e
69 79 43 75 5a 57 6d 6e 72 39 78 63 44 6d 55 53 53 75 52 52 47 67 76 6d 57 47 46 67 79
```

Obtemos a seguinte mensagem

```
7UqnxCJMTzJfhGmMb7D2PuUPa3qw314mzp8jqAV8mJDvZvdj3bGfSy4hLYhniyCuZWmn9
xcDmUSSuRRGgvmWGFgy
```

Na imagem abaixo é possível de ver um exemplo feito

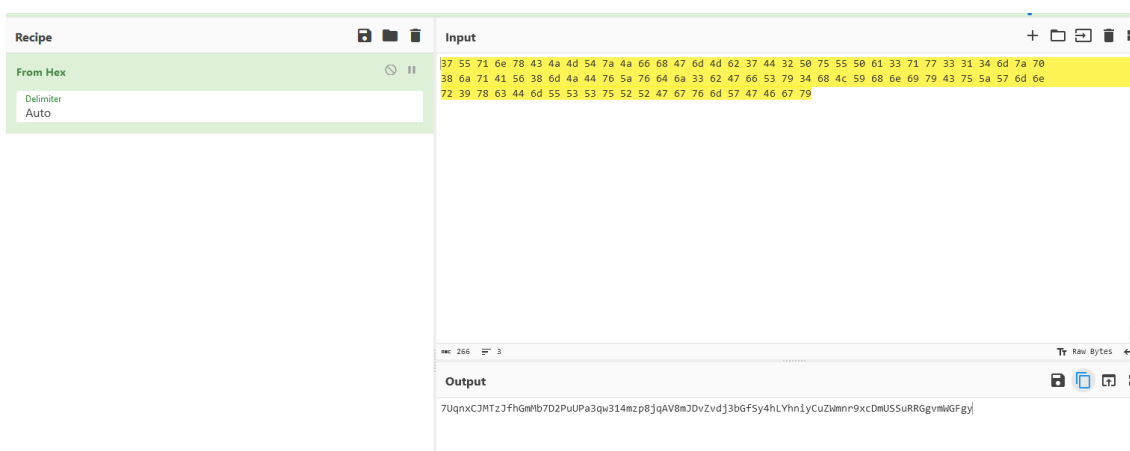


Figura 5 conversão hexadecimal

## Pergunta b)

O tipo de codificação é hexadecimal.

## Alínea 1.5

### Pergunta a)

Usando a aplicação **cyberchef**, passando a seguinte mensagem em base64.

