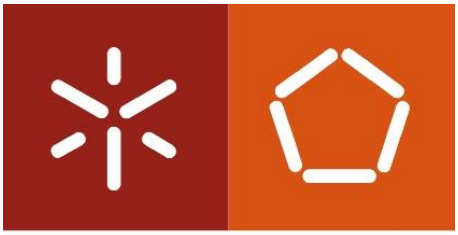

Universidade do Minho

Departamento de Informática

Mestrado em Engenharia Informática

Criptografia e Segurança de Informação

Engenharia de Segurança
Ficha Prática 3

Grupo Nº 3

Ariana Lousada (PG47034) Luís Carneiro (PG46541)
Rui Cardoso (PG42849)

29 de março de 2022

Capítulo 1

Parte V: Funções de sentido único

1.1 Pergunta P.V.1.1

1. Altere o programa que fez, de modo a permitir que o utilizador forneça uma chave com o tamanho que considerar adequado, e utilize uma função KDF para amplificar a entropia da chave que o utilizador lhe forneceu. Justifique as opções tomadas.

O código-fonte do programa, assim como a explicação de como o executar, encontra-se na área do nosso grupo no Github.

A função de derivação da chave usada foi o *scrypt*, tal como aconselhado na documentação do PyCryptodome, para novas aplicações e protocolos. Para além de ser pesada computacionalmente, é também exigente da memória e, como tal, mais segura contra ASICs. Estes são circuitos de chip integrados que são feitos para uma tarefa em específico e são mais eficientes que computadores normais (neste caso, feitos especificamente para inverter a função de derivação).

Os parâmetros usados no protocolo são a chave dada, o *salt*, o *key_len*, o *N*, o *r*, o *p* e o *num_keys*. Estes são, respetivamente, a chave a obfuscar, uma string usada para proteger contra ataques de dicionários que randomiza o output para a mesma chave, o tamanho de saída da(s) chave(s) derivada(s), o CPU/Custo de memória, o tamanho do bloco, o nível de paralelização e o número de chaves a gerar.

Na documentação são sugeridos valores recomendados para duas situações: logins interativos ou cifrar ficheiros. Foram escolhidos os referentes aos logins interativos, pois a diferença entre a chave dada pelo utilizador e o típico input de password não é grande; no caso dos ficheiros, o input esperado já é bem maior. Ou seja, o N é 2^{14}, o r é 8 e o p é 1.

O *salt* é gerado aleatoriamente para cifrar o ficheiro. Já para o decifrar, o *salt* é dado, de modo a, dada a chave original que o utilizador forneceu, chegar à mesma chave usada que cifrou, de facto, o ficheiro. O *key_len* é 32, visto que a chave usada pelo ChaCha20 para cifrar o conteúdo do ficheiro inicial tem de ter 32 bytes. Finalmente, o *num_keys* é 1, visto que só precisamos de derivar uma chave a partir da chave dada.

1.2 Pergunta P.V.1.2

1. Explique porque não deve utilizar uma função de hash normal para guardar a hash de uma password.

Existe vários métodos de utilizar funções de hash para guardar passwords.

Algumas funções utilizadas anteriormente, mas que já não são aconselhadas são MD5(RSA Labs - Donald Rivest) pois já foram encontradas colisões, isto é, passwords diferentes produzem a mesma hash, SHA-1 tem sido alvo de avanços recentes significativos também é desaconselhada.

Outras funções de hash utilizadas actualmente consideradas seguras são SHA-2 (2001 - NSA) e SHA-3(2015 - Keccak).

Apesar de estas funções serem seguras,quando ocorre uma *breach* de uma base de dados de hashed passwords um atacante através de *bruteforce* e aproveitando o facto de utilizadores utilizarem password comuns e inseguras, pode tentar descobrir a password como é exemplificado na pergunta seguinte.

Uma opção para ajudar a combater isto é a utilização de *Salted Hash*, isto acrescenta um valor único ao fim de cada password antes de esta passar pela função de hash.

