E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Segurança em Redes e Computadores

CÓDIGO: 21181

DOCENTE: Henrique S. Mamede

NOME: Hélio Emanuel Soares de Sousa

N.º DE ESTUDANTE: 2000027

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 2 de Novembro de 2020

TRABALHO / RESOLUÇÃO:

O trabalho foi desenvolvido no software Apache Netbeans IDE versão 12.0 na linguagem de programação Java.

Para a **cifra simétrica** optei pela definida na secção 3.3 em (Stallings, 2017) designada por Cifra de Transposição (permutações de colunas) em inglês Columnar Transposition Cipher.

Exemplo de aplicação da cifra contido em (Stallings, 2017) na página 107:

```
Key:          4 3 1 2 5 6 7
Plaintext:    a t t a c k p
              o s t p o n e
              d u n t i l t
              w o a m x y z
Ciphertext:   TTNAAPTMTSUOAODWCOIXKNLYPETZ
```

A implementação desta cifra foi realizada tendo por base os conceitos obtidos em (Stallings, 2017) e a implementação foi realizada tendo como base de partida os seguintes autores (HouseEducation, 2019), (Mathispower4u, 2013), (Pixel, 2019), (QuickCS, 2018), (VoxelPixel, 2019).

Funcionamento do programa do ponto de vista do utilizador:

Inicia com um menu de 3 opções, encriptar, desencriptar e sair. Existe um ciclo que corre enquanto não for selecionada a opção sair, se for introduzido texto durante o menu de opções o programa sai.

A opção 1 – encriptar o utilizador apenas precisa de colocar a mensagem e a chave que irá encriptar a mesma.

Na opção 2 – decifrar o utilizador terá de colocar uma mensagem previamente cifrada por este programa e a chave que cifrou a mensagem, se o fizer obterá o texto decifrado como resultado.

Funcionamento do programa do ponto de vista da programação:

O utilizador introduz uma mensagem a enviar, que é guardada em letras maiúsculas, sem espaços e caracteres especiais. Também introduz uma chave de encriptação que é convertida em letras maiúsculas e apenas consiste em uma palavra, apenas funciona com as 26 letras maiúsculas do alfabeto latino.

Foi criada uma classe designada CifraColTrans, que através dos métodos de acesso se guarda a mensagem e a chave.

Através do método atribuirNumColuna define-se a ordem de transposição das colunas tendo por base a posição no alfabeto de cada letra da chave, por exemplo, DCABEFG,

terá as posições de colunas [4][3][1][2][5][6][7], esse cálculo é gravado no atributo colTransChave, que é uma lista de números inteiros.

Antes da encriptação, temos de produzir a matriz da mensagem, mas dado que podemos ter uma matriz imperfeita é realizado um cálculo para preencher as células sem valor. Por exemplo, para uma mensagem com 25 letras e uma chave de 4 letras, iremos ter 3 células sem nenhuma letra.

Para fazer o preenchimento dessas células vazias optei por colocar espaços vazios. No fim do processo, obtém-se uma matriz em que todas as células têm um caractere.

A encriptação é realizada através do método encriptação, que recebe a matriz definida anteriormente como parâmetro. É construído um texto encriptado através da leitura das colunas pela ordem data pelo método obterNumColuna(getColTransChave()), como indicado em (Stallings, 2017).

No fim obtém-se um texto cifrado corrido e sem espaços entre as palavras originais, que é guardada no atributo mensagemEncriptada, através do respetivo método set.

Para decifrar, realiza-se o processo de reconstrução da matriz, recorrendo à chave que indica os números das colunas a partir do qual se faz a reconstrução da matriz original com os caracteres nas posições originais.

No fim, obtém-se a mensagem decifrada. Como nota, se for inserida uma mensagem encriptada e for colocada uma chave diferente da que foi utilizada para encriptar, o programa produz na mesma um resultado não verifica a validade da chave, pois isso invalidaria o propósito da mesma.

Exemplo:

Mensagem: ATTACK POSTPONED UNTIL TWO AM

Chave: DCABEFG ([4][3][1][2][5][6][7]).

Mensagem Cifrada: TTNAAPTMTSUOAODWCOIDKNLGPETK

Mensagem Decifrada: ATTACKPOSTPONEDUNTILTWOAM

Para a **cifra assimétrica** optei pela implementação do algoritmo RSA. Os conceitos teóricos que suportam a implementação foram obtidos na seção 9.2 de (Stallings, 2017) e consolidados através de (Woo, 2020), (AdminS, 2020). Para a implementação especifica na linguagem de programação JAVA foram utilizados os conceitos definidos em (AdminS, 2020), (Codedost, 2020), (Manish, 2020), (howtodoinjava, 2020), (mkyong, 2020), (scanftree.com, 2020), (thejavaprogrammer, 2020), para além dos conhecimentos que possuo em programação por objetos na linguagem JAVA.

