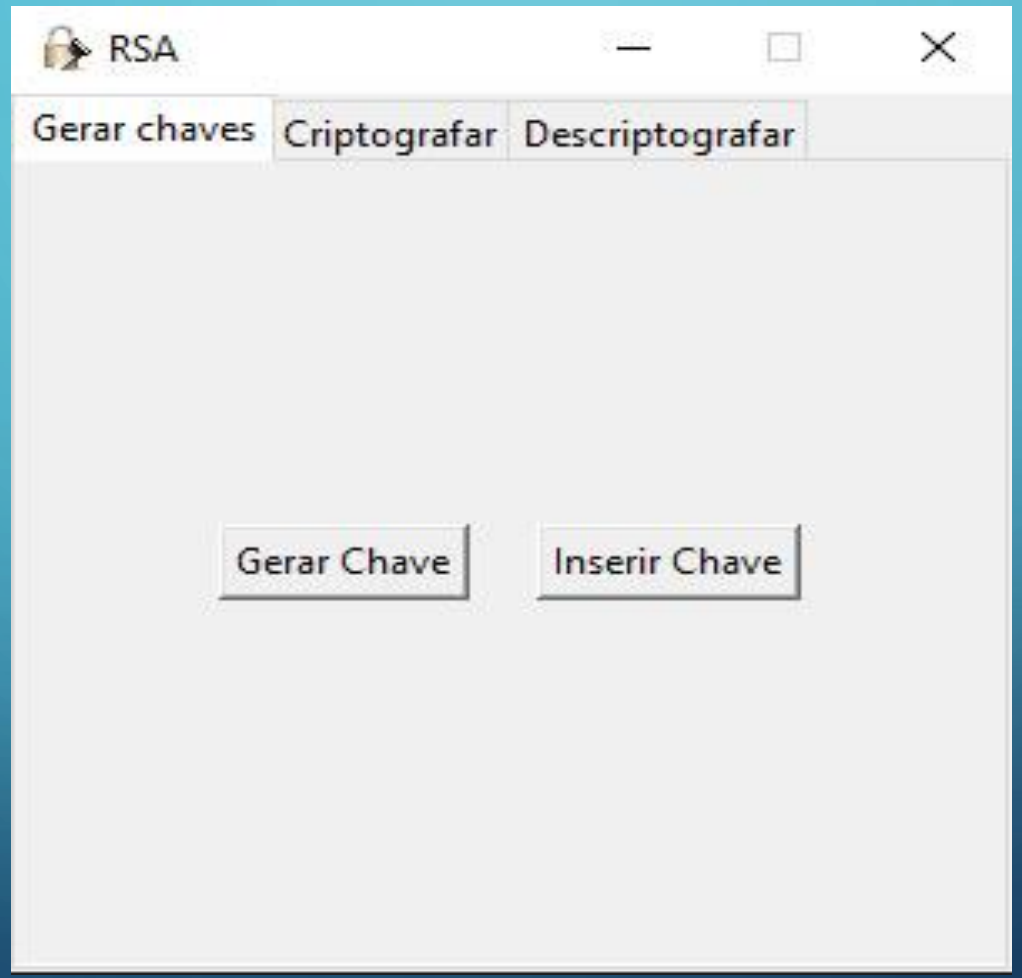
