



Elias Nogueira  
Isadora Rodrigues  
João Pedro Nunes  
Paloma Lacerda  
Yanka Raíssa

## **Criptografia RSA**

Maceió  
2020

**Elias Nogueira, Isadora Rodrigues, Paloma Lacerda, João Pedro Nunes, Yanka Raissa**

## **PROJETO FINAL**

### **Criptografia RSA**

Algoritmo escrito em Python capaz de gerar uma chave pública e encriptar/desencriptar uma mensagem

Apresentação textual do projeto,  
requerido para a obtenção  
de nota para disciplina  
Matemática Discreta.  
Docente: Bruno Pimentel

Maceió  
2020

## 1. Referente à linguagem de programação escolhida pela equipe

A linguagem utilizada no decorrer do projeto será Python, por acreditarmos que a mesma é de fácil atendimento e possui recursos que facilitam a construção do código. Além disso, outro fator determinante foi a falta de experiência dos integrantes com a linguagem, o que despertou o interesse em aprender como implementar a documentação e as regras de sintaxe da mesma.

## 2. Objetivos

- 1 -> Gerar as chaves
- 2 -> Encriptar a mensagem
- 3 -> Desencriptar a mensagem

### 2.1. Escolha do Usuário

Para fazer a criptografia desejada é necessário primeiramente solicitar ao usuário que digite qual operação ele deseja realizar. A variável responsável por armazenar essa escolha será chamada de *opção*.

Após obter o dado do usuário, o passo seguinte é utilizar uma estrutura de decisão (utilizamos if e elif) para fazer essa verificação.

```
134 while(1):
135     print('Escolha uma opção digitando o número correspondente:\n')
136     print('1 - Gerar Chave Pública\n')
137     print('2 - Encriptar\n')
138     print('3 - Desencriptar\n')
139     print('0 - Sair do programa\n')
140
141     opcao = int(input())
```

(figura 1)

Existem 3 cenários possíveis:

Se a variável *opção* tiver como valor 1 ele irá solicitar 2 números primos para que a chave possa ser gerada. Na sequência, será verificado com a função *e\_primo* se o número é primo.

```

143     if opcao == 1:
144         print('Digite dois números primos:')
145
146         #lendo e verificando se os valores são primos
147
148         primo1 = int(input())
149         if e_primo(primo1) == 0:
150             print('\nEsse número não é primo, tente novamente!')
151             print('_____ \n')
152             continue
153
154         primo2 = int(input())
155         if e_primo(primo2) == 0:
156             print('\nEsse número não é primo, tente novamente!')
157             print('_____ \n')
158             continue

```

(figura 2)

Se a variável *opção* tiver como valor 2 (Encriptar), vai ser solicitado ao usuário que digite a mensagem a ser encriptada. Para que isso aconteça é necessário que ele saiba as chaves públicas que foram criadas anteriormente. Ele pode verificar o valor dessas chaves no arquivo .txt criado no diretório armazenado no computador.

```

184     elif opcao == 2:
185         print('\nDigite mensagem que deseja encriptar, sem acentos e caracteres especiais:')
186         mensagem = input()
187
188         print('Digite as chaves públicas:')
189         chave_publi1 = int(input())
190         chave_publi2 = int(input())
191
192         encriptar(mensagem, chave_publi1, chave_publi2)#função para encriptar
193         test()
194         print('\nMensagem Encriptada com sucesso!\n')
195         print('_____ \n')
196

```

(figura 3)

E por fim, caso a variável assumo valor 3 (Desencriptar), será solicitado ao usuário que digite p, q e e respectivamente, para que assim, possa ser verificada as chaves públicas.

```

199     elif opcao == 3:
200         print('Digite p:')
201         p = int(input())
202         print('Digite q:')
203         q = int(input())
204         print('Digite e:')
205         e = int(input())
206
207         tontiente = (p - 1) * (q - 1)
208         chave1 = p*q
209         inverso = calcular_inverso(e,tontiente)#função para achar o inverso utilizado para descriptar a mensagem
210
211         descriptar(inverso, chave1)#função para descriptar
212         print('\n')
213         barra()
214         print('Mensagem Descriptada com sucesso!\n')
215         print('_____ \n')

```

(figura 4)