Pela pesquisa realizada nos autores anteriores verificou-se que a classe BigInteger era a mais adequada para realizar as operações matemáticas e em particular as que

envolvem o cálculo de módulos pois já possui as funções principais requeridas pelo algoritmo RSA sendo necessário somente introduzir as constantes. Outra vantagem é método getBytes() presente na classe String que permite a conversão de um texto em bytes, isso é particularmente útil pois transformamos carateres em números e podemos assim realizar operações matemáticas sobre os mesmos através da classe BigInteger, e depois devolve-se uma lista de Bytes que pode ser facilmente convertida em String.

Com estes conhecimentos tomou-se a opção de realizar 2 classes, uma classe main com o menu de interação para com o utilizador e uma classe CifraRSA onde serão definidos os métodos de acesso e o método construtor para a classe.

Na main, aparece um menu que solicita ao utilizador qual a opção a tomar, selecionando 1 o programa inicia automaticamente uma instância da classe CifraRSA bem como o seu método construtor.

Automaticamente serão gerados os 5 passos definidos página 296 em (Stallings, 2017).

Passo 1: Através do método probablePrime define-se os números primos p e q, tendo em conta o valor de bitlength e um número aleatório. O valor de bitlength deverá ser preferencialmente igual ou superior a 512. Como nota se for muito baixo, a cifra não é bem implementada pois os números p e q gerados são demasiado baixos para permitir que um numero elevado de carateres possa ser efetivamente cifrado, em fase de testes colocar bitlength = 16, é pertinente para verificar a conversão de apenas uma letra ou 2 letras e acompanhar o processo de encriptação e decriptação e realizar depuração do programa. No programa submetido ficou o valor de 1024.

Passo 2: Calcular n = pq

Passo 3: Calcular $\phi(pq) = (p-1)(q-1)$

Passo 4: A seleção do expoente **e** é realizada através do mesmo método que se obteve os números primos **p** e **q**.

Mas, o número **e** encontrado pode não respeitar as condições de ser coprimo de $\phi(n)$e menos que $\phi(n)$, pelo que recorrendo ao método gcd e compareTo realiza-se um ciclo while até encontrar o primeiro coprimo de $\phi(n)$ que seja menor que $\phi(n)$, isso faz-se adicionando uma unidade a **e**, até que apareça uma numero que satisfaça estas duas condições.

Passo 5: Calcular d, mais uma vez a utilidade da classe BigInteger é demonstrada, bastando para o efeito utilizar um método que já realiza a operação matemática que permite obter d através da multiplicação do inverso de e pelo modulo de $\phi(n)$, e ainda lança um erro caso n não seja coprimo de **e**.

Passos complementares: inicia-se os atributos PU, PR, que guarda logo as chaves publica e privada respetivamente. Facilita a obtenção das chaves. E inicia-se, as strings M e C_d_mod_n que serão utilizadas para guardar a mensagem a enviar e a mensagem final decifrada.

Depois, no código foram definidos os getters and setters para os atributos da classe, nem todos são utilizados, mas permite escalar a aplicação no futuro.

Dos métodos criados realçam-se 3 métodos, encriptar, decifrar e bytesToString. Mas, dado que partir daqui o processo continua na Main pois depende da mensagem a introduzir pelo utilizador, iremos abordar em detalhe estes métodos na sequência das instruções do programa.

O utilizador inicia o programa e aparece o menu de opções e escolha, neste programa temos apenas duas: 1.RSA Encriptar/Decifrar e 0.Sair. Este menu, está em ciclo, pelo que é possível ir testando diferentes textos sem ser necessário estar sempre a iniciar o programa.

Seleciona a opção 1, e introduz o texto que pretender, nos testes que realizei o programa conseguiu encriptar e decifrar todos os caracteres ASCII, só no caso de colocar o bitlenght muito baixo em relação à mensagem a transmitir é que não executa corretamente as operações por razões anteriormente explicadas.

Ao selecionar a opção 1, automaticamente temos a criação da instância inste sobre a qual iremos realizar todas as operações de encriptar e decifrar.

Solicita-se através da classe Scanner que o utilizador introduza uma mensagem.

A mensagem é guardada no atributo M, uma analogia com a letra M utilizada na descrição do algoritmo em (Stallings, 2017), os restantes símbolos seguem também esta analogia.

É feito o retorno da mensagem inserida para o utilizador.