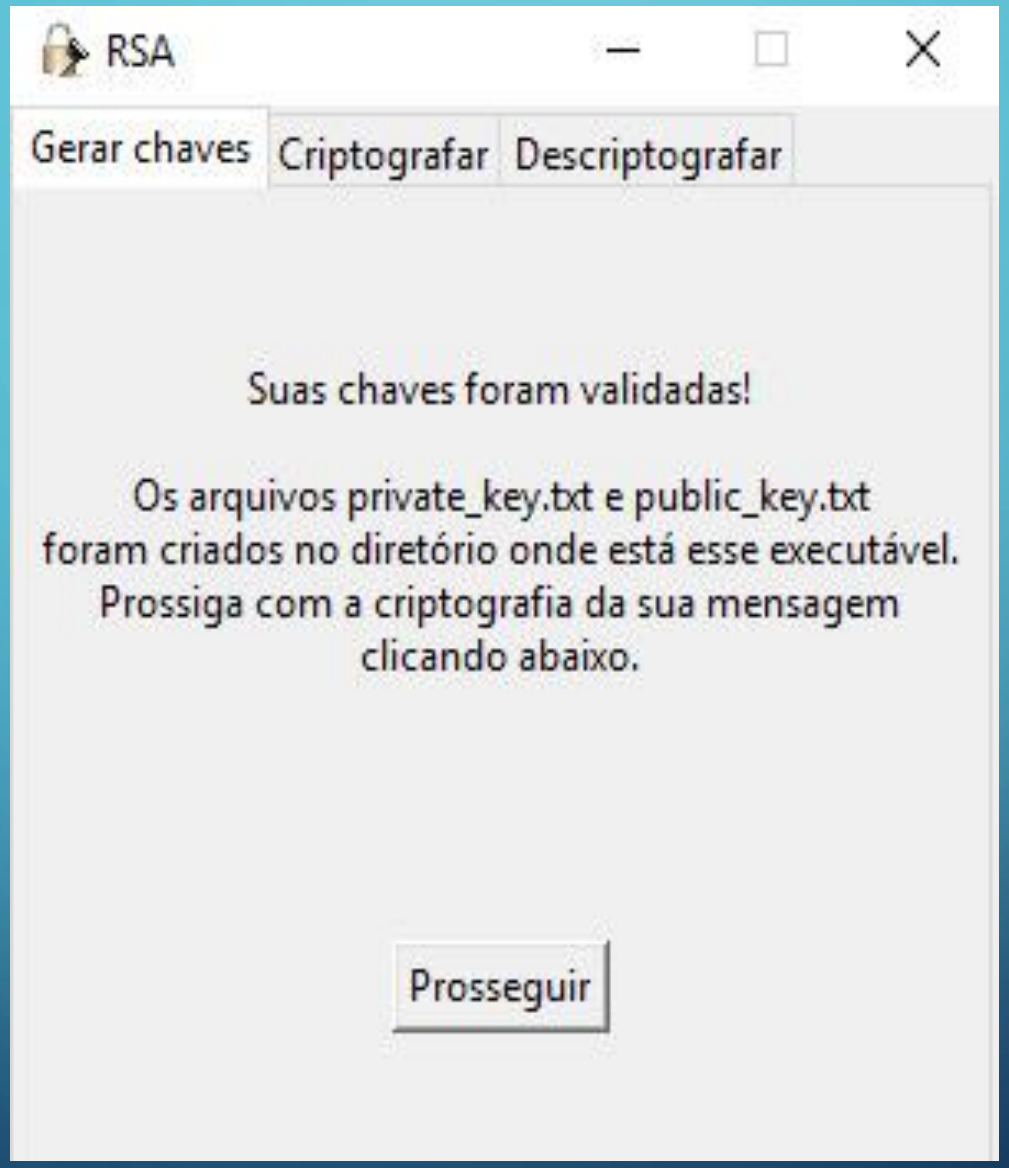
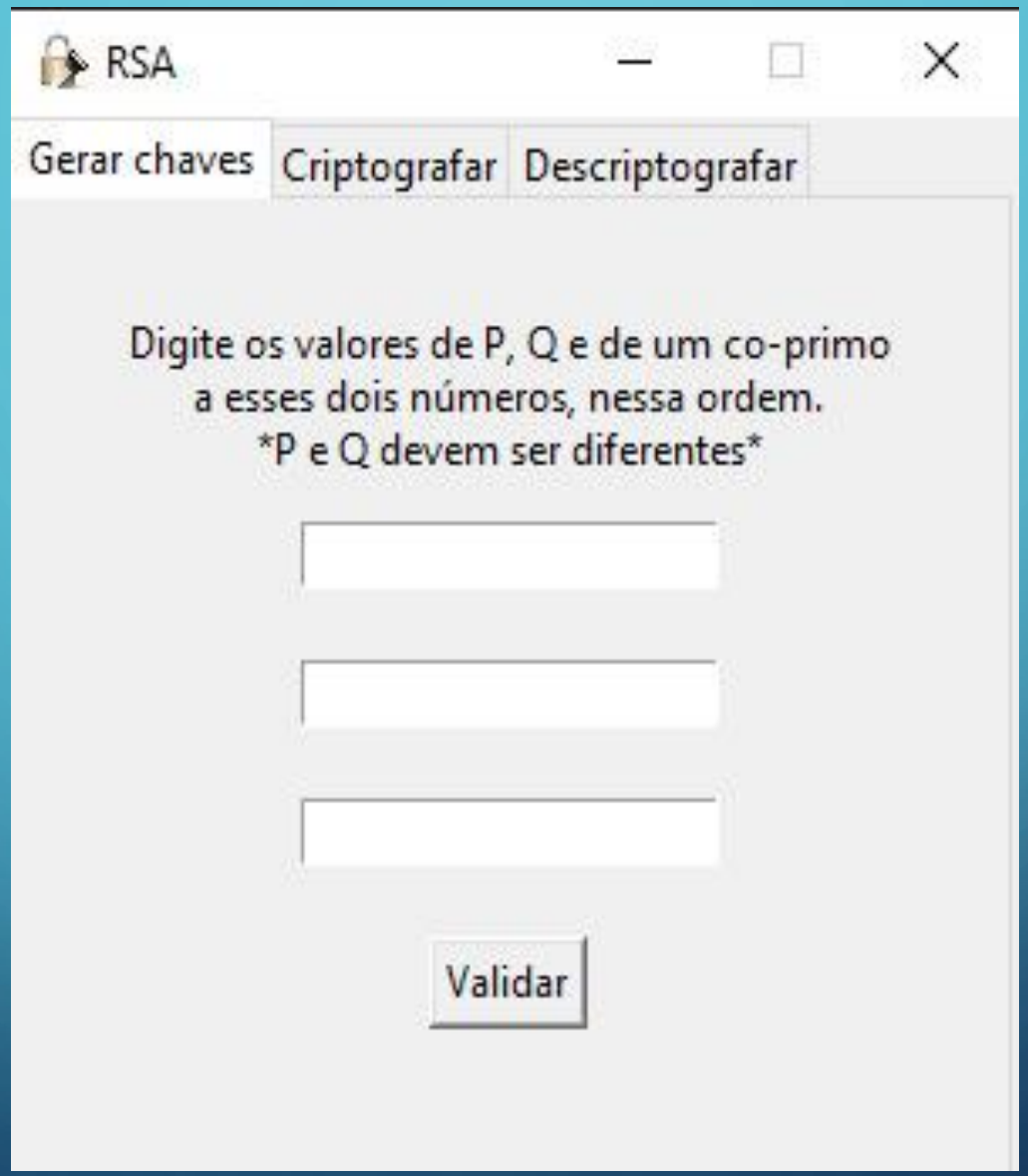