## 2.2. Geração das chaves públicas a partir de números primos (*função e\_primo*)

Para verificar se um número é primo, utilizaremos as propriedades da matemática, a primeira afirma que *toda raiz de um número primo que for menor que 2 é primo* e a outra é a definição de primo, *todo número primo é divisor apenas de 1 e dele mesmo*. Assim sendo, qualquer divisão exata que ocorra comprova que ele não é primo. Caso nenhum dos dois ocorra, incrementamos o valor do contador e verificamos até concluirmos se é primo ou não.

```

18 def e_primo(x):#função para descobrir se o número é primo ou não
19     i = 2
20
21     while(1):
22         if i > math.sqrt(x):#Limitando até a raiz do número
23             return 1
24         if x % i == 0:#verificando se existem divisores exatos
25             return 0
26         i = i + 1

```

(figura 5)

A *chave1* será gerada por meio da multiplicação dos 2 primos que o usuário inseriu e a *chave2* será um número coprimo de *totiente\_de\_euler* =  $(p-1)(q-1)$ , sendo p e q os respectivos números primos inseridos pelo usuário.

```

160         #gerando o tontiente de euler e a chave 1
161         tontiente = (primo1 - 1) * (primo2 - 1)
162         chave1 = primo1 * primo2

```

(figura 6)

Logo depois ele irá achar a chave pública 2, para isso iremos precisar de um inteiro  $E$  tal que  $1 < E < \text{totiente}$  de forma que  $E$  e o totiente sejam relativamente primos entre si achando o mínimo divisor comum (MDC) para fazer essa verificação, ou seja, esse valor não pode ter nenhum divisor comum com o totiente de euler. Assim implementamos:

```

164         #lendo a chave 2
165
166         print('\nDigite um número expoente que seja relativamente primo a', totiente)
167         chave2 = int(input())
168
169         if mdc(chave2, totiente) == 0: #verificando se ela é relativamente prima ao totiente de euler
170             print('\nEsse número não é válido, tente novamente!')
171             print('_____ \n')
172             continue

```

```

28 def mdc(x,y): #função para achar o MDC entre dois números, usando o algoritmo de Euclides
29     while(1):
30         if y == 1: #se y = 1 logo eles são primos entre si
31             return 1
32         if y == 0: #caso y chegue a ser igual a zero não são primos entre si
33             return 0
34
35         aux = y
36         y = x % y
37         x = aux

```

(figuras 7 e 8)

Pela definição de primo, temos que *qualquer MDC de 1 e x o número será primo*, logo se o y for 1 retornará verdadeiro como primo, pela mesma razão se y for 0 o retorno será falso para primo. Depois utilizamos o algoritmo de euclides para fazer a verificação do MDC, sempre substituindo o valor de x e y pelo resto da divisão e assim validando a veracidade do número.

Após gerar as 2 chaves será criado um arquivo .txt onde será armazenado o valor das chaves públicas

```

174         arquivo = open('Chave_Pública.txt', 'w')
175         arquivo.write(str(chave1))
176         arquivo.write('\n')
177         arquivo.write(str(chave2))
178         arquivo.close()
179         barra()
180         print('\nChaves geradas com sucesso!\n')
181         print('_____ \n')

```

(figura 9)

## 2.3. Encriptando a mensagem

(ver figura 3)

Após ler a mensagem a ser encriptada e as chaves públicas do usuário, iremos encriptar a mensagem por meio da *função encriptar*, colocando como parâmetro da função a mensagem e as chaves.

```
192         encriptar(mensagem, chave_publi1, chave_publi2)#função para encriptar
193         barra()
194         print('\nMensagem Encriptada com sucesso!\n')
195         print('_____ \n')
```

(figura 10)

*Obs: A função barra serve para colocar a barra de carregamento na tela no mesmo tempo em que a mensagem está sendo encriptada.*

```
5  #Funções para gerar a barra de progresso
6  def progress_bar(done):
7      print("\rProcessando: [{0:50s}] {1:.1f}%".format('#' * int(done * 50), done * 100),end='')
8
9
10 def barra():
11     for n in range(4):
12         progress_bar(n/3)
13         time.sleep(1)
14     print('\n')
```

(figura 11)