```
U1JDKE8gc2VncmVkbYBkaXN0byDDqSBuw6NvIHNIIGRlc2lzdGlyIGZhY2lsbWVudGUuIFNI  
ciBwZXJzaXN0ZW50ZS4pCg==
```

Obtemos a seguinte mensagem

SRC(O segredo disto é não se desistir facilmente. Ser persistente.)

Na imagem abaixo é possível de ver um exemplo feito

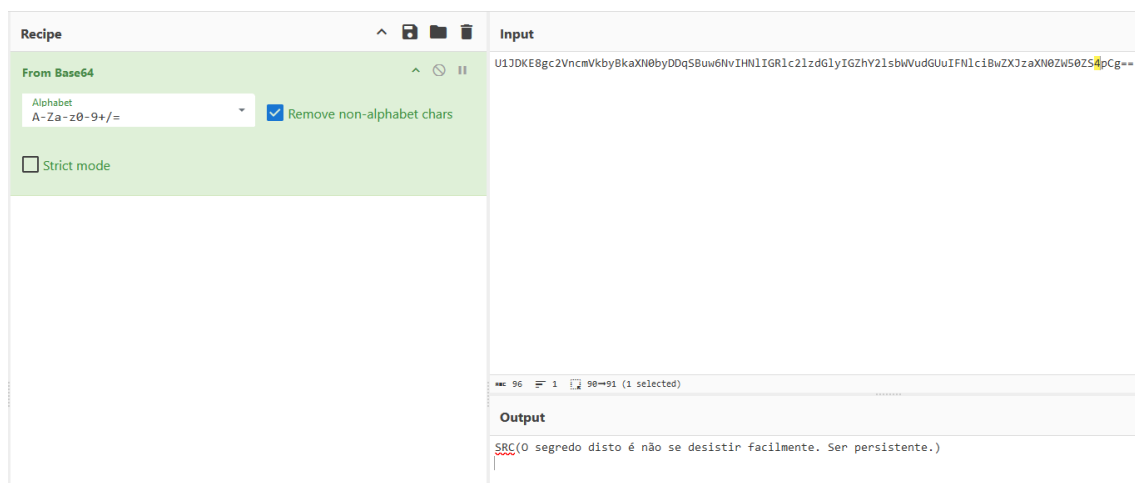


Figura 6 conversão duma sequência para a mensagem original

## Pergunta b)

Transformação de texto para base64.

## Alínea 1.6

### Pergunta a)

Usando o website dCode, com a cifra de cipher, com as seguintes mensagens

1. ("NMX(Dnoj nzmqz kmv oznoz yz phv xdamv hpdoj xjiczxdyv.")
2. POZ(Jxfp al jbpjl. X afczriaxab fox pryfkal klp bubozfzflp moxqfzlp.)

Para a primeira mensagem, obtemos o seguinte resultado

“("SRC(Isto serve para teste de uma cifra muito conhecida.)”

Na imagem abaixo é possível de ver o processo



Figura 7 cifra de cipher usando o dCode 1/2

Para a mensagem em 2), obtemos a seguinte mensagem

SRC(Mais do mesmo. A dificuldade ira subindo nos exercicios praticos.)

Na imagem abaixo, é possível de ver o resultado



Figura 8 cifra de cipher usando o dCode 2/2

Para ambos os casos, foi necessária fazer o processo 25 vezes.

## Alínea 1.7

### Pergunta a)

Usando a mensagem, vai ser codificado em MD5.

"SRC(Vamos ver os hashes.)"

Obtém se o seguinte valor

d7cf6746ebf88b48e5223429df2121fd

É possível de ver na imagem seguinte o exemplo aplicado

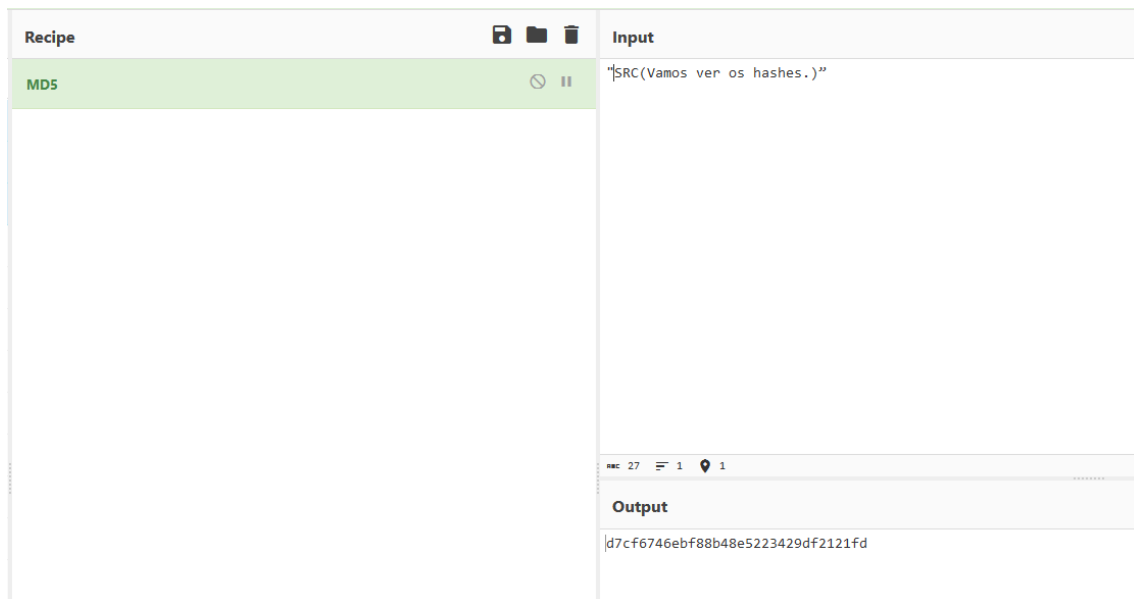


Figura 9 algoritmo MD5

### Pergunta b)

Usando a mensagem SRC(Vamos ver os hashes.)

Obtém se o seguinte resultado

38924ba0d78463eabb547c16a8ebbe43

É possível de ver na imagem seguinte o exemplo aplicado

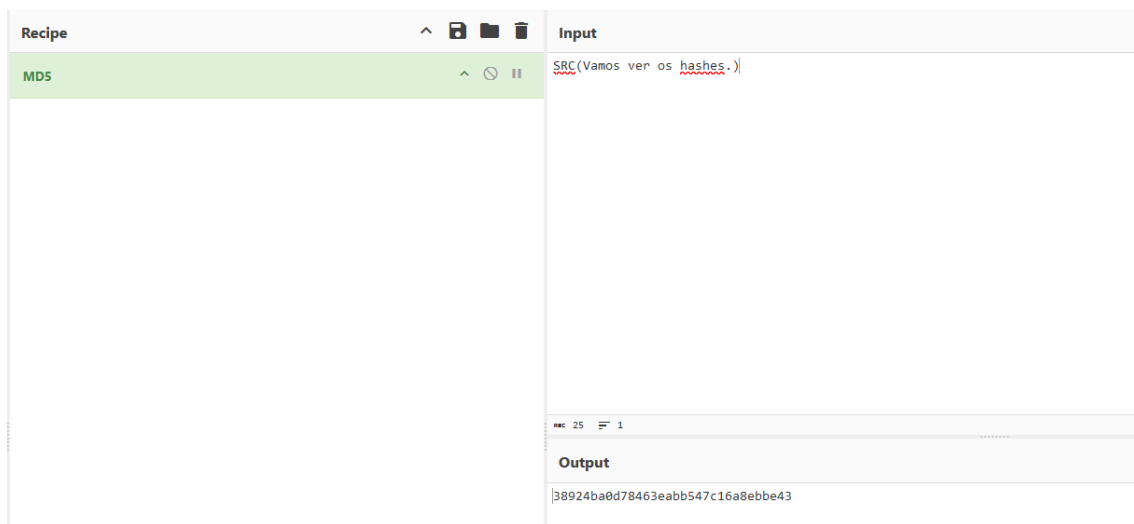


Figura 10 algoritmo MD5

Comparando os dois resultados, é possível de ver que os hashes MD5 são diferentes.

Essa discrepância ocorre devido a pequenas diferenças nas entradas de cada frase, como espaços, pontuação ou letras maiúsculas.

Nesse caso, os hashes são diferentes, embora as *strings* de entrada pareçam idênticas.

Isto demonstra a sensibilidade do MD5 a alterações em cada frase, razão pela qual o MD5 não é recomendado para fins criptográficos devido à sua vulnerabilidade a ataques de colisão.

No entanto, para fins não criptográficos, (somas de verificação ou pesquisas em tabelas hash) o MD5 ainda pode ser adequado.

### Pergunta c)

Usando o dcode, obtemos os seguintes resultados

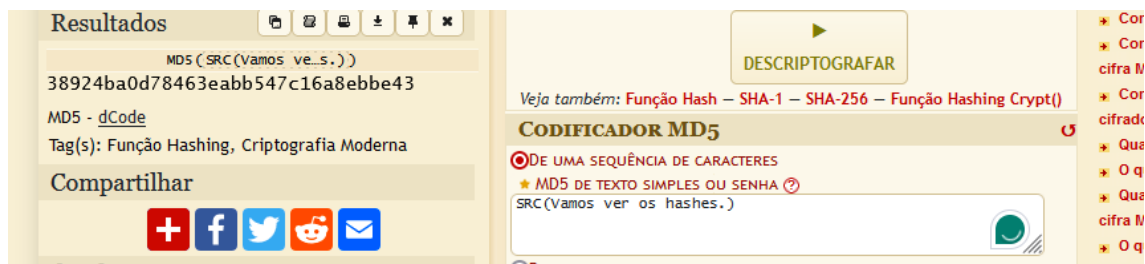


Figura 11 usando algoritmo MD5 pelo dCode



Figura 12 usando algoritmo MD5 pelo dCode

Como é possível de ver, os resultados são idênticos usando o MD5 em ambos os casos.

Experimentando com o algoritmo MD4 obtemos o seguinte resultado



Figura 13 usando algoritmo MD4 pelo dCode

Alterando o algoritmo é possível de ver que a mensagem codificada é diferente, quando usando diferentes algoritmos. Apenas é de notar que o começo de ambas as mensagens é a mesma ("3a...").



### Pergunta d)

Tentado descodificar ambas as mensagens na alínea a), é possível de ver que não é possível voltar para a mensagem original.

Tal facto deve-se ao facto de o MD5 não conseguir "descodificar" a mensagem de volta para o original. MD5 é uma função hash unidirecional, o que significa que ela converte dados de entrada em uma sequência de bytes de tamanho fixo (no caso do MD5, 128 bits ou 16 bytes), mas não é possível reverter esse processo para obter os dados originais a partir do hash.

A função MD5 é projetada para ser computacionalmente difícil de reverter. Isso significa que, dada uma sequência de hash, é extremamente difícil (teoricamente impossível em uma prática viável) encontrar uma entrada que gere esse hash específico. Portanto, o MD5 é usado principalmente para verificar a integridade dos dados ou para fins de verificação, não para criptografia ou segurança de dados.

### Pergunta e) e f)

Não é possível recuperar a mensagem original, visto que o MD5 não é reversível, o que significa que não é possível obter a mensagem original a partir do valor de hash gerado.

### Pergunta g)

Did you manage to decrypt?

R: Yes

What is the original message?

R: Test12345

Why were you able to decrypt this via Crackstation and the previous one not?

Está contida no MD5 na base de dados. Não existe o valor.

Usando a frase SRC2223 obtemos a seguinte sequência:

2d8d8b5b9d0bd229e43e675c7f9b9eba


No entanto, como o MD5 é uma função de hash criptográfica, não é possível obter o valor da mensagem original.

**Free Password Hash Cracker**

---

Enter up to 20 non-salted hashes, one per line:

Não sou um robô

  
Privacidade - Termos de Utilização

Crack Hashes

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults

Hash	Type	Result
662af1cd1976f09a9f8cecc868ccc0a2	md5	Test12345

**Color Codes:** Green Exact match, Yellow Partial match, Red Not found.

Figura 14 'crack' da mensagem usando o CrackStation

### Pergunta h)

Usando como exemplo a pass '123456789', obtemos a seguinte sequência usando o cyberchef: f7c3bc1d808e04732adf679965cccc34ca7ae3441

**Recipe**

MD5

SHA1

Rounds  
 80

**Input**

123456789

**Output**

f7c3bc1d808e04732adf679965cccc34ca7ae3441

Figura 15 usando SHA1 cyberchef

Se usarmos este código no crackstation, conseguimos recuperar esta mensagem.


Neste caso conseguimos obter usando o SHA1.

**Free Password Hash Cracker**

---

Enter up to 20 non-salted hashes, one per line:

Não sou um robô

  
Privacidade - Termos de Utilização

Crack Hashes

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults

Hash	Type	Result
f7c3bc1d808e04732adf679965cccc34ca7ae3441	sha1	123456789

**Color Codes:** Green Exact match, Yellow Partial match, Red Not found.

Figura 16 'crack' da mensagem usando o CrackStation

### Pergunta i)

Não existe reversibilidade nesta sequência pois as funções de hash são projetadas para serem irreversíveis. Estas funções de hash geram uma sequência de caracteres de tamanho fixo a partir de dados de entrada de qualquer tamanho, e mesmo uma pequena alteração nos dados de entrada deve produzir uma saída de hash significativamente diferente. Essa propriedade é crucial para várias aplicações de segurança e integridade de dados.

### Pergunta j)

Sim, visto que o MD5 é uma função de hash criptográfica que gera um valor de hash de tamanho fixo (128 bits).

No entanto, devido às suas vulnerabilidades, é possível encontrar entradas diferentes que resultam no mesmo valor de hash. Esses pares de entradas são chamados de colisões.

As duas strings fornecidas são exemplos de colisões MD5. Apesar de terem conteúdos diferentes, produzem o mesmo valor de hash.

### Pergunta k)

#### **Para a primeira string:**