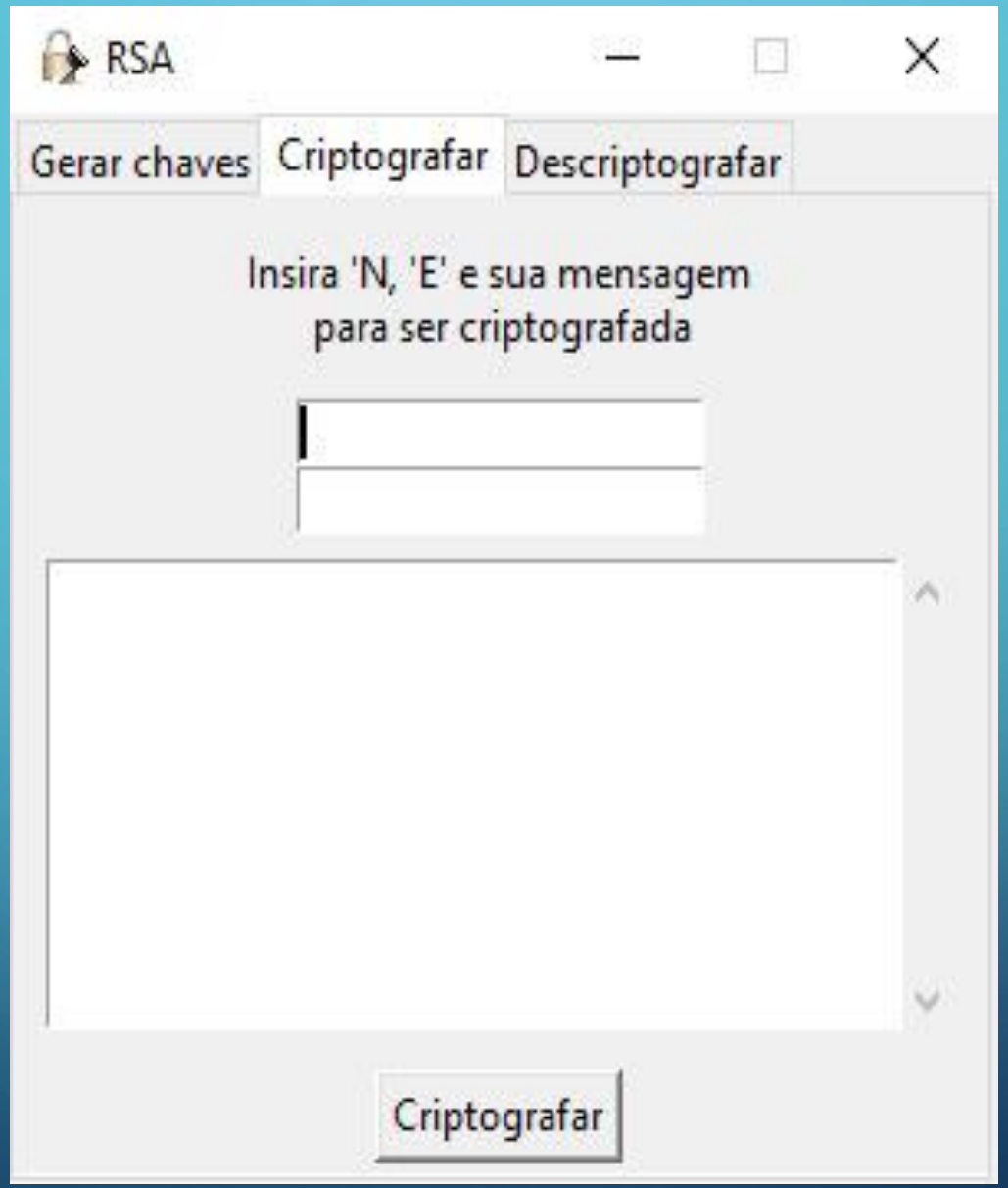
ALGORITMO DE CRIPTOGRAFIA/DESCRIPTOGRAFIA RSA