Para criptografar essa mensagem iremos considerá-la uma string e assim transformaremos letra por letra na mensagem codificada. Abrindo o arquivo .txt criado, atribuímos a mensagem encriptada nesse arquivo.

```
109 def encriptar(mensagem, chave_publi1, chave_publi2):
110     i = 0
111
112     open('Mensagem_Encriptada.txt', 'w')
113     arquivo = open('Mensagem_Encriptada.txt', 'a')#atribuindo os dados do arquivo Mensagem_Encriptada.txt para uma variavel
114
```

```

115     while(i < len(mensagem)):
116
117         if ord(mensagem[i]) == 32:#função ord retorna a posição da tabela ASCII
118             cript = 28#caso especial do caractere "espaço"
119         elif ord(mensagem[i]) > 64 and ord(mensagem[i]) < 91:#caso seja letra maiuscula
120             cript = ord(mensagem[i]) - 63#utilizando as posições da tabala ASCII que fique no intervalo desejado(a=2, z=27)
121         elif ord(mensagem[i]) > 96:#caso seja letra minuscula
122             cript = ord(mensagem[i]) - 95
123
124         cript = pow(cript,chave_publi2)%chave_publi1 #encriptando a letra
125
126         arquivo.write(str(cript))#salvando no arquivo
127         arquivo.write(' ')
128
129         i = i + 1
130
131     arquivo.close()

```

(figura 12)

Utilizando os valores da tabela ASCII e o alfabeto de letras A - Z, codificado com inteiros de 2 a 28, onde 2 = A, 3 = B,..., 27 = Z, 28 = espaço, teremos 2 possibilidades de escrita, maiúsculo ou minúsculo. Assim, para que ele fique no intervalo de A-Z ou seja maior que 64 e menor que 91 subtraímos 63 para que o valor fique no intervalo de 2 a 28.

Por exemplo, caso a letra seja Z que na tabela corresponde a 90 ao subtrair 90-63 teremos 27 que corresponde ao intervalo citado anteriormente. Após isso iremos utilizar a fórmula  $ac \equiv bc \pmod{m}$  para criptografar cada letra da mensagem. Pegamos o *array cript* e elevamos a chave pública 2 e encontramos o resto disso com a chave pública 1.

## 2.4. Desencriptando a mensagem

(ver figura 4)

Será necessário gerar o inverso do tontiente de euler, ou seja, *um número que multiplicado por e vai apresentar resto um ao ser dividido pelo tontiente de euler*. Para isso será necessário encontrar o mínimo divisor comum e encontrar os restos das divisões de cada divisor encontrado a representação na fórmula matemática ficaria:  $de \equiv 1 \pmod{(\text{totiente de euler})}$ .



```

40 def calcular_inverso(e,totiente_de_euler):#função para achar o inverso
41     divisores = [0]
42     contador = 0
43
44     #achando o inverso utilizando o algoritmo de Euclides
45
46     if totiente_de_euler > e:
47         aux = e
48         e = totiente_de_euler
49         totiente_de_euler = aux
50
51     mod = e
52
53     while totiente_de_euler > 0:
54         divisores.append(int(e/totiente_de_euler))
55
56         resto = e % totiente_de_euler
57         e = totiente_de_euler
58         totiente_de_euler = resto
59
60         contador += 1
61
62     verificar_negatividade = contador
63     n_anterior = 0
64     n_atual = 1
65     contador -= 1
66
67     while contador > -1:
68         n_substituto = n_atual
69         n_atual = divisores[contador] * n_atual + n_anterior
70         n_anterior = n_substituto
71         contador -= 1
72
73     if verificar_negatividade % 2 == 0:
74         n_anterior = n_anterior * (-1)
75         n_anterior = mod + n_anterior
76
77     return n_anterior

```

(figura 13)

Logo, ao encontrar o inverso iremos descriptografar a mensagem colocando como parâmetro da função o inverso encontrado e a chave 1. Assim como fizemos ao criptografar a mensagem, iremos considerá-la como um array e descriptografar letra por letra, dessa vez iremos pegar o número e elevar ao inverso e encontrar o resto da divisão com a chave 1, ( $letra = pow(int(n), inverso) \% chave1$ ) e após iremos transformar o resto de volta no número correspondente à tabela ASCII, somando +63 para que saia do intervalo 2-28.

```

81 def descriptar(inverso, chave1):
82     arquivo = open('Mensagem_Encriptada.txt', 'r')#abrindo o arquivo Mensagem_Encriptada.txt
83     mensagem_encrip = arquivo.read()#colocando os dados de Mensagem_Encriptada.txt em uma variavel
84     des_encrip = open('Mensagem_Descriptada.txt', 'w')#criando ou limpando o arquivo Mensagem_Descriptada.txt
85     arquivo.close()
86
87
88     i = 0
89
90     while(i < len(mensagem_encrip)):
91         n = ''
92         while(1):
93             if mensagem_encrip[i] == " ":
94                 letra = pow(int(n), inverso)%chave1#descriptando
95                 if letra == 28:#caso especial do espaço
96                     des_encrip.write(' ')
97                 else:
98                     des_encrip.write(chr(letra + 63))#convertendo para string pela tabela ASCII e salvando no arquivo
99                 break
100             n = n + str(mensagem_encrip[i])
101             i = i + 1
102         i = i + 1
103
104     arquivo.close()
105     des_encrip.close()