Como por exemplo no seguinte caso:

Utilizador1 tem a password 12345

Utilizador2 tem a password 12345

Se usarmos uma função *unsalted hash* nas passwords destes utilizadores o resultado vai ser o mesmo. No entanto se utilizarmos uma *salted hash* no fim da password vai ser acrescentado um valor único,por exemplo 12345+Q59f94g04fQx e 12345+R69b94Q63sr3 isto vai ter resultados diferentes após passar pela função de hash apesar de as passwords serem as mesmas.

2. Foi publicado na internet um ficheiro de passwords de acesso a um serviço online, tendo sido referido que a aplicação de guarda de passwords desse serviço utiliza o SHA256 e guarda essa representação das passwords em hexadecimal. Ou seja, a password do utilizador é guardada do seguinte modo: hex(SHA256(password)). Sabendo que a passsword representada por 96cae35ce8a9b0244178bf28e4966c2ce1b8385723a96a6b838858cdd6ca0a1e faz parte do top200 das passwords mais comuns https://nordpass.com/most-common-passwords-list/, indique qual é essa password, e explique os passos que deu para a encontrar, assim como o código que desenvolveu.

Através da ajuda do Notepad++ para formatar o texto, foi extraída a lista do top200 de passwords mais utilizadas segundo o website https://nordpass.com/most-common-passwords-list/ .

O programa feito para descobrir a password foi feito em Python com a importação da livraria *hashlib* para converter passwords em hashed passwords utilizando SHA256.

Foi criada uma lista de strings com as 200 passwords mais usadas extraídas anteriormente com o nome de *'top200'*.

Criou-se ainda uma outra variável com o nome *'hashedPasswordToCrack'* com a string da hash da password que se pretende descobrir, neste caso *'96cae35ce8a9b0244178bf28e4966c2ce1b8385723a96a6b838858cdd6ca0a1e'*.

Por último foi usado um ciclo *for* que percorre todas as passwords do top200 utiliza a livraria *'hashlib'* para converter a password em hash, compara esta com a *hashedPasswordToCrack*, caso sejam iguais então a password foi descoberta.

Nesta caso a password usada foi '123123'.

Na seguinte imagem pode ser visto o código todo utilizado.

```python
from hashlib import sha256 #Libraria para converter em HASH utilizando SHA256
#Lista do Top200 de passwords mais usadas
top200 = ['123456','123456789','picture1','password','12345678','111111','123123','12345',
'1234567890','senha','1234567','qwerty','abc123','Million2','000000','1234','iloveyou',
'aaron431','password1','qqww1122','123','omgpop','123321','654321','qwertyuiop','qwer123456',
'123456a','a123456','666666','asdfghjkl','ashley','987654321','unknown','zxcvbnm','112233',
'chatbooks','20100728','123123123','princess','jacket025','evite','123abc','123qwe','sunshine',
'121212','dragon','1q2w3e4r','5201314','159753','123456789','pokemon','qwerty123','Bangbang123',
'jobandtalent','monkey','1qaz2wsx','abcd1234','default','aaaaaa','soccer','123654','ohmnamah23',
'12345678910','zing','shadow','102030','11111111','asdfgh','147258369','qazwsx','qwe123',
'michael','football','baseball','1q2w3e4r5t','party','daniel','asdasd','222222','myspace1',
'asd123','555555','a123456789','888888','7777777','fuckyou','1234qwer','superman','147258',
'999999','159357','love123','tigger','purple','samantha','charlie','babygirl','88888888',
'jordan23','789456123','jordan','anhyeuem','killer','basketball','michelle','1q2w3e','lol123',
'qwerty1','789456','6655321','nicole','naruto','master','chocolate','maggie','computer','hannah',
'jessica','123456789a','password123','hunter','686584','iloveyou1','0987654321','justin','cookie',
'hello','blink182','andrew','25251325','love','987654','bailey','princess1','0123456','101010',
'12341234','a801016','1111','1111111','anthony','yugioh','fuckyou1','amanda','asdf1234','trustno1',
'butterfly','x4ivygA51F','iloveu','batman','starwars','summer','michael1','00000000','lovely',
'jakcgt333','buster','jennifer','babygirl1','family','456789','azerty','andrea','q1w2e3r4',
'qwer1234','hello123','10203','matthew','pepper','12345a','letmein','joshua','131313','123456b',
'madison','Sample123','777777','football1','jesus1','taylor','b123456','whatever','welcome',
'ginger','flower','333333','1111111111','robert','samsung','a12345','loveme','gabriel','alexander',
'cheese','passw0rd','142536','peanut','11223344','thomas','angel1']
#Hashed password que se pretende descobrir
hashedPasswordToCrack = '96cae35ce8a9b0244178bf28e4966c2ce1b8385723a96a6b838858cdd6ca0a1e'
#Ciclo para comparar as hash
for password in top200:
    if(sha256(password.encode('utf-8')).hexdigest()==hashedPasswordToCrack):
        print(password)
        break
```

