Universidade do Minho

Mestrado em Engenharia Informática

Engenharia de segurança

Ficha de exercício 2

Grupo 1

Rui Carlos Azevedo Carvalho - PG47633
Daniel Barbosa Miranda - PG47123
Ana Luísa Lira Tomé Carneiro - PG46983

IV.1 *Stream cipher* - Pergunta P1.1

O problema 1 consistia em criar um programa que conseguisse cifra e decifrar um ficheiro de texto utilizando a cifra simétrica Chacha20. Esta *stream cipher* consiste em "aproximar" a cifra *One-Time-Pad* por intermédio de um gerador de chaves (que produz uma sequência de chave a partir de uma chave de comprimento 256 bits) e é muito utilizada para a cifragem dos novos protocolos de transporte, nomeadamente o TLS 1.3.

A implementação do programa foi realizado em Python e utilizou a biblioteca *PyCryptodome* para a implementação da cifra Chacha20. A implementação inicia-se com a apresentação de um menu ao utilizador, para que este consiga escolher se pretende cifrar ou decifrar o ficheiro.

```
# Opções do menu do utilizador
menu = {
    0: 'SAIR',
    1: 'Cifragem',
    2: 'Decifragem',}

# Menu para determinar se o utilizador pertende
# cifrar ou  decifrar uma mensagem
while True:
    for key in menu.keys():
        print(key, '--', menu[key] )
    option = int(input('=>  '))
    if option == 1:
        encrypt()
    elif option == 2:
        decrypt()
    elif option == 0:
        exit()
    else:
        print('Opção Inválida.')
```

Caso o utilizador pretenda cifrar um ficheiro é chamada a função *encrypt* que irá perguntar ao utilizador qual o ficheiro a cifrar, chave a ser utilizada pela cifra ChaCha20 e o nome do ficheiro cifrado de output.

É de notar que a chave que realmente será utilizada nestes processos terá um tamanho de 32 bytes (256 bits) e será obtida a partir do *input* do utilizador. Para isso, caso a chave de input não tiver 32 bytes então o programa vai aumentar o compri-

mento, duplicando os valores da chaves. Desta forma se o input do utilizador for 12345, a chave que será utilizada pelo a cifra será 12345123451234512345123451234512.

```
#FUNÇÃO: Cifra ficheiro
def encrypt():

    #ficheiro a cifrar
    doc = input('Ficheiro a cifrar:  ')
    #chave a ser utilizada na cifra
    cipherKey = input('Chave:  ').encode('utf-8')
    #nome do ficheiro cifrado
    fileName = input('Ficheiro de output:  ')

    #retira a mensagem do ficheiro a cifrar
    f = open(doc, "r")
    msg = f.read()
    f.close()

    print("MENSAGEM")
    print(msg)

    #gera chave de 32 bytes utilizando o input do utilizador
    n = len(cipherKey)
    if n < 32:
        l = 32/n
        for i in range(0,int(l)):
            cipherKey += cipherKey[0:n]
    key = [cipherKey[i:i+32] for i in range(0, len(cipherKey),32)]

    #cria ficheiro cifrado
    ciphertext = buildCiphertext(msg,key[0])

    print("CIPHERTEXT - ** Armazenado no Ficheiro " + fileName + " **")
    print(json.dumps(ciphertext))

    #escreve a mensagem cifrada em bytes no ficheiro com nome igual à variavel fi
    output = open(fileName, "w")
    output.write(json.dumps(ciphertext))
    output.close()
```

Como forma de criar o ficheiro cifrado é utilizada a função *buildCiphertext* que recebe o ficheiro e a chave a ser utilizada no processo de cifra. Nesta função é gerada um valor *nonce* de 12 bytes a ser utilizado no algoritmo ChaCha20 e é criado um pacote representativo todo o ficheiro cifrado que contém o texto cifrado e o valor do *nonce*.

```
#FUNÇÃO: Cria o ciphertext através da cifra Chacha20
def buildCiphertext(msg,key):
```

```
    #cria o nonce de 12 bytes de forma pseudo-aletória
    nonce = get_random_bytes(12)

    #cria a cifra Chacha20 utilizando a chave e o nonce
    cipher = ChaCha20.new(key=key, nonce=nonce)
    #cifra a mensagem
    ciphertext = cipher.encrypt(dumps(msg))

    #transforma em bytes a mensagem, nonce e o salt para geração da chave
    ct = b64encode(ciphertext).decode('utf-8')
    nc = b64encode(nonce).decode('utf-8')

    #mensagem cifrada - ciphertext
    pkg = {'ciphertext' : ct, 'nonce': nc}

    return pkg
```

Caso o utilizador pretenda decifrar um ficheiro é chamada a função *decrypt* que irá perguntar ao utilizador qual o ficheiro a decifrar, chave a ser utilizada pela cifra ChaCha20 e o ficheiro de *output* com o *plaintext*.