```

(figura 14)

Essa string será armazenada em um arquivo .txt. Enquanto a mensagem é descriptografada será printado na tela uma barra de carregamento, para que o usuário saiba que os dados estão sendo processados. Assim que tudo for finalizado uma mensagem vai sinalizar que tudo ocorreu corretamente.

```

211         descriptar(inverso, chave1)#função para descriptar
212         print('\n')
213         barra()
214         print('Mensagem Descriptada com sucesso!\n')
215         print('_____ \n')

```

(figura 15)

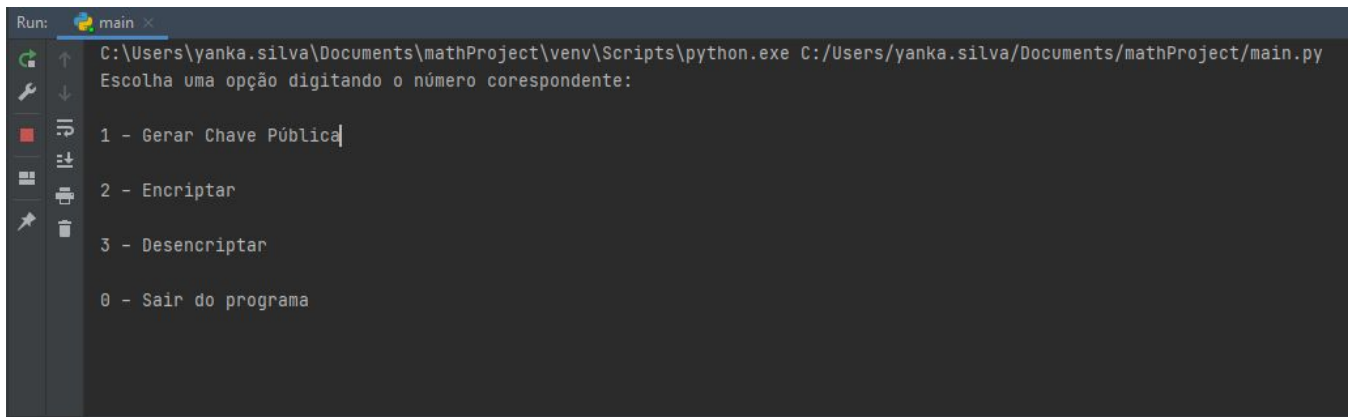
```

217         elif opcao == 0:
218             break

```

(figura 16)

