Universidade Federal do Piauí

Centro de Ciências da Natureza/CCN

Departamento de Computação

Disciplina: Segurança em Sistemas Computacionais

Docente: Carlos André Batista de Carvalho

Relatório Técnico Trabalho 1 - Implementação: Algoritmos de Criptografia

Discente:

Cayo Cesar L M Pires Cardoso

Teresina

Outubro de 2024

- **Introdução**

1. Resumo

Este trabalho apresenta a implementação de um programa de criptografia que realiza operações de cifragem e decifragem utilizando o algoritmo AES e assinatura digital com RSA. O programa é compatível com a ferramenta online CyberChef e permite a geração de chaves no formato PEM. Foram implementadas funções para cifragem/decifragem AES, geração de chaves, assinatura RSA e verificação de assinatura. O trabalho aborda também a análise dos requisitos, tecnologias utilizadas, implementação, testes e resultados.

2. Objetivos
    a. Objetivo Geral

        Desenvolver um programa que realize operações criptográficas com mensagens em texto, utilizando algoritmos de cifragem e assinatura digital.

    b. Objetivos Específicos

        1. Implementar funções de cifragem e decifragem utilizando o algoritmo AES.
        2. Desenvolver uma ferramenta para geração de chaves no formato PEM.
        3. Implementar funções de assinatura digital com RSA.
        4. Realizar testes e análise dos resultados.
        5. Desenvolver uma interface de usuário amigável.
        6. Tratar erros e exceções.

3. Requisitos Funcionais

    1. Cifragem/decifragem AES.
    2. Geração de chaves PEM.
    3. Assinatura digital RSA.
    4. Verificação de assinatura.
    5. Interface de usuário amigável.
    6. Opções de configuração.
    7. Validação de entrada.
    8. Tratamento de erros.
    9. Eficiência no processamento.

4. Requisitos Não Funcionais

    1. Desempenho.
    2. Segurança.

3. Usabilidade.
4. Manutenção.
5. Portabilidade.

- **Implementação**

1. Main

O arquivo “main.py” contido no projeto é utilizado como raiz, dele sai as escolhas dos menus de toda a aplicação.

```
import tkinter as tk
from tkinter import ttk
from encrypt_decrypt import aes_menu
from generator_keys import keygen_menu
from rsa import rsa_menu
```

Primeiramente tem-se as importações que basicamente são as importações da biblioteca de interface e a importação das funções dos menus das funcionalidades da aplicação.

```
def main_menu():
    root = tk.Tk()
    root.title("Sistema de Criptografia")
    root.geometry("400x400")

    style = ttk.Style()

    style.configure("TButton", font=("Arial", 12), padding=10)
    style.configure("TLabel", font=("Arial", 16), padding=10)

    def exit_program():
        root.quit()

    def open_aes_menu():
        aes_menu()
        return

    def open_rsa_keygen_menu():
        keygen_menu()
        return

    def open_rsa_operations_menu():
        rsa_menu()
        return
```

Através desse trecho de código que o programa acessa os menus de cada funcionalidade implementada, já que estão organizados em 3 arquivos .py diferentes.

2. Cifragem / Decifragem utilizando AES

O código abaixo descrito implementa uma interface gráfica e a lógica para cifrar e decifrar mensagens usando o algoritmo de criptografia AES. Ele permite que o usuário selecione o tamanho da chave, o modo de operação (CBC ou ECB) e o formato de saída/entrada (Hexadecimal ou Base64) para os arquivos gerados. A interface foi construída usando a biblioteca "tkinter" do Python, e a criptografia com a biblioteca "PyCryptodome" para manipulação do AES. Abaixo segue cada parte da explicação detalhada.

```
import base64
import binascii
import tkinter
from tkinter import ttk
import tkinter.messagebox
import tkinter.filedialog
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import os
```

Tem-se acima as importações necessárias para o funcionamento do algoritmo.

```
def encrypt_aes():
    try:
        message = tkinter.simpledialog.askstring("Mensagem", "Digite a mensagem a ser cifrada:")
        if not message:
            raise ValueError("A mensagem não pode estar vazia.")

        key_size = tkinter.simpledialog.askinteger("Tamanho da Chave", "Digite o tamanho da chave (128, 192 ou 256 bits):")
        if key_size not in [128, 192, 256]:
            raise ValueError("O tamanho da chave deve ser 128, 192 ou 256 bits.")

        mode_choice = tkinter.simpledialog.askstring("Modo de Operação", "Escolha o modo de operação (CBC ou ECB):").upper()
        if mode_choice not in ["CBC", "ECB"]:
            raise ValueError("Modo de operação inválido. Escolha entre CBC ou ECB.")

        output_format = tkinter.simpledialog.askstring("Formato de Saída", "Escolha o formato de saída (Hex ou Base64):").upper()
        if output_format not in ["HEX", "BASE64"]:
            raise ValueError("Formato de saída inválido. Escolha entre Hex ou Base64.")
```

Inicialmente no algoritmo de cifragem/decifragem, o programa abre uma janela pra escolha do usuário entre o algoritmo de cifragem e o de decifragem, trazendo a explicação primeiramente para o de cifragem, temos o algoritmo perguntando e salvando a mensagem que o usuário quer cifrar, depois ele pede pro usuário digitar o tamanho da chave, não permitindo que ele digite um tamanho diferente do que os tamanhos que o algoritmo suporta, posteriormente o usuário tem que escolher o modo de operação, se é CBC ou ECB, já que é uma configuração importante para execução porque dependendo do modo vai ser salvo um vetor de inicialização ou não na pasta, e por fim o usuário escolhe o formato de saída da mensagem cifrada, seja HEX ou

Base64, ja que o algoritmo trabalha com dados binários e precisa converter pro modo que o usuário solicitou.

```
key = get_random_bytes(key_size // 8)

iv = None
if mode_choice == "CBC":
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
else:
    cipher = AES.new(key, AES.MODE_ECB)

padded_message = message.encode()
if len(padded_message) % 16 != 0:
    padded_message += b' ' * (16 - len(padded_message) % 16)
ciphertext = cipher.encrypt(padded_message)

current_directory = os.getcwd()
```