```
d131dd02c5e6eec4693d9a0698aff95c  
2fcab58712467eab4004583eb8fb7f89  
55ad340609f4b30283e488832571415a  
085125e8f7cdc99fd91dbdf280373c5b  
d8823e3156348f5bae6dacd436c919c6  
dd53e2b487da03fd02396306d248cda0  
e99f33420f577ee8ce54b67080a80d1e  
c69821bcb6a8839396f9652b6ff72a70
```

#### **Para a segunda string:**

```
d131dd02c5e6eec4693d9a0698aff95c  
2fcab50712467eab4004583eb8fb7f89  
55ad340609f4b30283e4888325f1415a  
085125e8f7cdc99fd91dbd7280373c5b  
d8823e3156348f5bae6dacd436c919c6  
dd53e23487da03fd02396306d248cda0  
e99f33420f577ee8ce54b67080280d1e  
c69821bcb6a8839396f965ab6ff72a70
```

## Alínea 1.8

Mensagem: “Learn Computer Network Security doing practical exercises.”

Modo encriptação: DES (Triple DES Encrypt) com modo CBC.

Com os seguintes parâmetros:

1. Key = ffffffff (8 bytes)
2. IV = 00ff00ff00ff00ff (8 byte)

### Pergunta a) e b) e c)

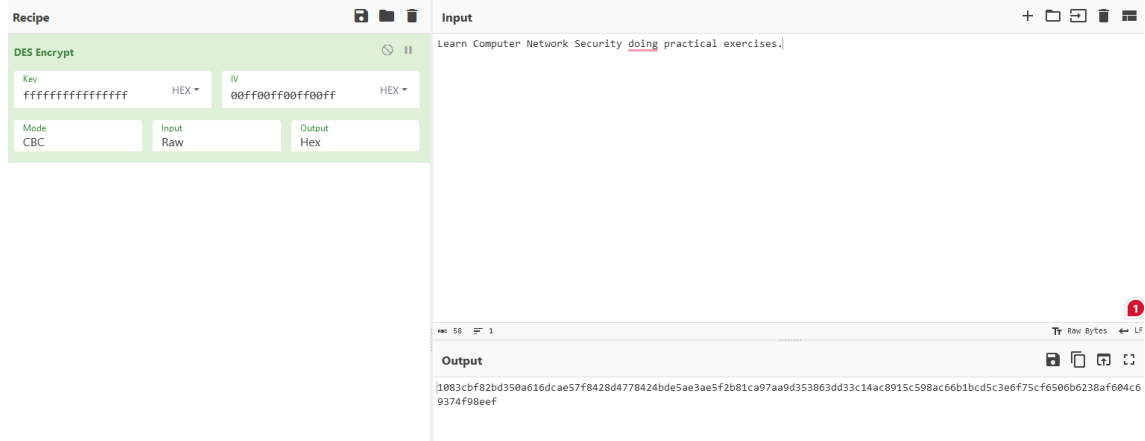


Figura 17 cyberchef usando o DES como encriptação

### Pergunta d)

R: Não, o número de bytes é diferente

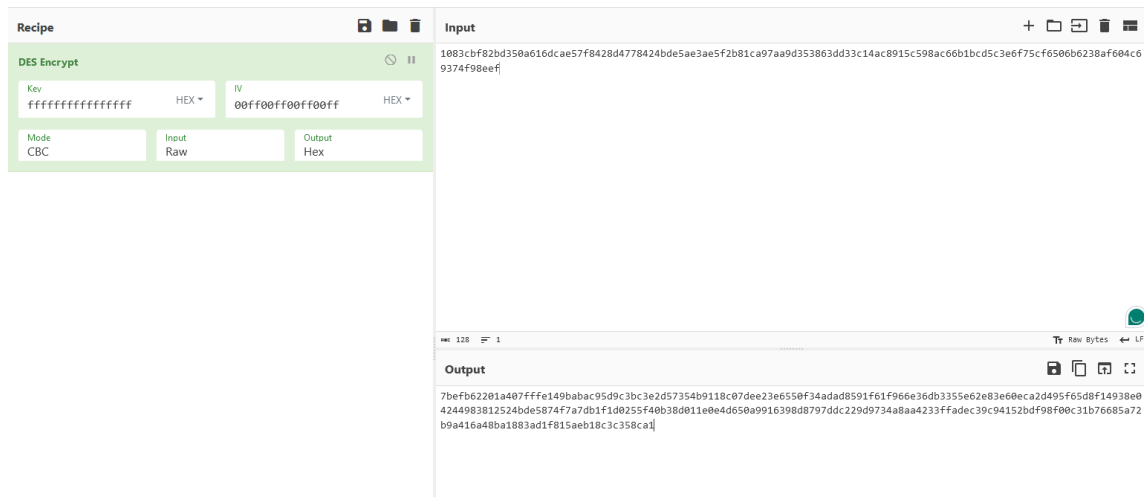


Figura 18 cyberchef usando o DES como encriptação

### Pergunta e)

Nesta pergunta, apenas é alterado o ‘modo’ como é usado o DES, obtendo diferentes resultados.

## Usando CFB

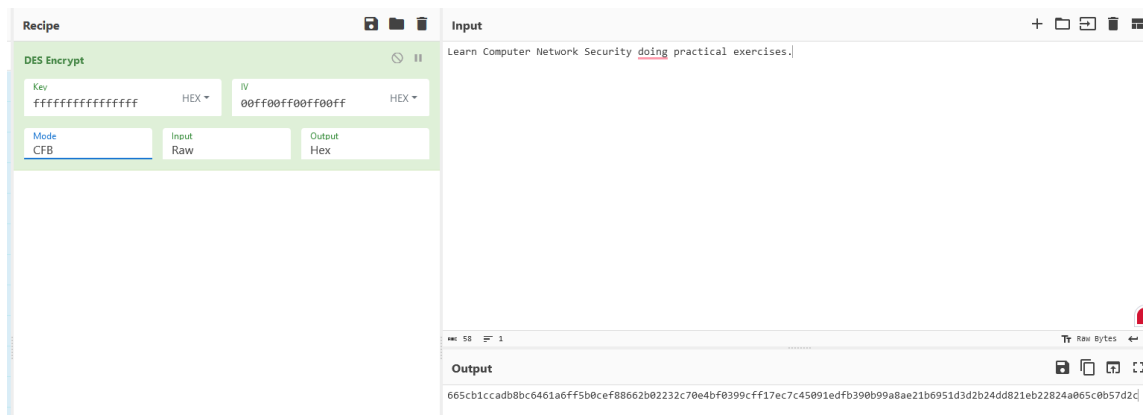


Figura 19 DES com CFB

## Usando OFB

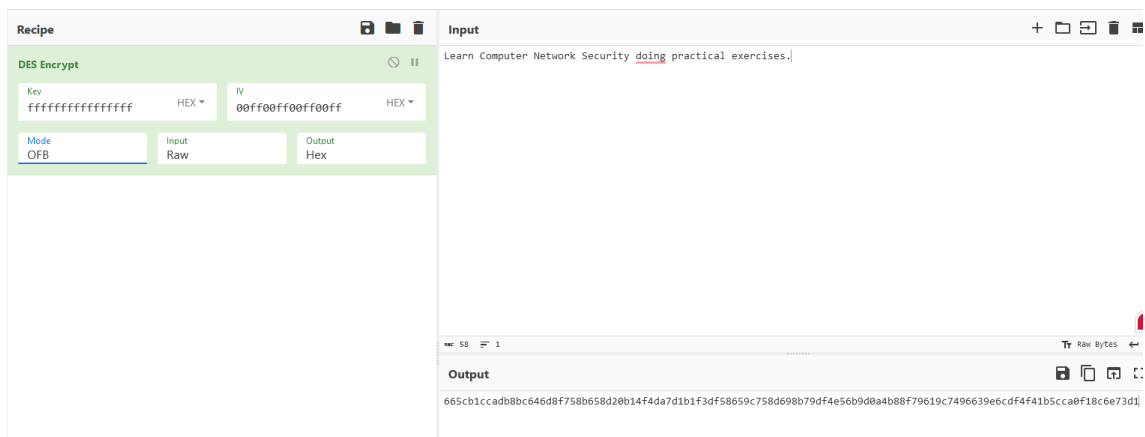


Figura 20 DES com OFB

## Usando CTR

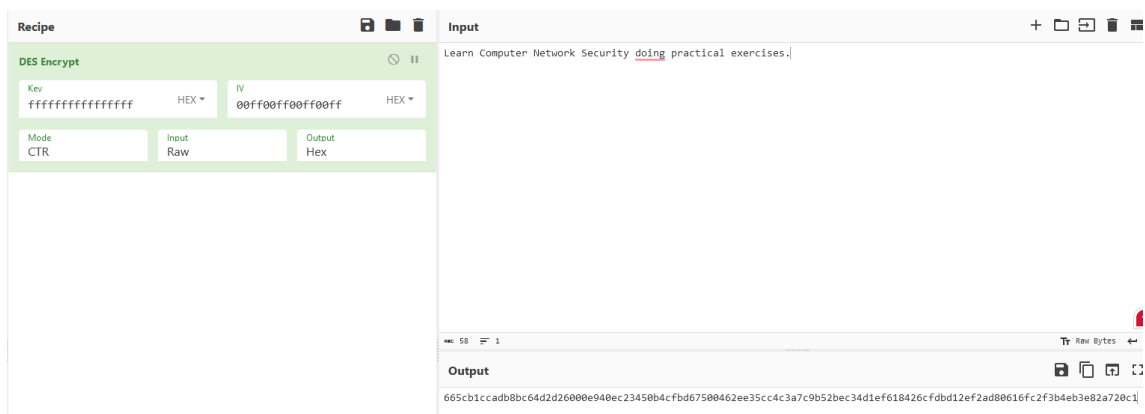


Figura 21 DES com CTR

## Usando ECB

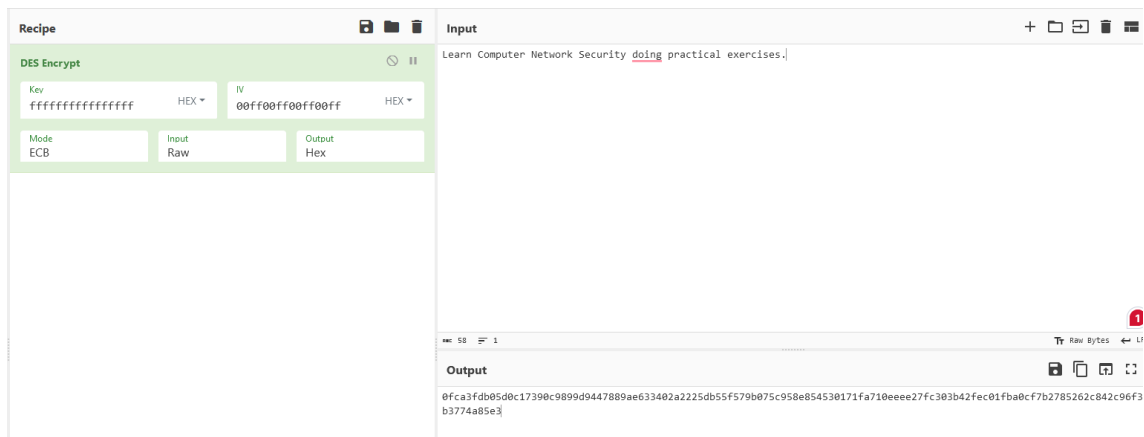


Figura 22 DES com ECB

## Pergunta f)

Usando CBC and CBC Padding, é possível de obter a mensagem original.

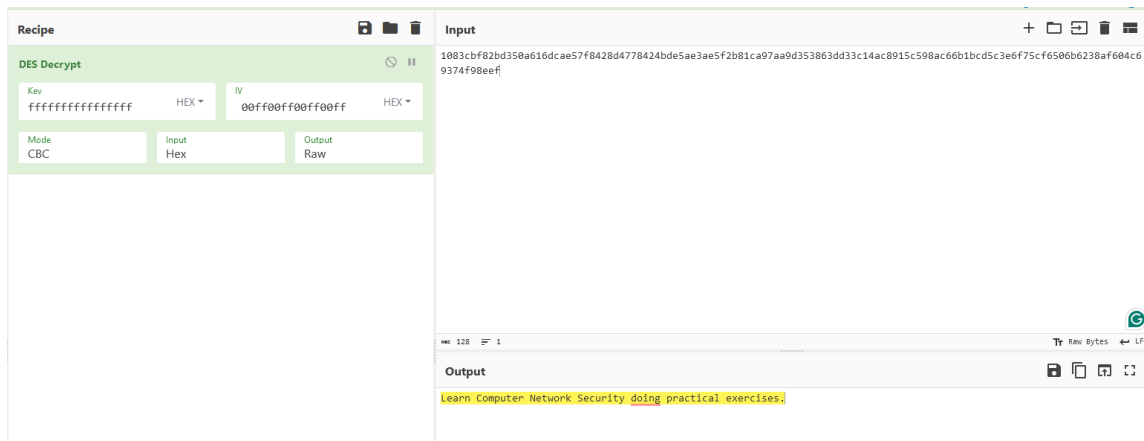


Figura 23 DES com CBC and CBD padding

Com os outros modos, não é possível recuperar a mensagem original

## Pergunta g)

Fazendo a decodificação, é possível de obter a mensagem original.

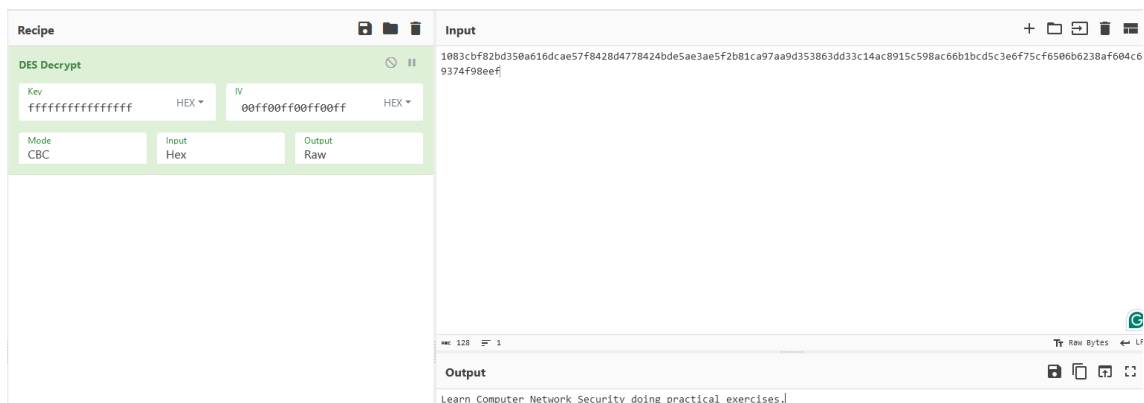


Figura 24 descriptação DES

### Pergunta h)

Ao mudar 1 bit da key fornecida não é possível recuperar a mensagem original.

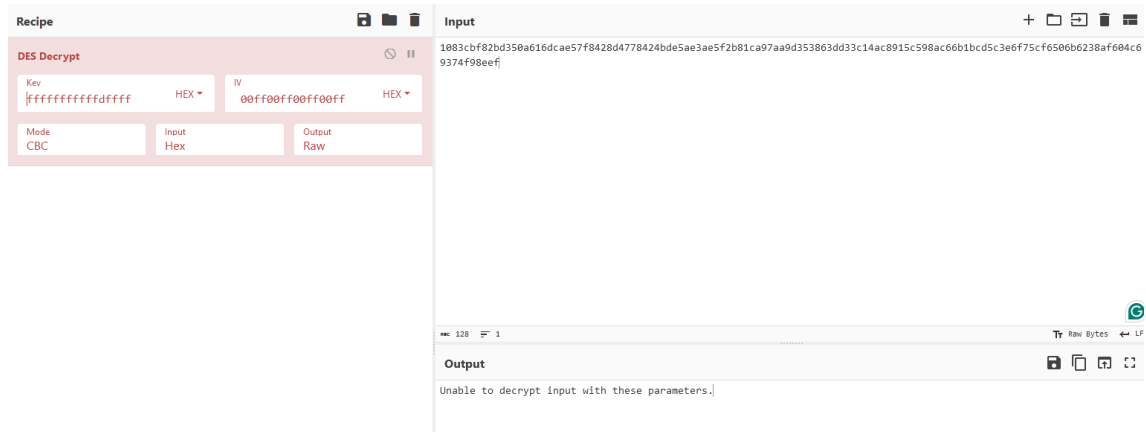


Figura 25 erro ao alterar um do bit da key

### Pergunta i)

Alterando 1 byte do parâmetro “IV” não é possível recuperar a mensagem original

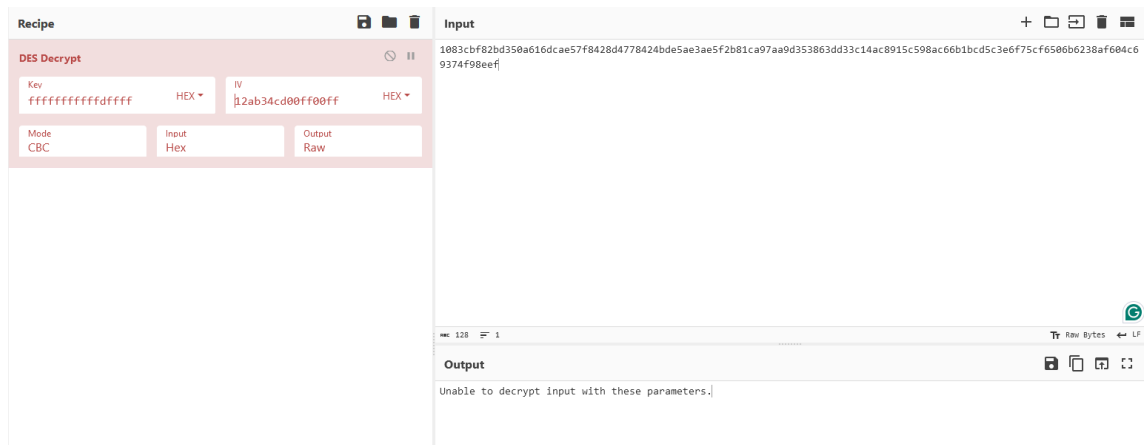


Figura 26 alteração de 1byte do parâmetro IV

### Pergunta j)

O algoritmo de criptografia DES não oferece resistência contra alterações na chave (key tampering) ou mudanças no IV (Initialization Vector). No entanto, o uso do modo CBC (Cipher Block Chaining) melhora a segurança do processo de criptografia.

No modo CBC, cada bloco de texto simples (plaintext) é combinado com o bloco de texto cifrado (ciphertext) anterior por meio de uma operação XOR antes da criptografia. Isso introduz difusão, tornando mais difícil para um atacante manipular o texto cifrado para afetar diretamente o texto simples.

Quanto há alteração da chave, se um atacante modificar a chave, o processo de criptografia resultará numa saída inválida. O destinatário não conseguirá descriptografar a mensagem corretamente a menos que possua a chave correta.