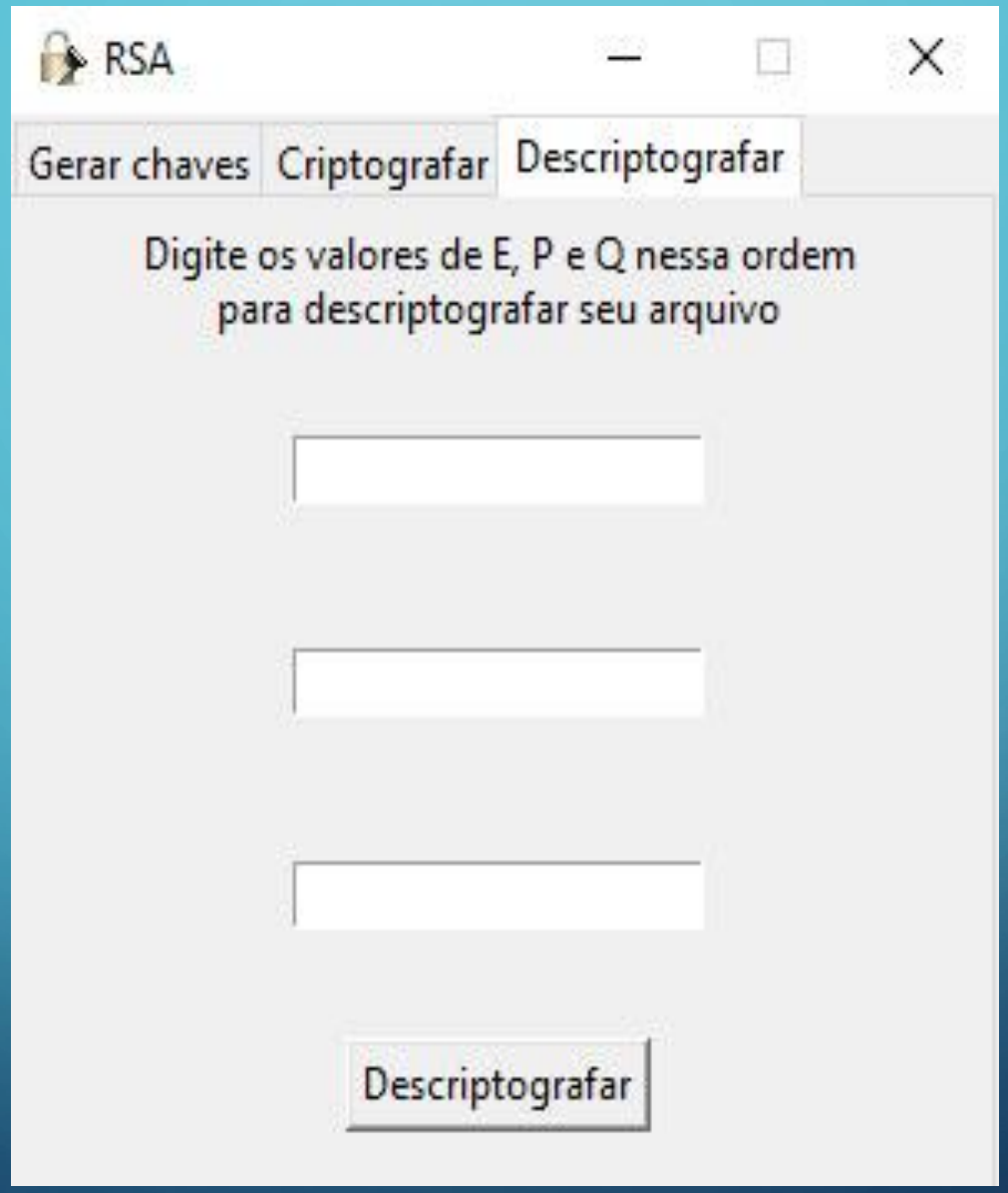
GUI


RSA
Gerar chaves
Criptografar
Descriptografar
Gerar Chave
Inserir Chave


RSA
Gerar chaves
Criptografar
Descriptografar
Suas chaves foram validadas!
Os arquivos private_key.txt e public_key.txt
foram criados no diretório onde está esse executável.
Prossiga com a criptografia da sua mensagem
clicando abaixo.
Prosseguir


RSA
Gerar chaves
Criptografar
Descriptografar
Digite os valores de P, Q e de um co-primo
a esses dois números, nessa ordem.
P e Q devem ser diferentes
Validar


RSA
Gerar chaves
Criptografar
Descriptografar
Insira 'N, 'E' e sua mensagem
para ser criptografada
Criptografar


RSA
Gerar chaves
Criptografar
Descriptografar
Digite os valores de E, P e Q nessa ordem
para descriptografar seu arquivo
Descriptografar

GERAR PRIMOS

```
def gen_prime(): #funcao que gera um numero primo aleatorio entre 10000 e 1000000

    n = secrets.randbits(32) #gerando numero aleatorio dentro do intervalo especificado

    while test_prime(n) == False: #esse ciclo serve para impedir que um numero aleatorio nao primo seja
    retornado
        n = secrets.randbits(32) #atribuindo um novo numero aleatorio a n ate que n seja primo

    return n

def phi(p, q): #funcao totiente
    return (p-1)*(q-1)

def co_primos(x): #funcao que retorna um numero co-primo do numero passado
    y = gen_prime()

    while euclides(x, y) !=1: #ciclo para garantir a condicao de coprimos(numeros que o mdc entre eles e 1)
        y = gen_prime() #enquanto o y nao atender essa condicao ele vai sendo gerado novamente

    return y
```