Dando progresso no algoritmo, esse trecho acima somente verifica se o formato escolhido pelo usuário foi CBC ou ECB, e como dito anteriormente se for CBC ele gera um vetor de inicialização, mas se for ECB não é preciso. Após isso, a mensagem é convertida para bytes e cifrada.

```
    if output_format == "HEX":
        ciphertext_out = binascii.hexlify(ciphertext).decode()
        key_out = binascii.hexlify(key).decode()
        if iv:
            iv_out = binascii.hexlify(iv).decode()
    else:
        ciphertext_out = base64.b64encode(ciphertext).decode()
        key_out = base64.b64encode(key).decode()
        if iv:
            iv_out = base64.b64encode(iv).decode()

    with open(os.path.join(current_directory, "mensagem_cifrada.txt"), "w") as f_msg:
        f_msg.write(ciphertext_out)
    with open(os.path.join(current_directory, "chave_aes.txt"), "w") as f_key:
        f_key.write(key_out)

    if mode_choice == "CBC" and iv is not None:
        with open(os.path.join(current_directory, "iv.txt"), "w") as f_iv:
            f_iv.write(iv_out)

    tkinter.messagebox.showinfo("Sucesso", "A mensagem foi cifrada e os arquivos foram salvos no diretório do programa!")

except Exception as e:
    tkinter.messagebox.showerror("Erro", str(e))
```

Após ser cifrada a mensagem é convertida para HEX ou Base64 e os arquivos são todos salvos no diretório raiz do programa.

```python
def decrypt_aes():
    try:
        file_ciphertext = tkinter.filedialog.askopenfilename(title="Selecione o arquivo com a mensagem cifrada")
        if not file_ciphertext:
            raise FileNotFoundError("Arquivo com a mensagem cifrada não foi selecionado.")

        file_key = tkinter.filedialog.askopenfilename(title="Selecione o arquivo com a chave AES")
        if not file_key:
            raise FileNotFoundError("Arquivo com a chave AES não foi selecionado.")

        mode_choice = tkinter.simpledialog.askstring("Modo de Operação", "Escolha o modo de operação (CBC ou ECB):").upper()
        if mode_choice not in ["CBC", "ECB"]:
            raise ValueError("Modo de operação inválido. Escolha entre CBC ou ECB.")

        input_format = tkinter.simpledialog.askstring("Formato de Entrada", "Escolha o formato de entrada (Hex ou Base64):").upper()
        if input_format not in ["HEX", "BASE64"]:
            raise ValueError("Formato de entrada inválido. Escolha entre Hex ou Base64.")
```

Já na parte de decifragem o programa inicia pedindo ao usuário alguns arquivos de inicialização, sendo eles: Mensagem Cifrada, Chave AES e se o usuário escolher CBC como modo de operação ele pede o arquivo IV, por fim ele pede pro usuário informar o formato de entrada, HEX ou Base64.

```python
with open(file_ciphertext, "r") as f_msg:
    ciphertext = f_msg.read()
with open(file_key, "r") as f_key:
    key = f_key.read()

if input_format == "HEX":
    ciphertext = binascii.unhexlify(ciphertext)
    key = binascii.unhexlify(key)
else:
    ciphertext = base64.b64decode(ciphertext)
    key = base64.b64decode(key)

key_length = len(key)
if key_length not in [16, 24, 32]:
    raise ValueError(f"Tamanho da chave inválido: {key_length * 8} bits. Use chaves de 128, 192 ou 256 bits.")
```

O trecho de código realiza a leitura dos arquivos que contêm a mensagem cifrada e a chave AES. Ele abre o arquivo da mensagem cifrada (file_ciphertext) em modo de leitura e armazena seu conteúdo em ciphertext. Da mesma forma, lê o arquivo da chave AES (file_key) e armazena em key. Dependendo do formato de entrada escolhido pelo usuário (Hexadecimal ou Base64), o código converte os dados lidos: se o formato for Hex, usa binascii.unhexlify para decodificar; se for Base64, utiliza base64.b64decode. Em seguida, o comprimento da chave é verificado para garantir que esteja entre 16, 24 ou 32 bytes (correspondendo a 128, 192 ou 256 bits). Se a chave não estiver dentro desses tamanhos, um erro é levantado, informando que o tamanho da chave é inválido.

```
iv = None
if mode_choice == "CBC":
    file_iv = tkinter.filedialog.askopenfilename(title="Selecione o arquivo com o vetor IV")
    if not file_iv:
        raise FileNotFoundError("Arquivo com o IV não foi selecionado.")
    with open(file_iv, "r") as f_iv:
        iv = f_iv.read()
    if input_format == "HEX":
        iv = binascii.unhexlify(iv)
    else:
        iv = base64.b64decode(iv)

if mode_choice == "CBC":
    cipher = AES.new(key, AES.MODE_CBC, iv)
else:
    cipher = AES.new(key, AES.MODE_ECB)

decrypted_message = cipher.decrypt(ciphertext)
decrypted_message = decrypted_message.rstrip(b' ')

current_directory = os.getcwd()
decrypted_file_path = os.path.join(current_directory, "mensagem_decifrada.txt")

with open(decrypted_file_path, "wb") as f_dec:
    f_dec.write(decrypted_message)

tkinter.messagebox.showinfo("Sucesso", f"A mensagem foi decifrada e salva em '{decrypted_file_path}'.")
```

O trecho de código trata da decifragem de uma mensagem no modo AES selecionado (CBC ou ECB). Inicialmente, a variável IV (Initialization Vector) é definida como None. Se o modo escolhido for CBC, o código solicita ao usuário que selecione um arquivo contendo o vetor IV. Caso o usuário não selecione um arquivo, um erro é gerado. Após carregar o vetor IV, o código utiliza a chave e o vetor apropriados para criar o objeto de cifragem com AES.new. Em seguida, a mensagem cifrada é decifrada e o preenchimento é removido, resultando na mensagem original. A mensagem decifrada é salva em um arquivo no diretório atual, e uma caixa de mensagem informa ao usuário que a operação foi bem-sucedida. Se ocorrer um erro durante o processo, uma mensagem de erro é exibida ao usuário.