Em relação às mudanças no IV, se for alterado, vai afetar a criptografia dos blocos subsequentes. No entanto, isso não expõe diretamente a chave nem permite uma descriptografia direta. Mudar o IV garante que mesmo se o mesmo texto simples for criptografado várias vezes com a mesma chave, o texto cifrado resultante será diferente, aumentando a segurança.

No entanto, é importante observar que o DES em si é considerado fraco pelos padrões modernos devido ao seu tamanho de chave curto (56 bits), tornando-o vulnerável a ataques de força bruta.

### Pergunta k)

Colocando uma chave aleatória, não é possível obter a mensagem original.

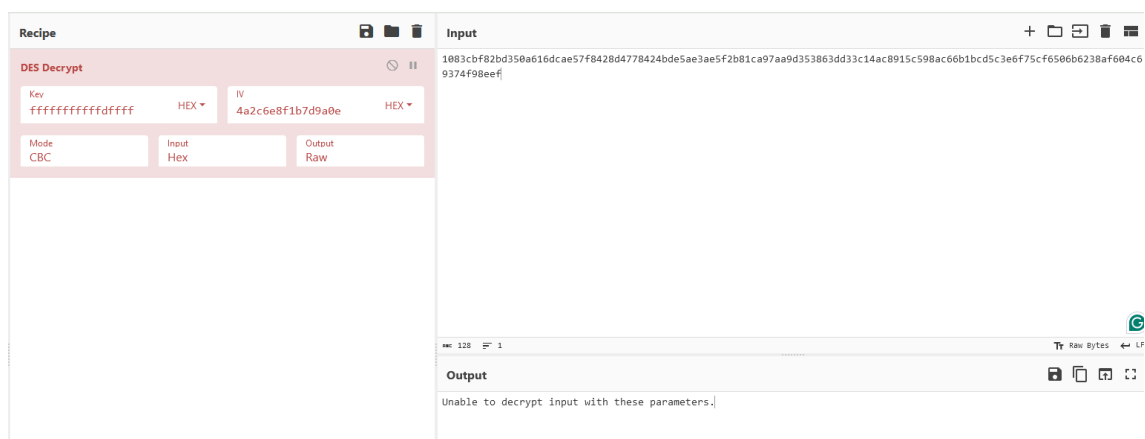


Figura 27 chave aleatória quando colocada no cyberchef

## Alínea 1.9

Dados fornecidos em baixo.

Mensagem: “The Computer Network Security subject is just one of several subjects taught in the area of cybersecurity at DEETC.”

Modo encriptação: 3DES (Triple DES Encrypt) com modo CBC.

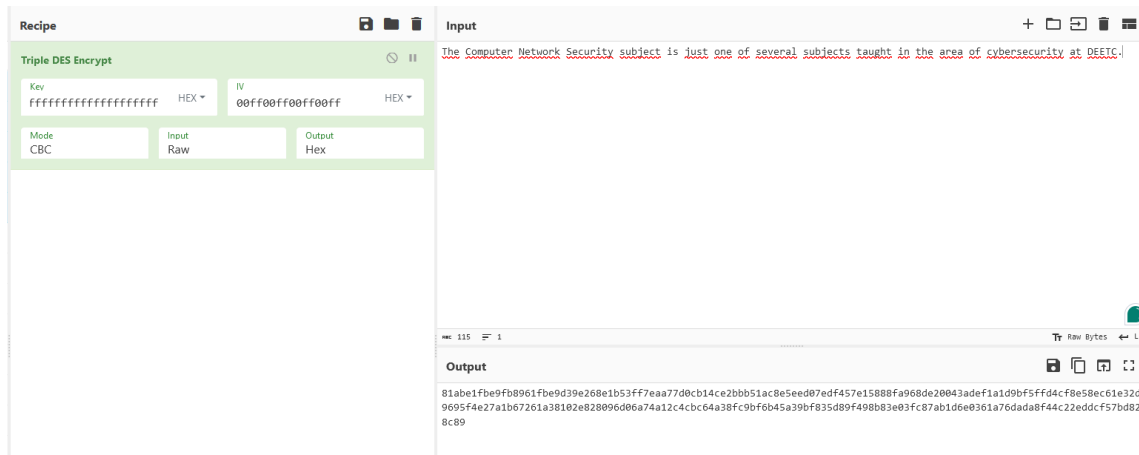
Com os seguintes parâmetros:

3. Key = ff (24 bytes, no spaces)
4. IV = 00ff00ff00ff00ff (8 byte)

### Pergunta a) e b) e c)

Usando 3DES com os parâmetros obtemos a seguinte sequência





### Pergunta d)

Não, o resultado é diferente do exercício anterior.

### Pergunta e)

A mensagem codificada é a seguinte:

81abe1fbe9fb8961fbe9d39e268e1b53ff7eaa77d0cb14ce2bbb51ac8e5eed07edf457e1588fa968de20043adef1a1d9bf5ffd4cf8e58ec61e32d9695f4e27a1b67261a38102e828096d06a74a12c4cbc64a38fc9bf6b45a39bf835d89f498b83e03fc87ab1d6e0361a76dada8f44c22eddcf57bd828c89

Fazendo decriptação obtemos a mensagem original

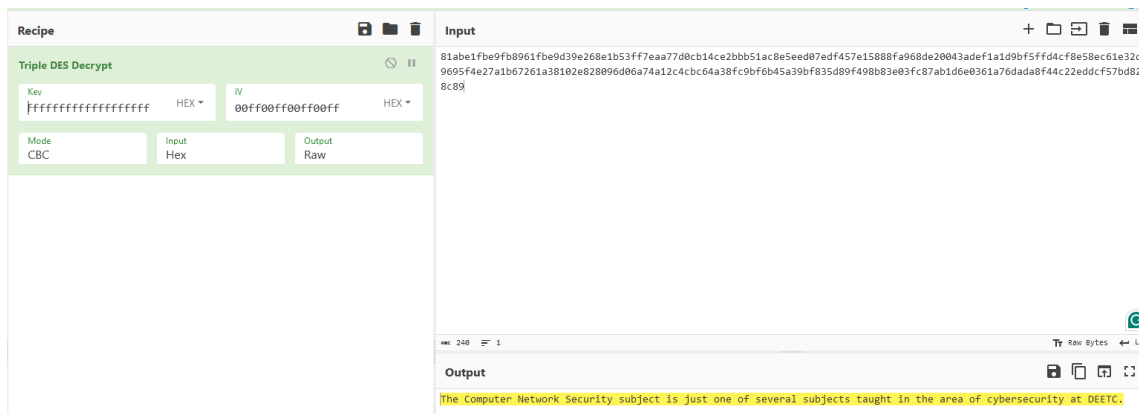


Figura 28 decriptação do triple DES

### Pergunta f)

Mudando um dos bits da Key, não é possível obter a mensagem original

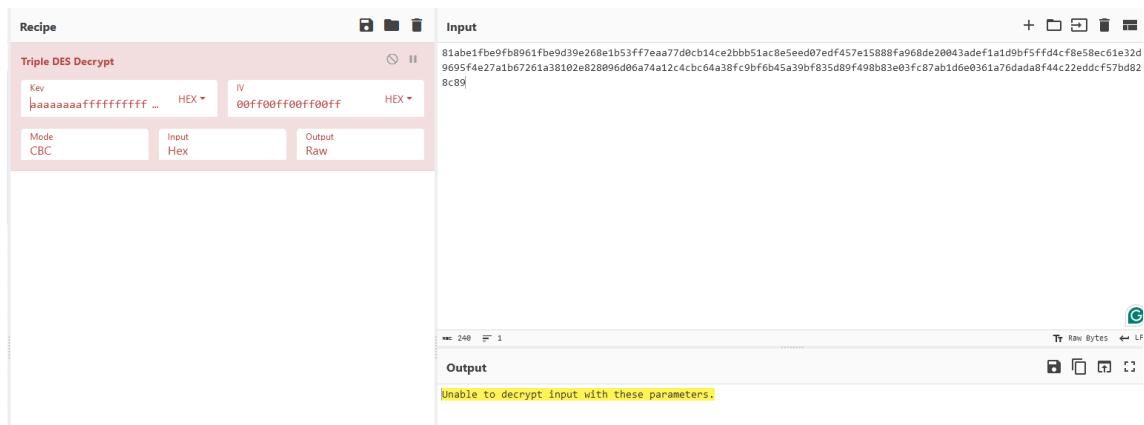


Figura 29 alteração de um do bit da key fornecida

### Pergunta g)

Alterando o IV, não é possível recuperar a mensagem original.

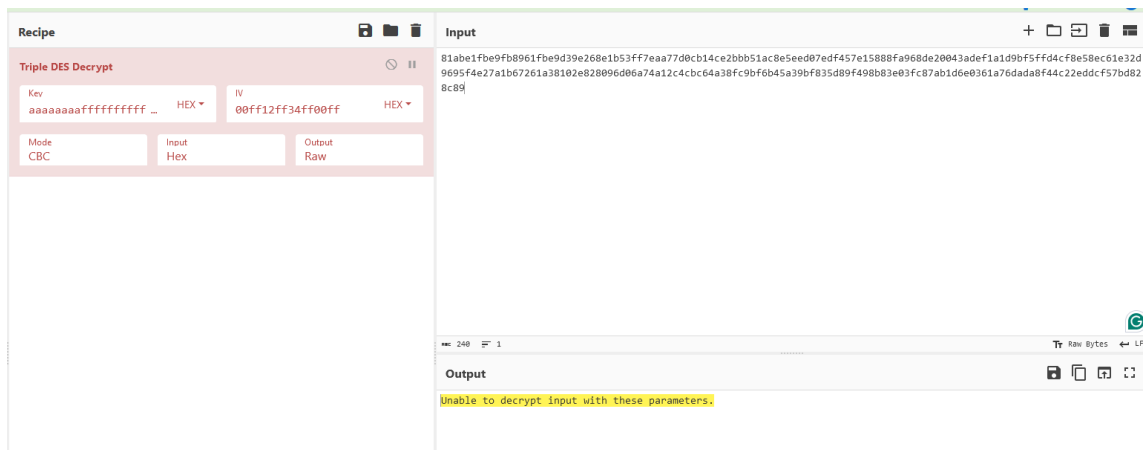


Figura 30 alteração do parâmetro IV

### Pergunta h)

DES (Data Encryption Standard) e 3DES (Triple DES) ambos sendo algoritmos de criptografia simétrica, diferem em termos de tamanho da chave e no número de rodadas realizadas durante a criptografia.

### Tamanho da Chave

- DES usa uma chave de 56 bits, que é considerada relativamente curta pelos padrões atuais.
- 3DES, aplica o DES três vezes em sequência com três chaves diferentes, resultando num tamanho de chave de 168 bits (usando três chaves de 56 bits). No entanto, devido ao tamanho efetivo da chave ser reduzido para 112 bits devido a ataques, é considerado proporcionar um nível de segurança aproximadamente equivalente ao de algoritmos de chave simétrica de 80 bits.

### Número de *rounds*:

- DES realiza 16 rodadas de criptografia.
- 3DES, dependendo do modo de operação, pode realizar 48 ou 16 rodadas de criptografia (se usado no modo de comprimento triplo ou no modo de encadeamento de bloco externo, respetivamente).

## Robustez

3DES é geralmente considerado mais robusto em comparação com o DES, principalmente devido ao seu tamanho de chave mais longo. No entanto, vale ressaltar que o 3DES foi descontinuado em favor de algoritmos de criptografia mais modernos como AES (Advanced Encryption Standard) devido à sua ineficiência computacional e suscetibilidade a certos ataques.