É de notar que a chave que realmente será utilizada neste processo terá um tamanho de 32 bytes (256 bits) e será obtida a partir do *input* do utilizador, tal como já foi mencionado.

```
#Função: Decifra um ficheiro
def decrypt():

    #ficheiro a decifrar
    doc = input('Ficheiro a decifrar:  ')
    #chave a ser utilizada na decifra
    cipherKey = input('Chave:  ').encode('utf-8')
    #nome do ficheiro com a mensagem decifrada
    fileName = input('Ficheiro de output:  ')

    #retira bytes da mensagem do ficheiro a dcifrar
    f = open(doc, "rb")
    msg = f.read()

    print("MENSAGEM")
    print(msg)

    #gera a chave de 32 bytes utilizando o input do utilizador
    ciphertext = json.loads(msg)
    n = len(cipherKey)
    if n < 32:
        l = 32/n
        for i in range(0,int(l)):
            cipherKey += cipherKey[0:n]
```

```
    key = [cipherKey[i:i+32] for i in range(0, len(cipherKey),32)]

    #cria a mensagem decifrada
    plaintext = buildPlaintext(ciphertext,key[0])

    print("PLAINTEXT - ** Armazenado no Ficheiro " + fileName + " **")
    print(loads(plaintext))

    #escreve no ficheiro a mensagem cifrada
    output = open(fileName, "w")
    output.write(loads(plaintext))
```

Como forma de criar o ficheiro decifrado é utilizada a função *buildPlaintext* que recebe o ficheiro e a chave a ser utilizada no processo de decifra. Nesta função é retirado do ficheiro o valor *nonce* de 12 bytes que foi gerado no processo de cifra para que o algoritmo ChaCha20 tenha os mesmos parâmetros que na cifragem, conseguindo-se assim decifrar o ficheiro corretamente.

```
#FUNÇÃO: Cria o plaintext através da cifra ChaCha20
def buildPlaintext(msg,key):

    #retira o valor nonce da mensagem cifrada
    nonce = b64decode(msg['nonce'])
    #retira a mensagem cifrada do ciphertext
    ciphertext = b64decode(msg['ciphertext'])

    #cria a decifra Chacha20 utilizando a chave e o non
    cipher = ChaCha20.new(key=key, nonce=nonce)
    #decifra mensagem
    plaintext = cipher.decrypt(ciphertext)

    return plaintext
```

Finalmente, a cifra ChaCha20 desenvolvida em Python encontra-se implementada na íntegra no seguinte link do repositório do Github. Para conseguir correr o programa pode-se utilizar o ficheiro *.txt* para cifra e posterior decifra presente no link

IV.2 *Block cipher* - Pergunta P2.1

Implementação

A classe implementada utiliza o algoritmo AES no modo GCM. Desta forma, definem-se duas constantes, uma com o número de bits da tag (128 bits) e a outra com o tamanhoe em bytes do IV (12 bytes).

```
1 private static final int TAG_SIZE = 128; //Tamanho em bits da tag
2 private static final int IV_SIZE = 12; //Tamanho em bytes do IV
```

Posteriormente define-se dois métodos que realizam as funcionalidades de cifra e decifra que o programa permite executar.

De facto, para efetuar a operação de cifra, é necessário converter a chave indicada pelo utilizador para um objeto do tipo "SecretKey", para poder ser compatível com a utilização do algoritmo AES:

```
1 byte[] keyBytes = key.getBytes(StandardCharsets.UTF_8);
2 SecretKey sk = new SecretKeySpec(keyBytes, 0, keyBytes.length, "AES");
```

Posteriormente gera-se o IV através de um objeto do tipo "SecureRandom", que utiliza um gerador pseudo aleatório mais seguro que o objeto "Random" que é mais comum de ser utilizador. Este IV possui então 12 bytes, sendo criado um objeto "GCMParameterSpec" que agrega os parametros da cifra (tamanho da tag e IV).

```
1 SecureRandom sr = new SecureRandom();
2 sr.nextBytes(iv);
3 GCMParameterSpec params = new GCMParameterSpec(TAG_SIZE, iv);
```