VERIFICAR SE É PRIMO

```python
def euclides(num1, num2): #algoritmo de euclides utilizado para saber o mdc

        resto = num1 % num2

        if resto == 0: #condicao de retorno do mdc
            return num2

        return euclides(num2, resto)

def test_prime(n): #funcao que checa se um numero e primo

    s = int(math.sqrt(n)) + 3 #segundo a matematica, na busca por um numero primo, so precisamos tentar
    encontrar um divisor ate a raiz quadrada do numero em questao

    if n % 2 == 0: #se o numero for par ja retorna que ele nao e primo
        return False #essa checagem e necessaria pois no ciclo abaixo o divisor incrementa de 2 em 2, se não
        houvessee essa checagem bugaria

    for x in range(3, s, 2): #ciclo que incrementa em 2 o divisor (inicializado em 3) ate ele atingir o limite
    s
        if n % x == 0: #caso seja encontrado um divisor antes do s o numero nao e primo
            return False

    return True #caso nao tenha sido encontrado um divisor nas condicoes acima, o numero e primo
```

CONVERTER INT PARA BIN

```python
def generate_list_int(convert, list_int_pow): #gerador de uma lista que contem a decomposicao do numero binario em potencias de base 2
    j = 0 #posicao na string

    for i in convert:
        if i == "0": #se o algarismo binario da posicao for 0, ele simplesmente parte para o proximo digito
            j+=1 #incremento do expoente ao qual a base 2 será elevada

        else:
            list_int_pow.append(pow(2, j)) #caso o digito daquela posicao seja 1, ele adiciona a lista uma potencia de base 2 cujo expoente e a posicao do digito na string(no caso o j)
            j+=1 #incremento do expoente ao qual a base 2 será elevada

def int_bin(div, convert): #funcao que converte um valor inteiro para binario, pega int e retorna string

    if div != 1:
        convert = convert + str(div%2)
        return int_bin(div//2, convert)

    else:
        convert = convert + str(div)
        return convert #note que retornando assim o numero binario esta invertido, contudo essa invercao e conveniente a funcao exp_mod_rap
```

EXPONENCIAL MODULAR RÁPIDA

```
def exp_mod_rap(list_int_pow, d, mod, exp, r): #funcao de exponenciacao modular rapida
    if list_int_pow[-1] >= exp: #se o ultimo valor da lista de espoentes for maior ou igual ao expoente atual
    a funcao executa

        if exp == 1: #checagem para definir o r na primeira execucao dessa funcao
            r = d % mod
            if exp in list_int_pow: #checagem para ver se o expoente esta na decomposicao do numero binario
                return r * exp_mod_rap(list_int_pow, d, mod, (exp*2), r) #retornando a mutiplicacao dos fators

        else: #atribuicao feita a r a partir do segundo caso em diante
            r = (r*r) % mod
            if exp in list_int_pow: #checagem para ver se o expoente esta na decomposicao do numero binario
                return r * exp_mod_rap( list_int_pow, d, mod, (exp*2), r) #retornando ja a multiplicacao dos
                fatores

        return exp_mod_rap(list_int_pow, d, mod, (exp*2), r) #caso nao caia em nenhuma das checagens acima a
        funcao nao multiplica

    else:
        return 1
```