O principal motivo para usar o 3DES como uma alternativa ao DES tem a ver com o seu aumento no tamanho da chave, o que proporciona um nível mais alto de segurança em comparação com o DES.

Além disso, o 3DES foi desenvolvido como uma forma de estender a vida útil do DES e fornecer compatibilidade com sistemas existentes que já utilizavam o DES.

Em resumo, embora o 3DES ofereça uma robustez aumentada em comparação com o DES devido ao seu tamanho de chave mais longo, ele não é tão seguro ou eficiente quanto algoritmos de criptografia modernos como AES. Portanto, embora ainda possa ser usado em certos contextos, geralmente é recomendado usar AES ou outros algoritmos modernos para novas aplicações que exijam criptografia forte.

## Alínea 1.10

### Pergunta a)

No sistema Linux, as senhas são normalmente armazenadas em formato **hash** por motivos de segurança. Quando um utilizador cria ou altera sua senha, o sistema aplica uma função **hash** criptográfica unidirecional antes de armazená-la no arquivo `/etc/shadow`, que é legível apenas pelo usuário root.

A estrutura de uma entrada no arquivo `/etc/shadow` geralmente se parece com isso:

**`nome_de_utilizador:hash_da_senha:outros_campos`**

Onde:

- **`'nome_de_utilizador'`**: é o nome de login do utilizador.
- **`'hash_da_senha'`**: representação hash da senha do usuário.
- **`'outros_campos'`**: podem incluir informações adicionais, como horário da última alteração de senha, expiração da senha, etc.

As funções de hash mais comumente usadas para hashing de senha em sistemas Linux são:

1. **SHA-512**: algoritmo de hashing padrão usado pelas distribuições Linux modernas. Ele emprega o algoritmo de hash SHA-512 para gerar um valor hash de 512 bits.

Essas funções de hash são consideradas seguras para armazenar senhas, pois são resistentes a ataques de força bruta. Além disso, os sistemas Linux frequentemente usam técnicas como salting (adição de dados aleatórios à senha antes do hash) para aumentar ainda mais a segurança.

### Pergunta b)

Os arquivos **`/etc/passwd`** e **`/etc/shadow`** têm finalidades diferentes no tratamento da autenticação de usuários e gerenciamento de senhas:

1. **`/etc/passwd`**: armazena informações essenciais sobre contas de utilizadores. Cada linha no arquivo representa uma conta de utilizador e contém vários campos separados por dois pontos (`:`).

Esses campos geralmente incluem o nome do utilizador, senha criptografada (historicamente), ID do usuário, ID do grupo, nome completo do utilizador, diretoria *home* e *shell* padrão. No entanto, sistemas Linux modernos geralmente armazenam um caractere "x" ou um espaço reservado no campo de senha em vez da senha criptografada real. Isso ocorre porque armazenar senhas criptografadas em `/etc/passwd` representa um risco de segurança, já que o arquivo é legível por todos os usuários.