De seguida cria-se o objeto da cifra que vai ser utilizador. Neste caso é então o algoritimo AES no modo GCM sem a utilização de padding, já que no modo GCM não existe a necessidade de o utilizar.

```
1 Cipher AES_128_Cipher = Cipher.getInstance("AES/GCM/NOPADDING");
2 AES_128_Cipher.init(Cipher.ENCRYPT_MODE, sk, params);
```

Por fim realiza-se a leitura do ficheiro de input através de uma "CipherInputStream" que lê para um buffer de bytes com 512 de tamanho e aplica de imediato a cifra inicializada anteriormente. Utiliza-se um buffer para não carregar o ficheiro de uma vez só para a memória. Este buffer possui uma capacidade máxima de 512 já que com o modo GCM não existe a necessidade de ler um tamanho igual ao tamanho do bloco, que neste caso seria 128 bits, ao contrário de outros modos.

```
CipherInputStream in = new CipherInputStream(new
    ↪ FileInputStream(input), AES_128_Cipher);

while((readBytes = in.read(buffer)) != -1){
    out.write(buffer, 0, readBytes);
    out.flush();
}
```

Para a função de decifrar o *ciphertext*, realiza-se um processo semelhante, só que desta vês utiliza-se o IV criado pelo processo de cifra. É feita de igual forma a transformação da chave indicada pelo utilizador num objeto do tipo "SecretKey" e inicializada a mesma cifra.

```
GCMParameterSpec params = new GCMParameterSpec(TAG_SIZE, iv);

//Criacao da chave secreta a partir da chave fornecida pelo
    ↪ utilizador
byte[] keyBytes = key.getBytes(StandardCharsets.UTF_8);
SecretKey sk = new SecretKeySpec(keyBytes, 0, keyBytes.length, "AES");

Cipher AES_128_Cipher = Cipher.getInstance("AES/GCM/NoPadding");

AES_128_Cipher.init(Cipher.DECRYPT_MODE, sk, params);
```

O processo de decifragem, utiliza uma "CipherInputStream" para ler blocos do *ciphertext* para um buffer e decifra-os com o objeto que contém a cifra e que foi inicializado anteriormente. O resutaldo é escrito no ficheiro de output através de uma "BufferedOutputStream", que utiliza um buffer antes de escrever para ficheiro, de modo a evitar escritas constantes no disco.

```
CipherInputStream in = new CipherInputStream(new
    ↪ FileInputStream(input), AES_128_Cipher);
int readBytes;

while((readBytes = in.read(buffer)) != -1){
    out.write(buffer, 0, readBytes);
    out.flush();
}
```

A função principal do programa começa por verificar a validade do número de argumentos passados como input na linha de comandos. A chave de cifra é o último argumento e caso não exista é necessário pedi-la ao utilizador.

```
//A chave e o ultimo argumento
key = args[args.length - 1];

//Caso a chave nao tenha sido introduzida pelo utilizador o programa
    ↪ pede-lhe neste passo
```

```
5  if((args.length == 4 && opt.equals("CIFRA")) || (args.length == 4 &&
   ↪  opt.equals("DECIFRA"))){
6      System.out.println("Introduza a chave: ");
7      key = sc.nextLine();
8  }
```

Caso a opção escolhida pelo utilizador seja de cifra este começa por utilizar o método indicado anteriormente que trata deste processo. Caso o IV que este devolve seja nulo, ou seja, a operação falhou por algum motivo, é imprimida uma mensagem de erro no ecrã. Caso contrário é indicado o sucesso da operação e os ficheiros onde foi escrito o *ciphertext* e o IV. Obviamente, estes são passados pelo utilizador. Para escrever o IV no ficheiro utiliza-se um método auxiliar que utiliza uma "FileOutputStream" para realizar a escrita.

```
1  if(opt.equals("CIFRA")){
2      //Caso a operacao seja de cifra chama o metodo auxiliar que trata
       ↪  desse processo
3      iv = encipherHandler(key, args[1], args[2]);
4
5      //Caso o resultado nao seja nulo escreve o IV no ficheiro
6      //Caso contrario imprime uma mensagem de erro.
7      if(iv != null){
8          writeIV(args[3], iv);
9          System.out.println("O Ficheiro foi cifrado. O Resultado está no
           ↪  ficheiro: " + args[2]);
10         System.out.println("O IV foi escrito no ficheiro: " + args[3]);
11     }else
12         System.out.println("A operação falhou!");
```