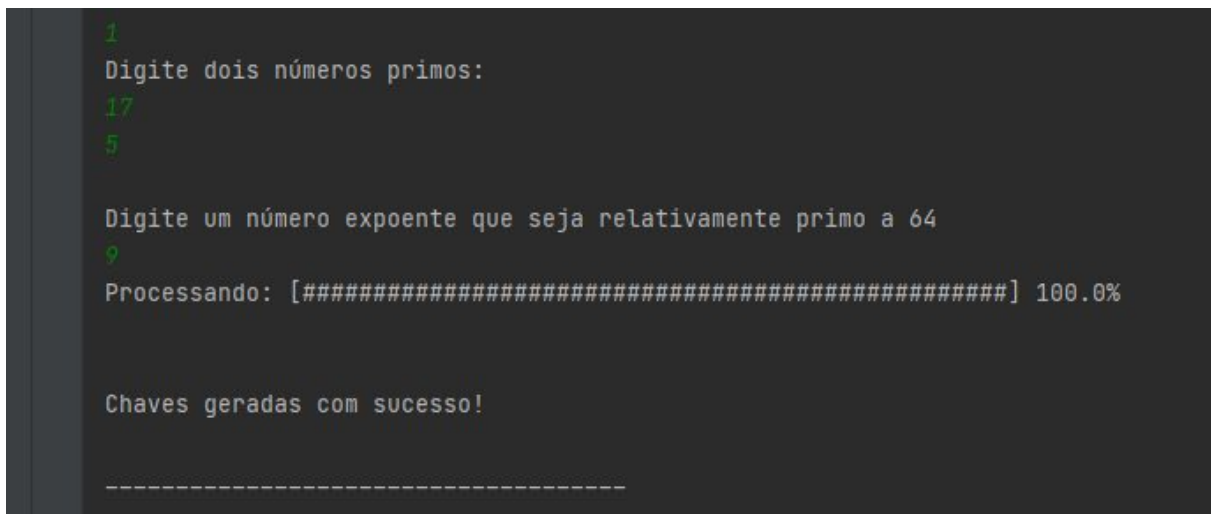
### 3. Demonstração



```
Run: main x
C:\Users\yanka.silva\Documents\mathProject\venv\Scripts\python.exe C:/Users/yanka.silva/Documents/mathProject/main.py
Escolha uma opção digitando o número correspondente:

1 - Gerar Chave Pública
2 - Encriptar
3 - Desencriptar
0 - Sair do programa
```

(figura 17 - Menu de opções)

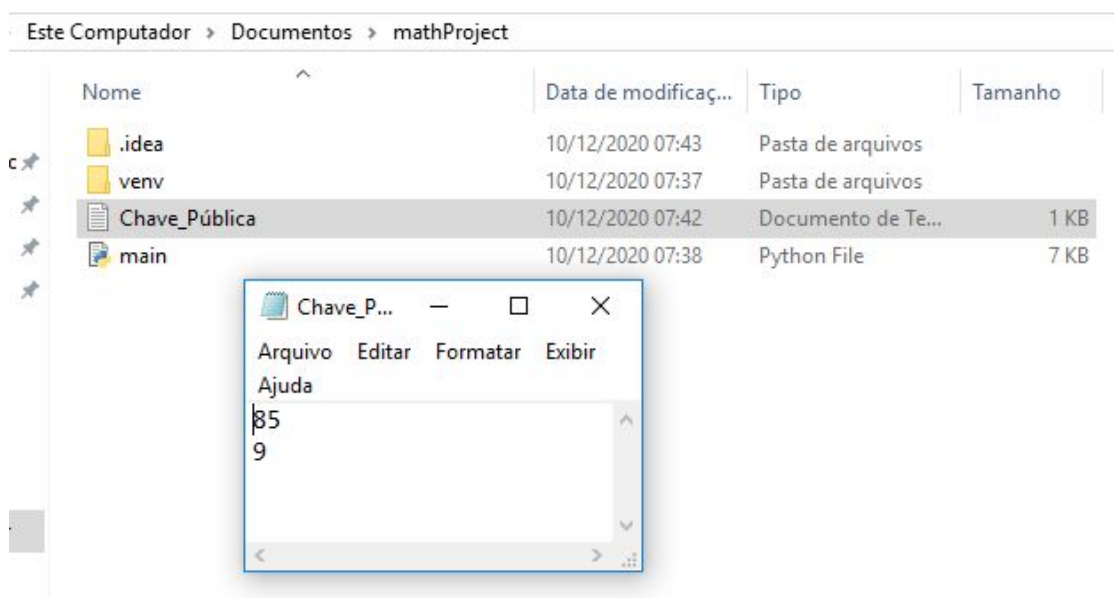


```
1
Digite dois números primos:
17
5

Digite um número expoente que seja relativamente primo a 64
9
Processando: [#####] 100.0%

Chaves geradas com sucesso!
-----
```

(figura 18 - Opção escolhida = 1)



(figura 19 - Arquivo .txt com as chaves criado na pasta do projeto)