A mensagem é convertida em bytes através do método getBytes() da classe String e guardada no atributo mToBytes através o método setMToBytes

É feito o retorno da mensagem original, mas em lista de valor de bytes, recorrendo para isso ao método bytesToString, que a partir de um ciclo for constrói uma String com o numero de cada Byte, dado que estamos a trabalhar com a tabela de valores ASCII cada numero decimal dado por um Byte corresponde a um carater, e ao ser colocado na classe String podemos imprimir os valores para a consola onde o utilizador pode compreender mais facilmente quais os caracteres. Este método bytesToString será utilizado sempre que temos uma lista byte[] e pretendemos verificar o texto contido na mesma.

A encriptação é realizada tendo como parâmetro a lista byte[] mToBytes e mais uma vez recorrendo a um dos métodos da classe BigInteger, o modPow que executa a

operação C = M^e mod n sobre a lista byte[] mToBytes e através do método toByteArray() produz outra lista byte[]. Que através do método setEncriptado é copiada para o atributo encriptado, uma lista de byte[].
Produz-se o retorno desta nova lista de byte[] encriptado através do método bytesToString.
O passo seguinte é o decifrar que utiliza os mesmos conceitos existindo apenas a troca do valor do expoente de **e** para **d** como definido no algoritmo RSA.
E também neste caso ocorre o retorno desta lista de byte[] decifrada, o atributo para este caso foi designado por C_d_mod_n que sabemos que terá de ser igual a M para que o programa esteja a funcionar corretamente.

Dos testes que realizei o programa funciona, encripta e decifra automaticamente textos.
Uma evolução do programa seria executar o processamento do Algoritmo de acordo com a Figura 9.7 RSA Processing of Multiple Blocks da página 298 de (Stallings, 2017), não obstante, tendo em conta os objetivos requeridos no trabalho considero que o trabalho apresentado é suficiente para demonstrar a aquisição de conhecimentos relativo ao conceito de chave assimétrica e a forma matemática como o mesmo é possível de ocorrer, que no caso deste algoritmo baseia-se fortemente na teoria dos números, e em particular nas propriedades matemáticas dos números primos.
Realço no fim como extra os conhecimentos adquiridos referentes á classe BigInteger que providencia os métodos que realizam os cálculos matemáticos mais complexos e ao método getBytes() da classe String que em conjunto são a base deste programa.

REFERÊNCIAS

AdminS. (2020). *MANUAL RSA ENCRYPTION AND DECRYPTION WITH JAVA*. Retrieved from https://www.youtube.com/watch?v=JlGGBgKP54A

Codedost. (2020). *https://codedost.com/*. Retrieved from https://codedost.com/css/java-program-rsa-algorithm/

HouseEducation. (2019). *rail fence cipher in java (made easy)*. Retrieved from https://www.youtube.com/watch?v=reKq19WH4q8

howtodoinjava. (2020). *Convert byte[] Array to String and Vice-versa*. Retrieved from https://howtodoinjava.com/: https://howtodoinjava.com/java/array/convert-byte-array-string/

Manish, B. (2020). *https://www.sanfoundry.com*. Retrieved from https://www.sanfoundry.com/java-program-implement-rsa-algorithm/

Mathispower4u. (2013). *Cryptography: Transposition Cipher*. Retrieved from https://www.youtube.com/watch?v=sHsnH1u03e4

mkyong. (2020). *How to convert byte[] array to String in Java*. Retrieved from https://mkyong.com/: https://mkyong.com/java/how-do-convert-byte-array-to-string-in-java/

Pixel, V. (2019). Retrieved from Columnar Transposition Cipher Encryption and Decryption in Java: https://www.youtube.com/watch?v=9j2veckXChk

prototypeprj. (2020). *https://prototypeprj.blogspot.com/*. Retrieved from https://prototypeprj.blogspot.com/2020/07/rsa-w-java.html

QuickCS. (2018). *columnar transposition cipher | COMPUTER CRYPTOGRAPHY | COLUMNAR CIPHER ENCRYPTION AND DECRYPTION*. Retrieved from https://www.youtube.com/watch?v=p9jz-Dg_aQQ&feature=emb_err_woyt

scanftree.com. (2020). *https://scanftree.com/*. Retrieved from scanftree: https://scanftree.com/programs/java/implementation-of-rsa-algorithmencryption-and-decryption-in-java/

Stallings, W. (2017). *Cryptography and Network Security* (7 (Global Edition) ed.). Harlow, England: Pearson Education Limited.

thejavaprogrammer. (2020). *https://www.thejavaprogrammer.com/*. Retrieved from RSA Algorithm in Java (Encryption and Decryption) : https://www.thejavaprogrammer.com/rsa-algorithm-in-java/

VoxelPixel. (2019). *VoxelPixel Columnar Transposition Cipher*. Retrieved from https://github.com/VoxelPixel/CiphersInJava/blob/master/ColTransCipher.java

Woo, E. (2020). *WOOTUBE*. Retrieved from https://misterwootube.com/2020/03/18/about/: https://www.youtube.com/watch?v=4zahvcJ9glg