3. Geração de Chaves

A função que implementa a parte do projeto que trata da geração de chaves segue o mesmo padrão da cifragem e decifragem, está em um arquivo .py onde contem o menu que é chamado a partir do menu implementado no main.py, os menus serão mostrados em seções seguintes.

```python
def generate_keys():
    try:
        # Obtém o tamanho da chave
        key_size = int(key_size_var.get())

        if key_size not in [1024, 2048]:
            raise ValueError("O tamanho da chave deve ser 1024 ou 2048 bits.")

        # Gera as chaves
        private_key = RSA.generate(key_size)
        private_key_pem = private_key.export_key()

        public_key = private_key.publickey()
        public_key_pem = public_key.export_key()

        # Salva as chaves em arquivos
        with open("chave_privada.pem", "wb") as f_priv:
            f_priv.write(private_key_pem)

        with open("chave_publica.pem", "wb") as f_pub:
            f_pub.write(public_key_pem)

        tk.messagebox.showinfo("Sucesso", "Chaves geradas e salvas com sucesso!")

    except Exception as e:
        tk.messagebox.showerror("Erro", str(e))
# Variável para armazenar o tamanho da chave
key_size_var = tk.StringVar(value="1024")
```

A função que gera as chaves inicia pedindo pro usuário escolher, ele verifica se o tamanho escolhido é 1024 ou 2048, após essa verificação ele gera uma chave privada e uma pública usando RSA e exporta com formato .pem pro diretório raiz do projeto.

4. Criação/Validação de Assinatura Utilizando RSA

Seguindo na implementação das funções, esse tópico trata da Criação/Validação Utilizando RSA. O código necessita do funcionamento do código anterior pois ele precisa da entrada das chaves públicas e privadas.

```
def sign_rsa():
    try:
        file_to_sign = filedialog.askopenfilename(title="Selecione o arquivo a ser assinado")
        if not file_to_sign:
            raise FileNotFoundError("Arquivo a ser assinado não foi selecionado.")

        private_key_file = filedialog.askopenfilename(title="Selecione o arquivo com a chave privada (.pem):")
        if not private_key_file:
            raise FileNotFoundError("Arquivo com a chave privada não foi selecionado.")

        sha_version = simpledialog.askstring("Versão do SHA-2", "Escolha a versão do SHA-2 (256, 384, 512):")
        if sha_version not in ["256", "384", "512"]:
            raise ValueError("Versão inválida do SHA-2. Escolha entre 256, 384 ou 512.")

        output_format = simpledialog.askstring("Formato de Saída", "Escolha o formato de saída (Hex ou Base64):").upper()
        if output_format not in ["HEX", "BASE64"]:
            raise ValueError("Formato de saída inválido. Escolha entre Hex ou Base64.")

        with open(private_key_file, "rb") as f_priv:
            private_key = RSA.import_key(f_priv.read())

        with open(file_to_sign, "rb") as f_file:
            message = f_file.read()
```

Na função de Assinatura é possível o usuário fazer algumas configurações de parâmetros, primeiramente ele entra com o arquivo a ser assinado, após isso ele entra com a chave privada no formato “.pem” , ele pode também escolher a versão do SHA-2 e o formato de saída, se é Hex ou Base64.

```
        if sha_version == "256":
            h = SHA256.new(message)
        elif sha_version == "384":
            h = SHA384.new(message)
        elif sha_version == "512":
            h = SHA512.new(message)

        signature = pkcs1_15.new(private_key).sign(h)

        if output_format == "HEX":
            signature_out = binascii.hexlify(signature).decode()
            filename = "assinatura_hex.txt"
        else:
            signature_out = base64.b64encode(signature).decode()
            filename = "assinatura_base64.txt"

        with open(filename, "w") as f_sign:
            f_sign.write(signature_out)

        messagebox.showinfo("Sucesso", f"Assinatura gerada e salva em {filename} com sucesso!")

    except Exception as e:
        messagebox.showerror("Erro", str(e))
```

Após a entrada do usuário o arquivo é assinado utilizando RSA e é gerado um arquivo de texto com a assinatura, que será utilizado na verificação.

```
def verify_rsa_signature():
    try:
        signature_file = filedialog.askopenfilename(title="Selecione o arquivo com a assinatura")
        if not signature_file:
            raise FileNotFoundError("Arquivo com a assinatura não foi selecionado.")

        file_to_verify = filedialog.askopenfilename(title="Selecione o arquivo a ser verificado")
        if not file_to_verify:
            raise FileNotFoundError("Arquivo a ser verificado não foi selecionado.")

        public_key_file = filedialog.askopenfilename(title="Selecione o arquivo com a chave pública (.pem):")
        if not public_key_file:
            raise FileNotFoundError("Arquivo com a chave pública não foi selecionado.")

        sha_version = simpledialog.askstring("Versão do SHA-2", "Escolha a versão do SHA-2 (256, 384, 512):")
        if sha_version not in ["256", "384", "512"]:
            raise ValueError("Versão inválida do SHA-2. Escolha entre 256, 384 ou 512.")

        output_format = simpledialog.askstring("Formato da Assinatura", "Formato da assinatura (Hex ou Base64):").upper()
        if output_format not in ["HEX", "BASE64"]:
            raise ValueError("Formato de assinatura inválido. Escolha entre Hex ou Base64.")

        with open(public_key_file, "rb") as f_pub:
            public_key = RSA.import_key(f_pub.read())

        with open(signature_file, "r") as f_sign:
            signature_encoded = f_sign.read()
```

Na verificação o usuário entra com as configurações necessárias, no caso, as mesmas que ele entrou para assinar o documento.