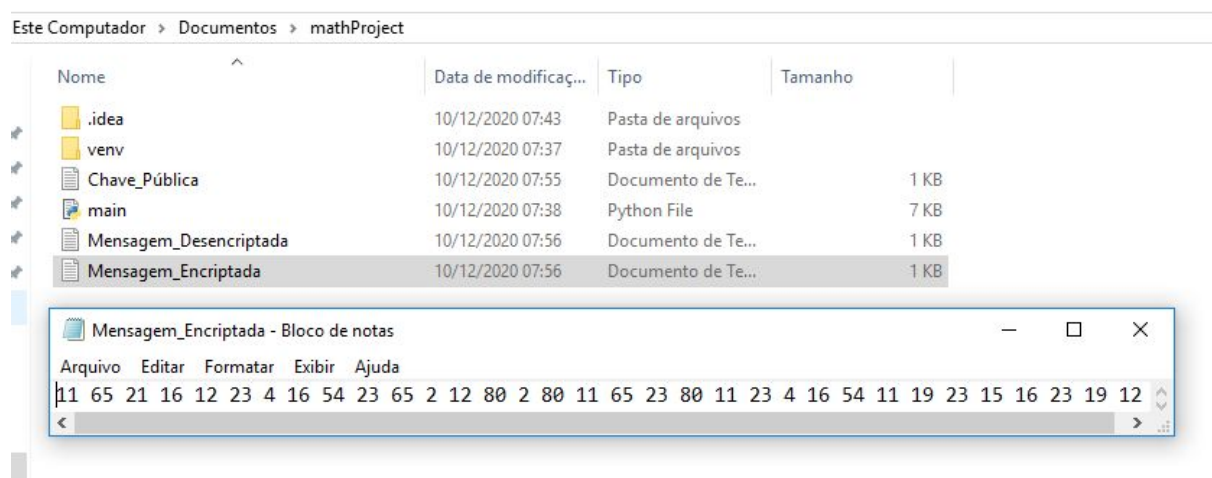
```
2

Digite mensagem que deseja encriptar, sem acentos e caracteres especiais:
Estau com saudades de comer no RU
Digite as chaves públicas:
85
9
Processando: [#####] 100.0%

Mensagem Encriptada com sucesso!

-----
```

(figura 20 - Opção escolhida = 2)



(figura 21 - Arquivo .txt com a mensagem encriptada criada na pasta do projeto)

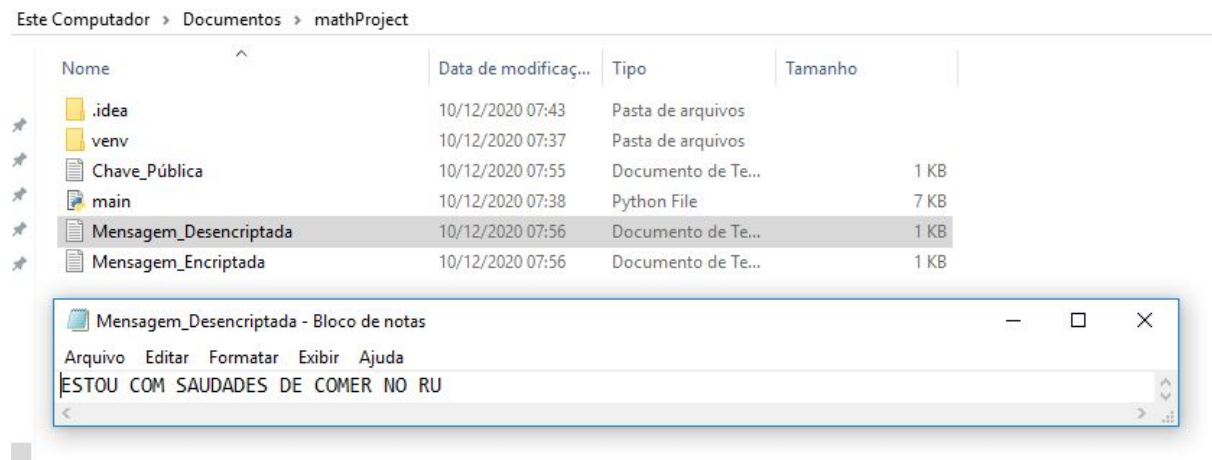
```
3
Digite p:
17
Digite q:
5
Digite e:
9

Processando: [#####] 100.0%

Mensagem Desencryptada com sucesso!

-----
```

(figura 22 - Opção escolhida = 3)



(figura 23 - Arquivo .txt com a mensagem descriptografada criada na pasta do projeto)

```
Escolha uma opção digitando o número correspondente:  
  
1 - Gerar Chave Pública  
  
2 - Encriptar  
  
3 - Desencriptar  
  
0 - Sair do programa  
  
0  
  
Process finished with exit code 0
```

(figura 24 - Encerramento do programa)