2. **`/etc/shadow`**: criado para abordar as preocupações de segurança ao armazenar senhas criptografadas em `/etc/passwd`. Contém informações da senha criptografada para contas do utilizador, entre outros dados relacionados com segurança. O campo de senha criptografada em `/etc/shadow` só é legível pelo utilizador *root*, fornecendo uma camada adicional de segurança. Portanto, é considerado um local mais seguro para armazenar hashes de senha.

Em suma, **`/etc/passwd`** existe principalmente para armazenar informações gerais de contas do utilizador, enquanto o **`/etc/shadow`** existe especificamente para armazenar de forma segura *hashes* de senha e outros dados sensíveis relacionados à segurança.

### Pergunta c)

Usando o Crackstation, para o utilizador *root* é possível de ver a seguinte informação

Password original: popcorn123!


Tipo de hash(ou método de hash): SHA1

### Free Password Hash Cracker

---

Enter up to 20 non-salted hashes, one per line:

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults



Não sou um robô

Privacidade - Termos de Utilização

Hash	Type	Result
f4244e539b46496d146fa1159a2a188aedd7295f	sha1	popcorn123!

**Color Codes:** Green Exact match, Yellow Partial match, Red Not found.

*Figura 31 crackstation para o utilizador root*

Para o utilizador *admin* é possível de ver a seguinte informação

Password original: password123!


Tipo de hash(ou método de hash): SHA1

### Free Password Hash Cracker

---

Enter up to 20 non-salted hashes, one per line:

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults



Não sou um robô

Privacidade - Termos de Utilização

Hash	Type	Result
adbbd3aa5619f2932733104eb8cee08f6fd2693	sha1	password123!

**Color Codes:** Green Exact match, Yellow Partial match, Red Not found.

*Figura 32 crackstation para o utilizador admin*

Para o utilizador *Joe* é possível de ver a seguinte informação

Password original: dogsname

Tipo de hash(ou método de hash): MD5

**Free Password Hash Cracker**

---

Enter up to 20 non-salted hashes, one per line:

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1\_bin)), QubesV3.1BackupDefaults

☐

Não sou um robô

reCAPTCHA

[Privacidade](#) - [Termos de Utilização](#)

Hash	Type	Result
906b8e437c1fd25c70efd94e78f5e23b	md5	dogsname

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

*Figura 33 crackstation para o utilizador joe*

Para o utilizador *Security* não foi possível encontrar a password original.

**Free Password Hash Cracker**

---

Enter up to 20 non-salted hashes, one per line:

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1\_bin)), QubesV3.1BackupDefaults

☐

Não sou um robô

reCAPTCHA

[Privacidade](#) - [Termos de Utilização](#)

Hash	Type	Result
aa1722b7818cf3a9ced224805f3ee5fd	Unknown	Not found.

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

*Figura 34 crackstation para o utilizador Security*

## Pergunta d)

Os hashes fornecidos não incluem *salt*. Geralmente é adicionado às senhas antes de serem ‘hasheadas’ para aumentar a segurança. Ele é usado para prevenir ataques de dicionário.

## Pergunta e)

Aumentar o número de rounds numa função de hash aumenta a sua segurança, tornando-a mais resistente a vários ataques criptográficos, incluindo ataques de força bruta e ataques de colisão. Com cada rodada adicional, a complexidade da computação aumenta, tornando mais difícil um atacante reverter ou encontrar vulnerabilidades no algoritmo.