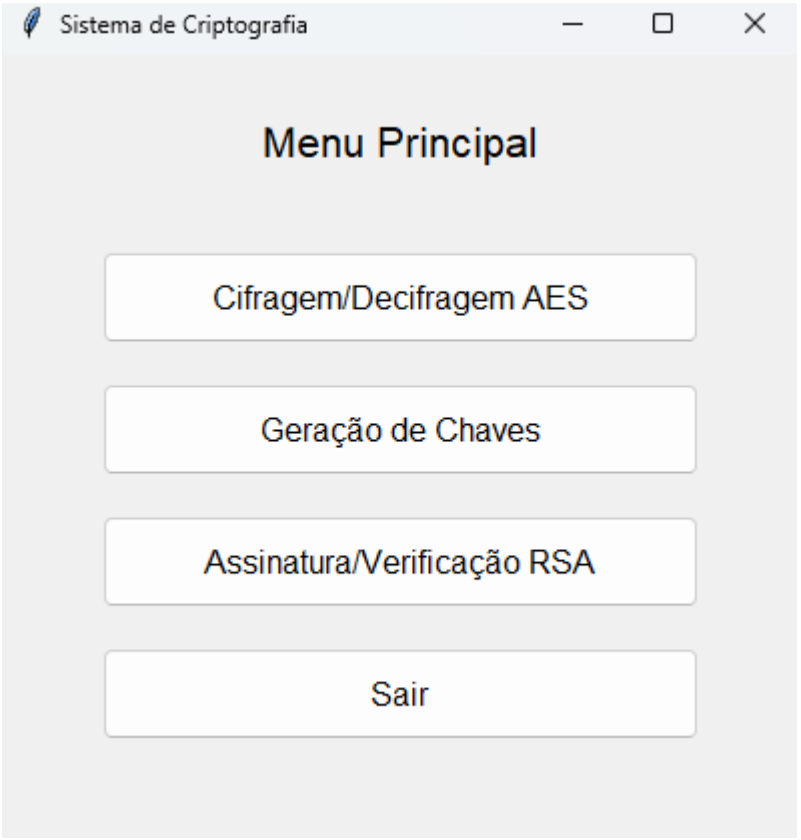
```
        signature = binascii.unhexlify(signature_encoded) if output_format == "HEX" else base64.b64decode(signature_encoded)

        with open(file_to_verify, "rb") as f_file:
            message = f_file.read()

        if sha_version == "256":
            h = SHA256.new(message)
        elif sha_version == "384":
            h = SHA384.new(message)
        elif sha_version == "512":
            h = SHA512.new(message)

        pkcs1_15.new(public_key).verify(h, signature)
        messagebox.showinfo("Sucesso", "A assinatura é válida!")

    except (ValueError, TypeError):
        messagebox.showerror("Erro", "A assinatura é inválida!")
    except Exception as e:
        messagebox.showerror("Erro", str(e))
```

Na verificação é feita a entrada com a chave pública e é verificado se a assinatura é válida ou inválida.

- Testes

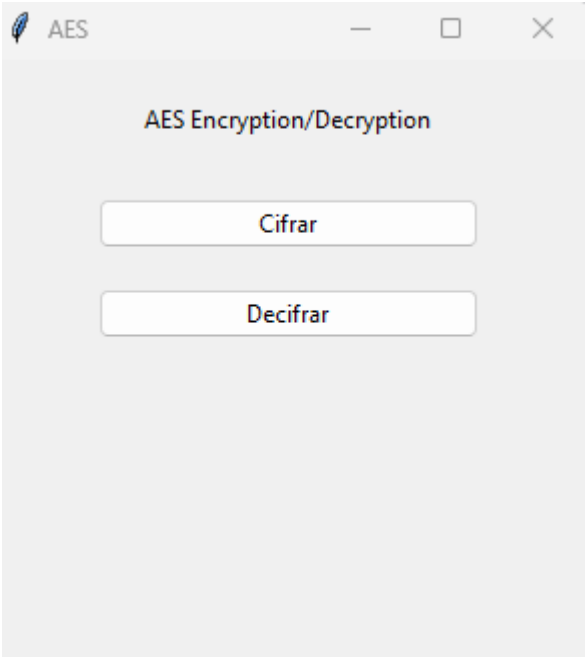
1. Main

   Todos a execução do programa tem que ser feita executando o arquivo main.py, ele que gerencia qual menu de aplicação irá executar através da escolha dos botoes da interface.

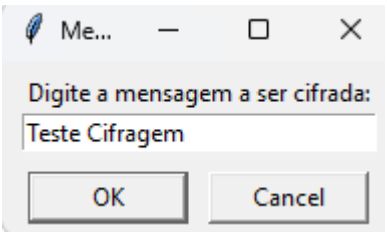


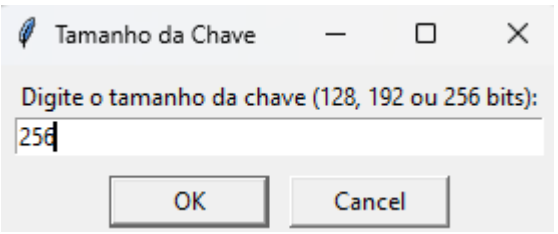
2. Cifragem/Decifragem utilizando AES

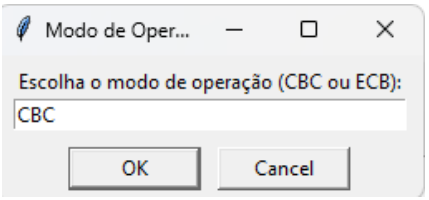
   Após executar o arquivo main e clica em Cifragem/Decifragem AES é possível abrir o menu especifico da função.