Figura 1.1: Código para descobrir a hashed password.

E o resultado do mesmo.

```
123123
[Finished in 97ms]
```

Figura 1.2: Resultado do código.

1.3 Pergunta P.V.1.3

1. Utilizando o openssl indique qual é o comando linha que tem de utilizar para obter o HMAC-SHA1 de todos os ficheiros numa diretoria.

De modo a obter o HMAC-SHA1 para um determinado conjunto de ficheiros o comando `openssl` a utilizar é o seguinte:

```
openssl dgst -sha1 -hmac chave_para_o_hmac *
```

```
arrow@arrowVM:~/Documents/ES$ ls
file2.txt  file.txt  hello.txt
arrow@arrowVM:~/Documents/ES$ openssl dgst -sha1 -hmac 'dsadsa' *
HMAC-SHA1(file2.txt)= 448ac09ece1d2e4c8e1b91120af587c2575b5aaa
HMAC-SHA1(file.txt)= 9f31328cc5b0d110c56fe3369109684f54206221
HMAC-SHA1(hello.txt)= 08879790cd827b3e9ae245d1f946f67fe103c314
arrow@arrowVM:~/Documents/ES$
```

Figura 1.3: Exemplo de aplicação do comando numa diretoria

2. O que teria de fazer para saber se (e quais) os ficheiros foram alterados, desde a última vez que efetuou a operação indicada no ponto anterior?

Caso o utilizador que executou o comando mencionado na alínea anterior tenha guardado o valor do HMAC-SHA1 do ficheiro, este pode simplesmente comparar e verificar se este se mantém.

```
arrow@arrowVM:~/Documents/ES$ openssl dgst -sha1 -hmac 'dsadsa' *
HMAC-SHA1(file2.txt)= 3b8bbde022f948e6331125553f32489b135445be
HMAC-SHA1(file.txt)= 9f31328cc5b0d110c56fe3369109684f54206221
HMAC-SHA1(hello.txt)= 08879790cd827b3e9ae245d1f946f67fe103c314
arrow@arrowVM:~/Documents/ES$ cat hello.txt
hellothere
arrow@arrowVM:~/Documents/ES$ nvim hello.txt
arrow@arrowVM:~/Documents/ES$ cat hello.txt
hello
arrow@arrowVM:~/Documents/ES$ openssl dgst -sha1 -hmac 'dsadsa' *
HMAC-SHA1(file2.txt)= 3b8bbde022f948e6331125553f32489b135445be
HMAC-SHA1(file.txt)= 9f31328cc5b0d110c56fe3369109684f54206221
HMAC-SHA1(hello.txt)= 0e7d6dc4d55324f7dc749d3f50a0767a9976a7b2
arrow@arrowVM:~/Documents/ES$
```

Figura 1.4: Exemplo de alteração de ficheiro

Através da figura, é possível detetar diferenças entre o valor HMAC-SHA1 do ficheiro `hello.txt` antes e após a sua modificação.