ENCRIPT

```
def encript():
    #entradas providas pelo usuario
    n = int(e7.get())                       #PEGANDO O VALOR DE 'N' DA ENTRADA E CONVERTENDO PARA INTEIRO
    e = int(e8.get())                       #PEGANDO O VALOR DE 'E' DA ENTRADA E CONVERTENDO PARA INTEIRO
    mensagem  = scroll.get('1.0', END)      #CAPTURANDO O TEXTO DIGITADO NA SCROLLEDTEXT
    mensagem = mensagem.lower()

    i = 0
    error = 0
    criptografado = ""
    while i < int(len(mensagem) - 1):
        #objetos auxiliares
        list_int_pow = [] #lista auxiliar que vai guardar dados para o funcionamento da funcao exponencial
        modular rapida
        #x = dicionario1[mensagem[i]]#valor de determinado caracter atribuido de acordo com o dicionario
        try :
            x = dicionario1[mensagem[i]]
        except :
            error = -1
            break
        x = dicionario1[mensagem[i]]#valor de determinado caracter atribuido de acordo com o dicionario
        convert = int_bin(e, "") #convertendo o valor de e (o expoente da potenciacao) em binario
        generate_list_int(convert, list_int_pow) #armazenando em "list_int pow" os valores na base 2  que
        decompoem o valor e

        y = exp_mod_rap(list_int_pow, x, n, 1, 0) #exponenciacao modular rapida para criptografar a mensagem

        criptografado = criptografado + str(y%n) +" "#gerando string que contem a mensagem criptografada
        i+=1
```

```
#manipulando arquivo de saida
if error == -1:
    messagebox.showerror("Erro", "Caractere inválido.\nNão use acentos nem pontuações.") #BOX DE ERRO PARA
    CARACTERES INVÁLIDOS
else:
    arquivo = open("encripted.txt", "w") #gerando arquivo txt que guardará o texto descriptografado
    arquivo.write(criptografado) #escrevendo a mensagem criptografada no arquivo
    arquivo.close() #fechando o arquivo
    #RETIRA TODOS WIDGETS DA ABA CRIPTOGRAFAR
    lb2.forget()
    e7.forget()
    e8.forget()
    scroll.forget()
    btn5.forget()
    #EMPACOTA LABEL INFORMATIVA
    lb3.pack(side = TOP, pady = 50)
```

DECRIPT

```
def decript():
    #entradas do usuario
    e = int(e3.get())                #PEGANDO O VALOR DE 'E' E CONVERTENDO PARA INTEIRO
    p = int(e4.get())                #PEGANDO O VALOR DE 'P' E CONVERTENDO PARA INTEIRO
    q = int(e5.get())                #PEGANDO O VALOR DE 'Q' E CONVERTENDO PARA INTEIRO

    #calculo dos valores necessarios
    n = p * q
    tot_n = ((p-1) * (q-1))
    d = inversom_m(e, tot_n) #inverso multiplicativo de e, fundamental para a descriptografia

    #manipulação do arquivo de entrada
    arquivo_cript = open("encripted.txt", "r") #abrindo arquivo criptografado indicado pelo usuario
    mensagem = arquivo_cript.read() #atribuindo conteudo do arquivo criptografado a uma string

    #transformando o conteudo do arquivo em uma lista contendo o numero que representa cada letra criptografada
    lista = mensagem.split(" ")

    arquivo_cript.close() #fechando arquivo de entrada

    #processo de descriptografia
    desc = "" #string vazia que v1ai armazenar a mensagem descriptografada
```

```
for item in lista:
    #objetos auxiliares
    list_int_pow = [] #lista auxiliar que ira guardar dados para o funcionamento da funcao exponencial
    modular rapida
    if item == '': #condicao pra nao bugar no ultimo item da lista que sempre vai ser vazio
        break
    x = int(item) #atribui a x o inteiro da lista que vai ser descriptografado em um caracter
    #r = (x**d) % n
    convert = int_bin(d, "") #convertendo expoente d em binario para poder iniciar a exponenciação
    generate_list_int(convert, list_int_pow) #armazenando em uma lista a decomposição do expoente em
    potencias de base 2
    y = exp_mod_rap(list_int_pow, x, n, 1, 0) #executando a exponenciacao modular rapida com o expoente d

    desc = desc + dicionario2[y%n] #concatenando a mensagem com o caracter que acabou de ser
    descriptografada
```

- Alunos:

- Carlos Noslin
- Jorge Lucas
- Rafael Emílio
- Rick Santos
- Wagner Anthony