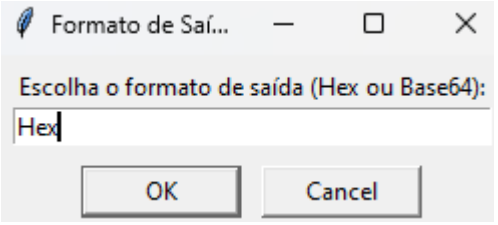


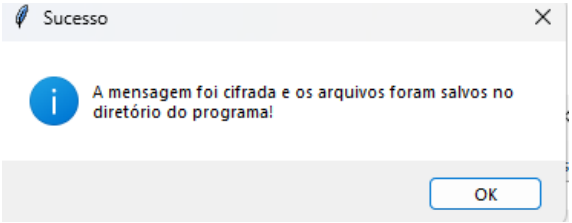
   Clicando em Cifrar é possível o usuário entrar com os parâmetros para a cifragem. Esses parâmetros estão dispostos nas imagens abaixo:











Após ser feito a cifragem, o programa já gera a chave e o vetor de inicialização (Pois foi escolhido o modo CBC) que serão usados na deciifragem.

chave_aes.txt

```
1   a79c65ae7d553e745861f49078355a93fe8720293059265ba2e846284658d1eb
```

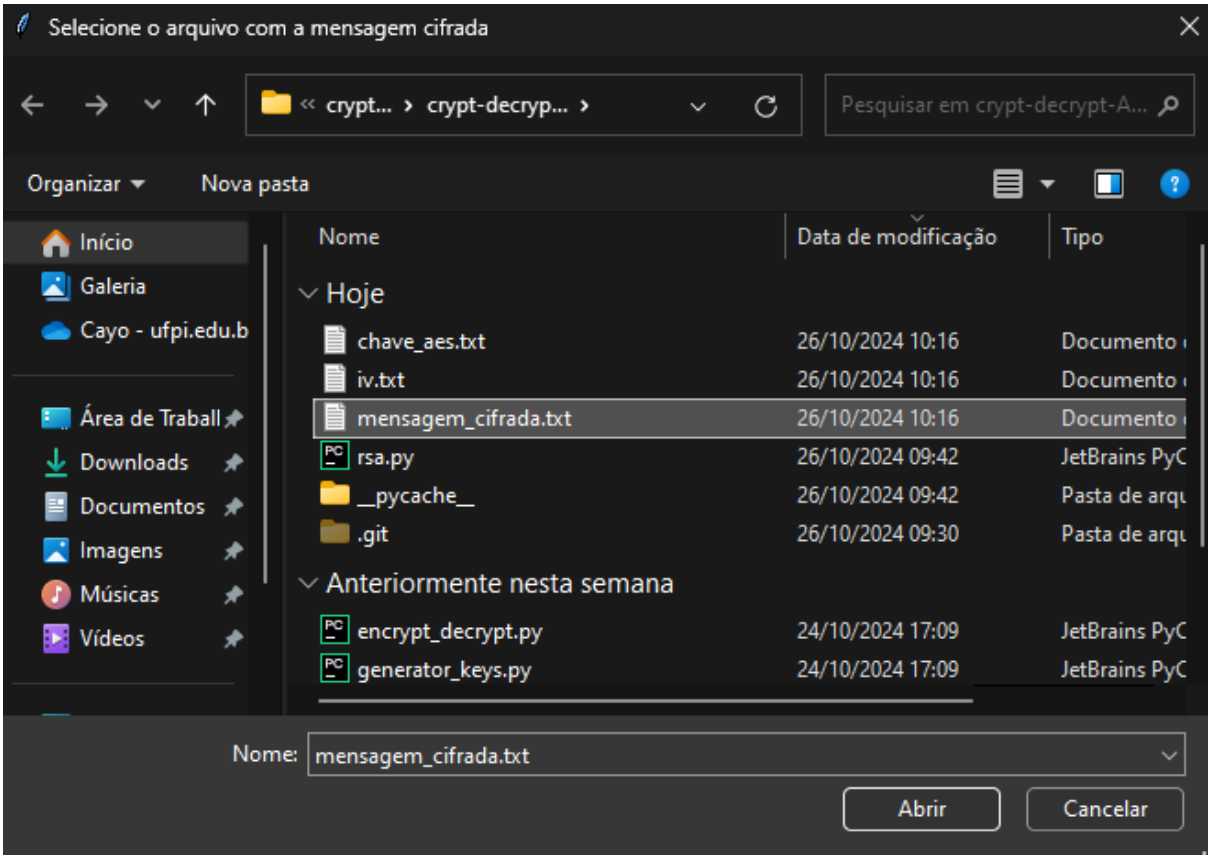
iv.txt

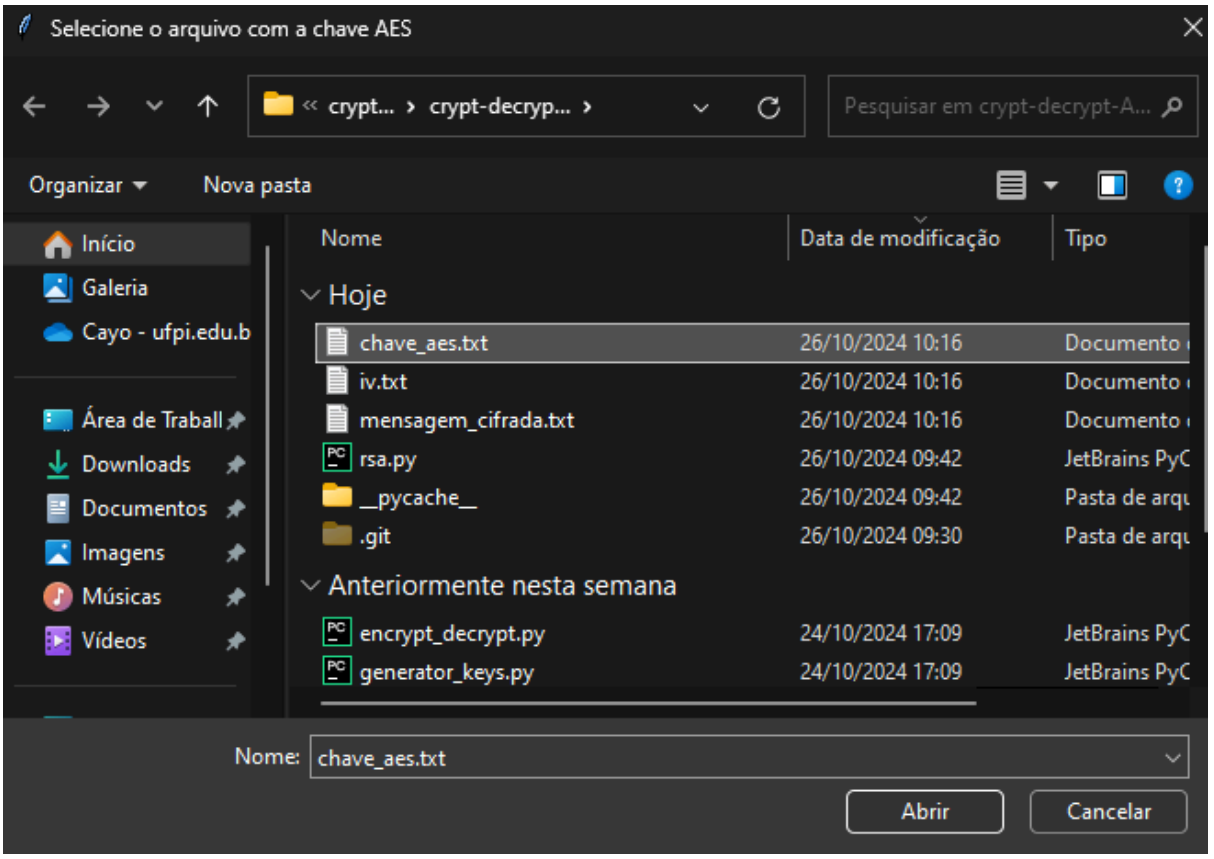
```
1   f58a257991b87b9abdfb9644419c8c8c
```

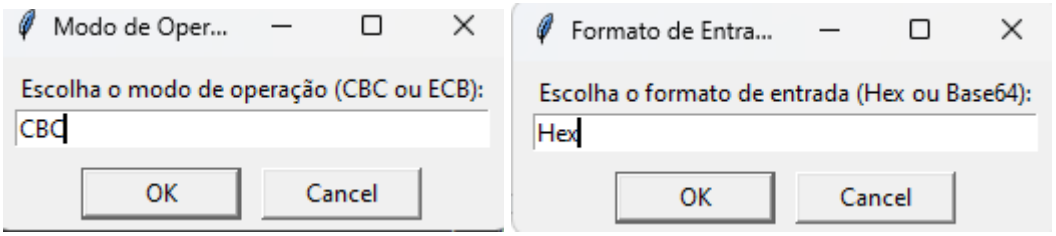
mensagem_cifrada.txt

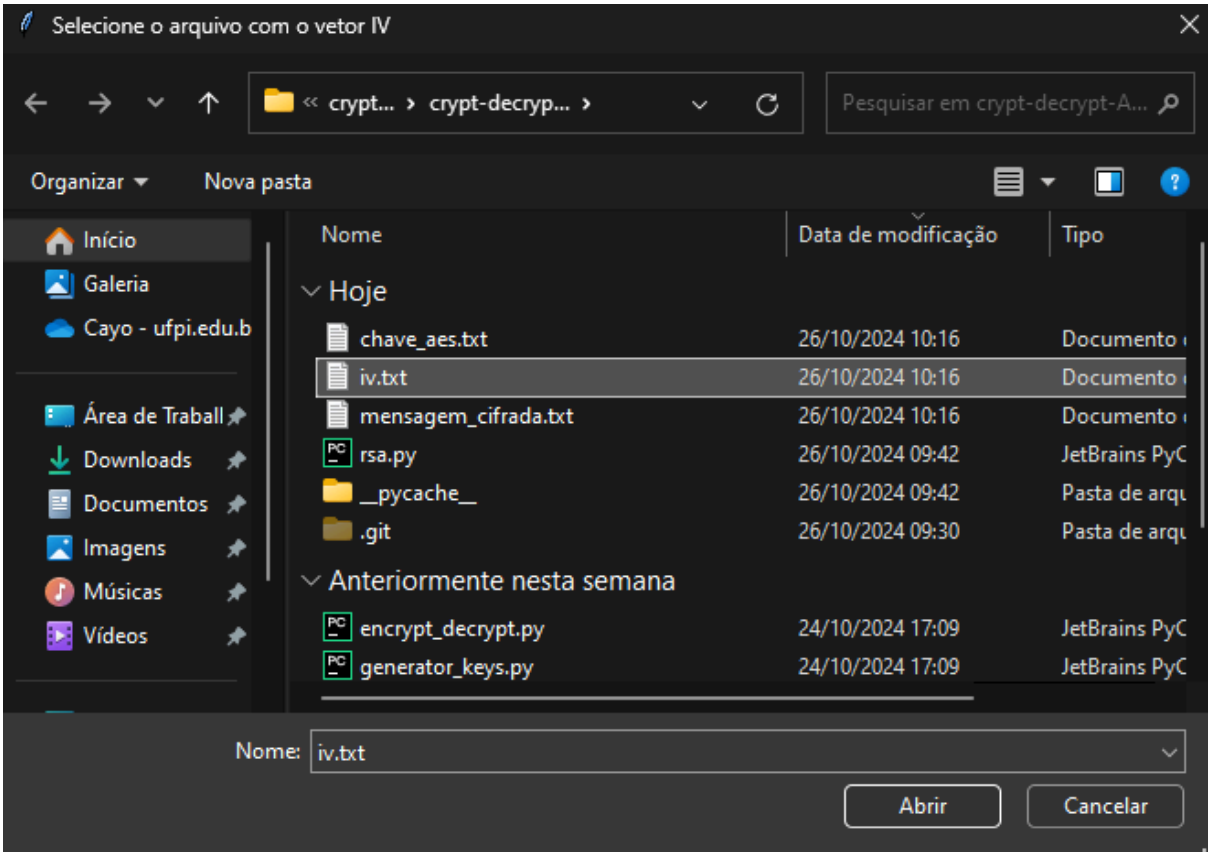
```
1   b1d95b94d97eaa978b6cefd2140123bb
```