Caso a opção escolhida pelo utilizador seja a de decifrar, é carregado o IV presente no ficheiro indicado pelo utilizador para um array de bytes, através do método auxiliar readIV. Se esta operação for um sucesso então é chamado o método explicado anteriormente que realiza o processo de decifragem do *ciphertext*. Se este falhar então é escrita uma mensagem de erro no ecrã, caso contrário o ficheiro que foi alterado (ficheiro de output indicado pelo utilizador).

```
1  } else if (opt.equals("DECIFRA")){
2      //Caso a operacao seja de decifra carrega o IV para um array de
       ↪  bytes
3      //a partir do ficheiro indicado
4      byte [] ivBytes = readIV(args[3]);
5
6      //Caso o array nao seja nulo e possua os 12 bytes
7      //chama o metodo que realiza o processo de decifragem
8      if (ivBytes != null && ivBytes.length == IV_SIZE){
9          success = decipherHandler(key, args[1], args[2], ivBytes);
10
11         if(success)
```

```
12              System.out.println("O Ficheiro foi decifrado. O Resultado
                ↪  está no ficheiro: " + args[2]);
13          else
14              System.out.println("A operação falhou!");
15      } else
16          System.out.println("O IV tem de ter 12 bytes!");
17      }
```

Forma de utilização

Para utilizar este programa pode ser necessário compilá-lo primeiro. Este encontra-se na diretoria "Grupo1/Pratica 1/TP2/PerguntaP.IV.2.1/AES_GCM_128".

Seguem-se os comandos para realizar tanto a operação de cifra ou decifra de um ficheiro indicado pelo utilizador:

java AES_GCM_128 cifra [IF] [OF] [IV] [KEY]

java AES_GCM_128 decifra [FI] [OF] [IV] [KEY]

- FI: Ficheiro de input
- OF: Ficheiro de output
- IV: Ficheiro onde o IV será escrito/lido
- KEY: Chave que se pretende utilizar (16 bytes)

De facto, se se pretender decifrar um determinado texto deve-se utilizar a mesma chave utilizada no processo de cifra e o IV que foi gerado nesta operação.

Exemplos de utilização

De seguida é demonstrado um exemplo de utilização do programa. Este começa aplicar o comando de cifra a um ficheiro de input (in.txt), passando-lhe como argumentos o ficheiro de output (out.txt), o ficheiro onde está presente o IV (iv.txt) e a chave utilizada para cifrá-lo.

```
[rui:rui-Lenovo-Y520-15IKBN | ~/Universidade/4ano/2sem/CSI/ES/Grupo1/Pratica 1/TP2/PerguntaP.IV.2.1] ↝java AES_GCM_128 cifra in.txt out.txt iv.txt aaaabbbbaabb1234
O Ficheiro foi cifrado. O Resultado está no ficheiro: out.txt
O IV foi escrito no ficheiro: iv.txt
```

Figure 1: Utilização do comando de cifra

O conteúdo do ficheiro de entrada é o seguinte:

```
1 Teste que verifica se o programa executa corretamente.
```

Figure 2: Ficheiro de teste a ser cifrado

Após realização desta operação o ficheiro indicado pelo utilizador para armazenar o IV fica com o seguinte valor:

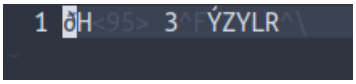

Figure 3: IV gerado pela funcionalidade de cifra

O ficheiro de output (ciphertext) é o seguinte:

Figure 4: Output gerado pela funcionalidade de cifra

De seguida utiliza-se o comando que realiza o processo inverso, ou seja, a decifra com o ficheiro de input o de output do comando anterior. Já o ficheiro de output é outro (out2.txt) e o ficheiro de IV é o mesmo, para utilizar o mesmo IV gerado pela Cifra. A chave que é passada também é a mesma utilizada pela operação de cifra:

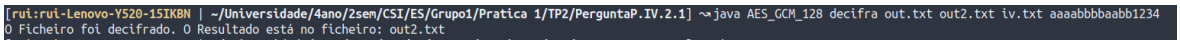

Figure 5: Utilização do comando que decifra o ficheiro

Assim o ficheiro que se obtém possui o seguinte conteúdo, pelo que é possível notar que se volta a obter o texto inicial:

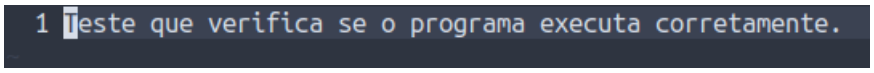

Figure 6: Output gerado pela funcionalidade de decifra