No entanto, é importante observar que o SHA-1 é considerado criptograficamente comprometido e inseguro devido à descoberta de ataques teóricos que podem encontrar colisões (entradas diferentes produzindo o mesmo hash) mais rapidamente do que o tempo de força bruta esperado. Portanto, aumentar o número de rodadas para o SHA-1 não mitigaria suas vulnerabilidades de segurança fundamentais.

Para segurança mais forte, é recomendável usar funções de hash mais recentes como SHA-256, SHA-384 ou SHA-512, que oferecem tamanhos de hash maiores e não são vulneráveis aos mesmos ataques que afetam o SHA-1.

### Pergunta f)

Mudando um dos valores das passwords, não é possível recuperar a password original.

No entanto, pegando no exemplo do primeiro utilizador podemos usar fazer o seguinte:

- Password modificado: Password123
- Tipo de hash: MD5

Usando o cyberchef, obtemos a seguinte sequência

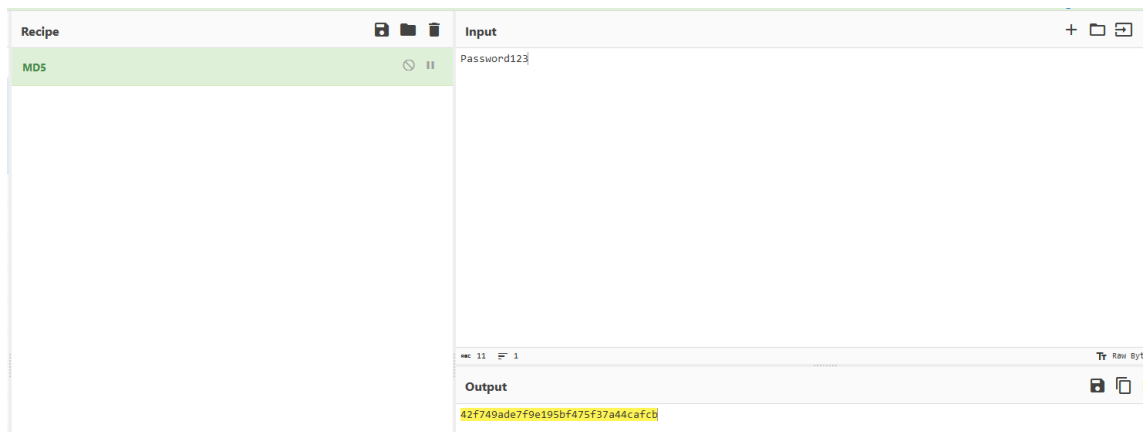


Figura 35 cyberchef com uma das passwords modificadas

Usando o crackstation, podemos ver que esta password também pode ser recuperada, mas como foi já dito, não é possível recuperar a original

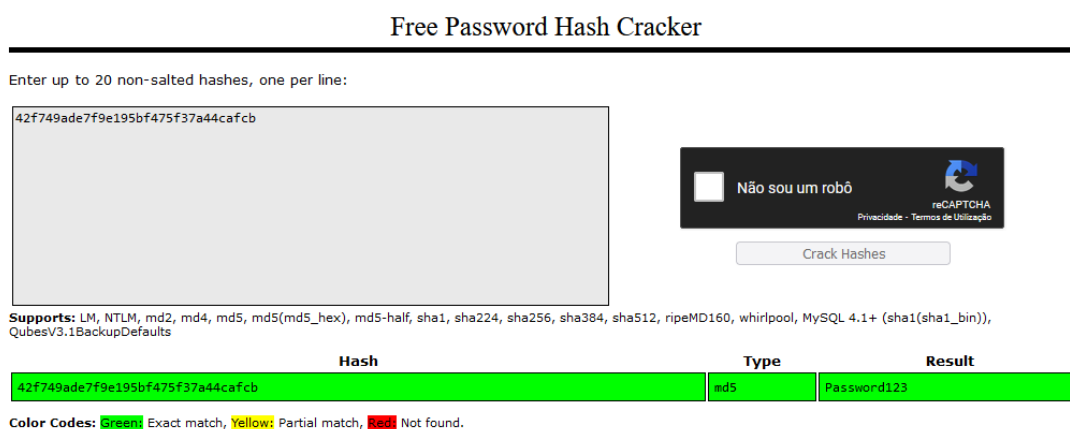


Figura 36 crackstation da password modificada

### Pergunta g)

Sim, há algumas práticas de segurança inadequadas evidentes nas informações

## Uso de Senhas Fracas

As senhas usadas para as contas ('dogsnome', 'password123!', 'popcorn123!') parecem ser relativamente fracas e facilmente adivinháveis. Senhas fortes e complexas são essenciais para prevenir acesso não autorizado às contas.

## Possível Falta de Políticas de Senha

As senhas escolhidas são simples e não atendem aos requisitos típicos de complexidade de senha (por exemplo, comprimento mínimo, inclusão de números, caracteres especiais, etc.).

## Uso de Algoritmos de Hash Inseguros

As senhas usam os algoritmos de hash SHA1 ('root' e 'admin') e MD5 ('Joe'). Embora o SHA1 seja considerado mais seguro que o MD5, ambos os algoritmos são vulneráveis a vários tipos de ataques, incluindo ataques de força bruta e colisão. É recomendável usar algoritmos de hash mais fortes, como SHA-512 ou bcrypt, para a hash das senhas.

## Possível Falta de Salt

Os hashes fornecidos não parecem incluir *salt*. O uso de *salt* é crucial para adicionar aleatoriedade às senhas, tornando-as mais resistentes a diferentes ataques. Sem *salt*, senhas idênticas produzem o mesmo hash, facilitando os atacantes identificar senhas comuns.

## Alínea adicional

Devido ao facto de o exercício 1.11 não conseguir perceber muito bem, pois não deu para entender o sucedido, o elemento do grupo vai demonstrar, em código usando *python* alguns dos algoritmos estudados neste trabalho prático.

Todo o código produzido pode ser encontrado na secção "Anexo".

Os algoritmos produzidos foram os seguintes:

- MD5,
- SHA1,
- SHA256,
- DES (encriptação e desencriptação),
- Triple DES(encriptação e desencriptação).

Na figura é possível de ver os resultados obtidos, para testes foi usada a *string* "Hello world" como parâmetro de mensagem.