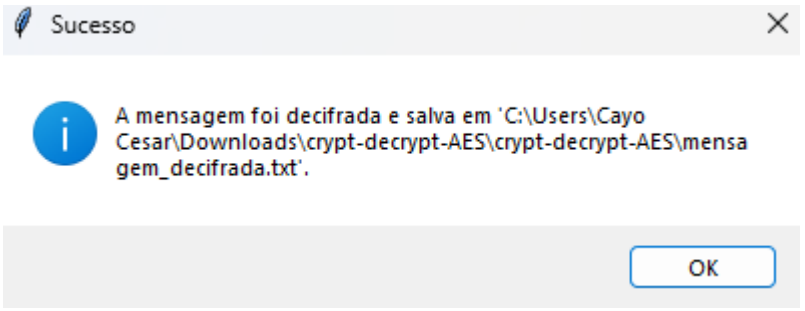
Pulando pra decifragem temos que entrar com os arquivos e escolher os mesmos parâmetros da cifragem.

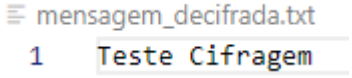

Selecione o arquivo com a mensagem cifrada
crypt... › crypt-decryp... ›
Pesquisar em crypt-decrypt-A...
Organizar
Nova pasta
Início
Galeria
Cayo - ufpi.edu.b
Área de Traball
Downloads
Documentos
Imagens
Músicas
Vídeos
Nome
Data de modificação
Tipo
Hoje
chave_aes.txt 26/10/2024 10:16 Documento
iv.txt 26/10/2024 10:16 Documento
mensagem_cifrada.txt 26/10/2024 10:16 Documento
rsa.py 26/10/2024 09:42 JetBrains PyC
__pycache__ 26/10/2024 09:42 Pasta de arqu
.git 26/10/2024 09:30 Pasta de arqu
Anteriormente nesta semana
encrypt_decrypt.py 24/10/2024 17:09 JetBrains PyC
generator_keys.py 24/10/2024 17:09 JetBrains PyC
Nome: mensagem_cifrada.txt
Abrir
Cancelar


Selecione o arquivo com a chave AES
crypt... › crypt-decryp... ›
Pesquisar em crypt-decrypt-A...
Organizar
Nova pasta
Início
Galeria
Cayo - ufpi.edu.b
Área de Traball
Downloads
Documentos
Imagens
Músicas
Vídeos
Nome
Data de modificação
Tipo
Hoje
chave_aes.txt 26/10/2024 10:16 Documento
iv.txt 26/10/2024 10:16 Documento
mensagem_cifrada.txt 26/10/2024 10:16 Documento
rsa.py 26/10/2024 09:42 JetBrains PyC
__pycache__ 26/10/2024 09:42 Pasta de arqu
.git 26/10/2024 09:30 Pasta de arqu
Anteriormente nesta semana
encrypt_decrypt.py 24/10/2024 17:09 JetBrains PyC
generator_keys.py 24/10/2024 17:09 JetBrains PyC
Nome: chave_aes.txt
Abrir
Cancelar





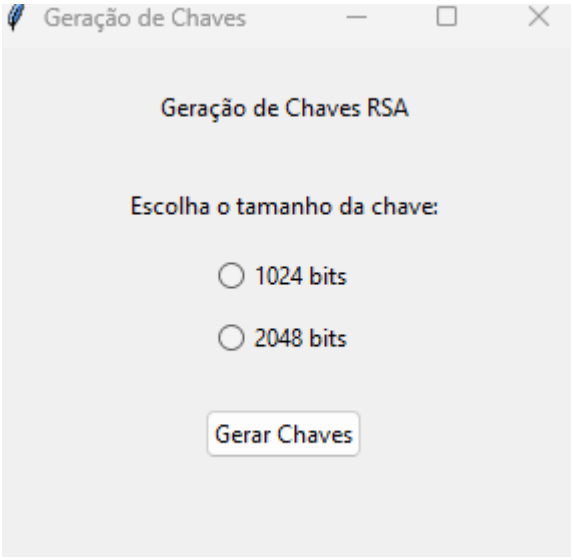


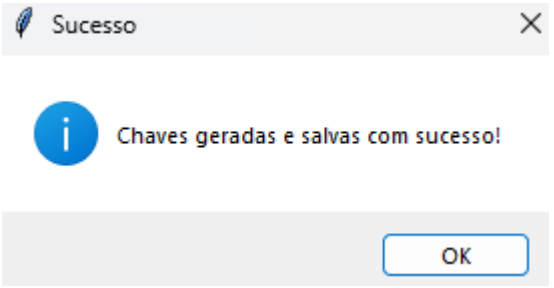


Após toda a execução é gerado o arquivo decifrado, idêntico ao que o usuário entrou no início da cifragem.

3. Geração de Chaves

A função de geração de chaves permite ao usuário escolher o tamanho das chaves (1024 ou 2048 bits) através da interface gráfica.