Em suma, é possível detetar modificações de ficheiros comparando os seus valores do HMAC-SHA1 ao longo do tempo.

Capítulo 2

Parte VI: Acordo de chaves

2.1 Pergunta P.VI.1.1

Desenvolva, na linguagem que preferir e utilizando uma biblioteca criptográfica à sua escolha, um programa linha de comando que permita visualizar o acordo de chave entre a Alice e o Bob, utilizando o protocolo Diffie-Hellman, assim como a posterior comunicação de mensagens cifradas (pela chave acordada) entre esses mesmos dois intervenientes.

```
import random

class DHKE:
    def __init__(self,G,P):
        self.G_param = G
        self.P_param = P

    def generate_privatekey(self):
        self.pk = random.randrange(start = 1,stop = 10,step = 1)

    def generate_publickey(self):
        self.pub_key = pow(self.G_param,self.pk) % self.P_param

    def exchange_key(self,other_public):
        self.share_key = pow(other_public,self.pk) % self.P_param


#Simulacao da troca de chaves entre duas entidades. Considerando as duas entidades Bob e
     Alice.

Alice = DHKE(5,22)
Bob = DHKE(5,22)

Alice.generate_privatekey()
Bob.generate_privatekey()

print("------------Private Keys------------------\n")
print("Alice Private Key Generated is ",Alice.pk,"\n")

print("Bob Private Key Generated is ",Bob.pk,"\n")
print("------------End of Private Keys------------\n\n")


Alice.generate_publickey()
Bob.generate_publickey()

print("------------Public Keys------------------\n")
print("Alice Public Key Generated is ",Alice.pub_key,'\n')

print("Bob Public Key Generated is ",Bob.pub_key,'\n')
print("------------End of Public Keys-----------\n\n")

#Troca de chaves entre a Alice e o Bob

Alice.exchange_key(Bob.pub_key)
Bob.exchange_key(Alice.pub_key)
```

```
46
47 print("------------Shared Key Derieved-----------------\n")
48 print("Shared Key Generated now by Alice : ",Alice.share_key,'\n')
49
50 print("Shared Key Generated now by Bob : ",Bob.share_key,'\n')
51 print("------------End of Shared Key Derieved-----------\n")
```

No *script* definido anteriormente a lógica do protocolo Diffie-Hellman encontra-se implementada na classe DHKE.

Em primeiro lugar, os parâmetros globais são inicializados para cada entidade(G=5 e P=23). De seguida a Alice e o Bob geram cada um a respetiva chave privada, onde cada um escolhe um número arbitrário entre 1 e 10. As entidades prosseguem então a gerar as suas chaves públicas($G^{privatekey} * mod\ p$). De seguida trocam as chaves públicas entre si através do método `exchange_key()`. Por fim a chave comum é gerada e imprimida n ecrã do utilizador.

Exemplo de output:

```
------------Private Keys-----------------

Alice Private Key Generated is  8

Bob Private Key Generated is  4

------------End of Private Keys------------


------------Public Keys-----------------

Alice Public Key Generated is  15

Bob Public Key Generated is  9

------------End of Public Keys-----------


------------Shared Key Derieved-----------------

Shared Key Generated now by Alice :  3

Shared Key Generated now by Bob :  3

------------End of Shared Key Derieved-----------
```

Figura 2.1: Output gerado pelo *script* desenvolvido.

Capítulo 3

Referências

- P.V.1.1
  - https://pycryptodome.readthedocs.io/en/latest/src/protocol/kdf.html
  - https://www.investopedia.com/terms/a/asic.asp
- P.V.1.3
  - https://helpmanual.io/man1/openssl-dgst-ssl/
  - https://osxdaily.com/2012/02/09/verify-sha1-hash-with-openssl/
  - https://pascua.iit.comillas.edu/palacios/seguridad/openssl.pdf
- P.VI.1.1
  - https://cryptobook.nakov.com/key-exchange/diffie-hellman-key-exchange