```
Result Hash with md5: ed076287532e86365e841e92bfc50d8c
Result Hash with sha1: 2ef7bde608ce5404e97d5f042f95f89f1c232871
Result Hash with sha256: 7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284add200126d9069

Result Hash with md5 using base64: 7Qdih1MuhjZehB6Sv8UNJA==
Result Hash with sha1 using base64: Lve95gj0VATpfV8EL5X4nxwjKHE=
Result Hash with sha256 using base64: f40xZX/x/F05Lc6BSKHWXfwtSx+j1ncoSt3SABJtkGk=

DES Encrypted Message: d35264e8079ccaa968a23360c402c157
DES Decrypted Message: Hello World!

Triple DES Encrypted Message: d35264e8079ccaa968a23360c402c157
Triple DES Decrypted Message: Hello World!
```

*Figura 37 resultado obtido em python*

## Conclusões

De modo a recapitular todos os aspetos e conceitos aprendidos durante o desenvolvimento do projeto, irá ser feito um breve resumo dos diferentes conceitos.

A primeira abordagem consistiu em estudar alguns dos algoritmos que foram mencionados no enunciado do projeto.

Após isso, aplicar os conceitos através do uso de algumas ferramentas (como por exemplo o cyberchef, e o crackstation). Comparar os resultados, e aplicados outros tipos de algoritmos de hash, confirmar se ambos davam os mesmos resultados.

Todos os resultados obtidos foram satisfatórios quando aplicadas as diferentes técnicas.

No entanto, algumas dificuldades surgiram durante o processo deste trabalho prático.

Um das dificuldades foi durante o uso da ferramenta MD5 Online. O grupo não percebeu muito bem como usar esse tipo de ferramenta, talvez devido à falta de conhecimento ou não se sentir muito à vontade com o uso deste website.

Gostaríamos de destacar que caso houvesse mais tempo, poderíamos explorar a “fundo” alguns dos muitos algoritmos que existem, no que diz respeito à parte das linguagens de programação, usufruindo alguns dos conhecimentos da disciplina e aplicar um pouco programação na descoberta da área de redes.

No que podia ser melhorado no projeto, não vejo aquilo que possa ser melhorado, visto que o objetivo era estudar e conhecer muitos dos diferentes algoritmos estudados na disciplina, e usufruir das ferramentas que existem e como são aplicadas nesta área.

No entanto, apesar das dificuldades sentidas durante a realização do projeto, o grupo conseguiu realizar todos os objetivos deste projeto com sucesso, consolidando todos os diferentes conceitos e técnicas aprendidas durante a execução do projeto.

## Bibliografia

[1] Cyberchef: <https://gchq.github.io/CyberChef/>

[2] Crackstation: <https://crackstation.net/>

[3] dCode: <https://www.dcode.fr/en>

[4] MD5 online: <https://www.md5online.org/>

## Anexo

Nesta secção é possível de encontrar os exemplos feitos em *python*, do código criado para o estudo de alguns dos algoritmos estudados

```
import hashlib
import base64
import binascii
from pyDes import des, triple_des, PAD_PKCS5

def convert_message_to_hash(message, hash_function):
    global hash_object

    if hash_function == 'md5':
        hash_object = hashlib.md5()
    elif hash_function == 'sha1':
        hash_object = hashlib.sha1()
    elif hash_function == 'sha256':
        hash_object = hashlib.sha256()

    hash_object.update(message.encode())
    result_hash = hash_object.hexdigest()

    print("Result Hash with " + hash_function + ": ", result_hash)

def convert_message_to_hash_with_base64(message, hash_function):
    global hash_object
    if hash_function == 'md5':
        hash_object = hashlib.md5(message.encode()).digest()
    elif hash_function == 'sha1':
        hash_object = hashlib.sha1(message.encode()).digest()
    elif hash_function == 'sha256':
        hash_object = hashlib.sha256(message.encode()).digest()
    result_hash_base64 = base64.b64encode(hash_object)

    print("Result Hash with " + hash_function + " using base64: ",
          result_hash_base64.decode())

def DES_encryption(message, des_key, isToDEncrypt):
    des_cipher = des(des_key, padmode=PAD_PKCS5)
    encrypted_data_des = des_cipher.encrypt(message.encode())
    if isToDEncrypt:
        encrypted_data_des_hex = binascii.hexlify(encrypted_data_des)
        print('DES Encrypted Message: ',
              encrypted_data_des_hex.decode())
        return encrypted_data_des_hex.decode()
    else:
        decrypted_data_des = des_cipher.decrypt(encrypted_data_des)
        print('DES Decrypted Message: ', decrypted_data_des.decode())
        return decrypted_data_des.decode()

def triple_DES_encryption(message, des_key, isToDEncrypt):
    triple_des_cipher = triple_des(des_key, padmode=PAD_PKCS5)
    encrypted_data_triple_des =
    triple_des_cipher.encrypt(message.encode())
    if isToDEncrypt:
        encrypted_data_triple_des_hex =
```

```
binascii.hexlify(encrypted_data_triple_des)
    print('Triple DES Encrypted Message: ',
encrypted_data_triple_des_hex.decode())
    return encrypted_data_triple_des
else:
    decrypted_data_triple_des =
triple_des_cipher.decrypt(encrypted_data_triple_des)
    print('Triple DES Decrypted Message: ',
decrypted_data_triple_des.decode())
    return decrypted_data_triple_des

message = "Hello World!"
des_key = b"12345678"
triple_des_key = b"123456781234567812345678"

convert_message_to_hash(message, 'md5')
convert_message_to_hash(message, 'sha1')
convert_message_to_hash(message, 'sha256')

print("")
convert_message_to_hash_with_base64(message, 'md5')
convert_message_to_hash_with_base64(message, 'sha1')
convert_message_to_hash_with_base64(message, 'sha256')
print("")

DES_encryption(message, des_key, True)
DES_encryption(message, des_key, False)
print("")

triple_DES_encryption(message, triple_des_key, True)
triple_DES_encryption(message, triple_des_key, False)
```