Depois da execução as chaves são salvas no diretório raiz do projeto.

chave_privada.pem

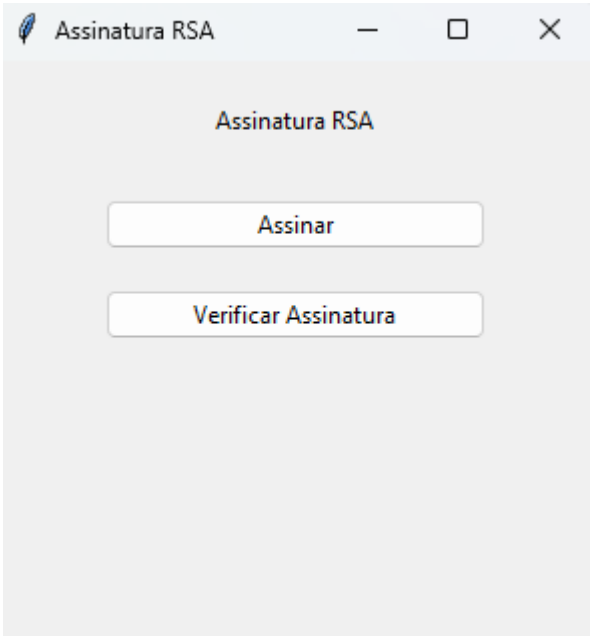
```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDgGgJuNjivHQMrs1eIUzBILvNDL4zaewtdXlPl7vIBUJG/QIG0
BhwlZxNKa3WGNhs8sAxtFJmGePo96yCCRkahFmOsvEAQIfIgsJ3+l/lphS4zHeFt
uWtPAWTFDLx+7D5bQrTeCxgITWbGkUL066NJ1i245oNX+pAixvIyTGabPQIDAQAB
AoGAVP/YgygCuPy2mqkz3qu66O4R655J/QZliZ0BEnnehbwjA9j6OSeGHnOpn/UB
Rw4Xp52YoPkKwDP7qmeX1YHFQJdUpzDw/tgcXGkYB+NP35asp9nNCtpYmqEIiK0Q
KLIvncDw2eRnRe4sl3H8zRvN37uq0m4e7IA99nnzXc2a+8sCQQDjUdjVbhKZTPAe
wScrEpTKp1y5vmFKbyVr21u0cWmo4iGGVocuiRkjMvCwn/zNQDTW4velgx027CRY
J84PcN9nAkEA/GA4oy5+2n36KYWfOhxmlA78bgucvsjyfrVAPJN2/ofjM0v+byyS
stCGGlkV2EYOA6h42shaSW5xitH0Df1duwJACdjm0sAU8C0kDcECSFxis7CyVJ2J
q2bLMTPschGjqRw6pY4kBwELZLn1MUGM+vs4DsMARVw7rMBoBqWayJJ0KwJBAIS7
ho4sUgNP5lYYNNo3Pav5BcJK0TF2NjN12EcDt72+cjv0Z+jIpI7vJNiOqfKJRa2e
2ZEXdoTQVV8RdS42ZykCQDLoFWyL/bhrsXOsjSamJFbDQD0KKod2rZ94XS6GWAEM
mTPDOpRvnQsCqwFOzikvoRckRK/AR3k+DOWAbBgbkuk=
-----END RSA PRIVATE KEY-----
```

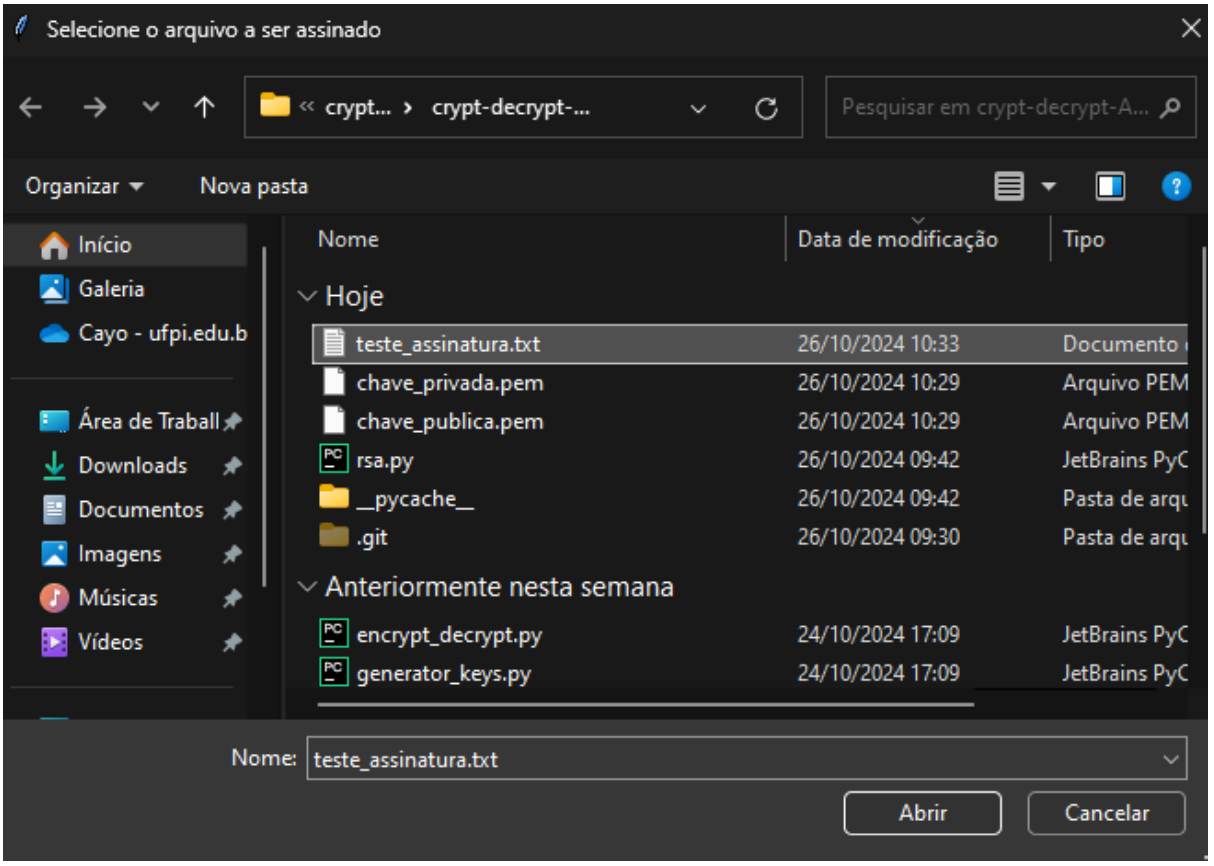
chave_publica.pem

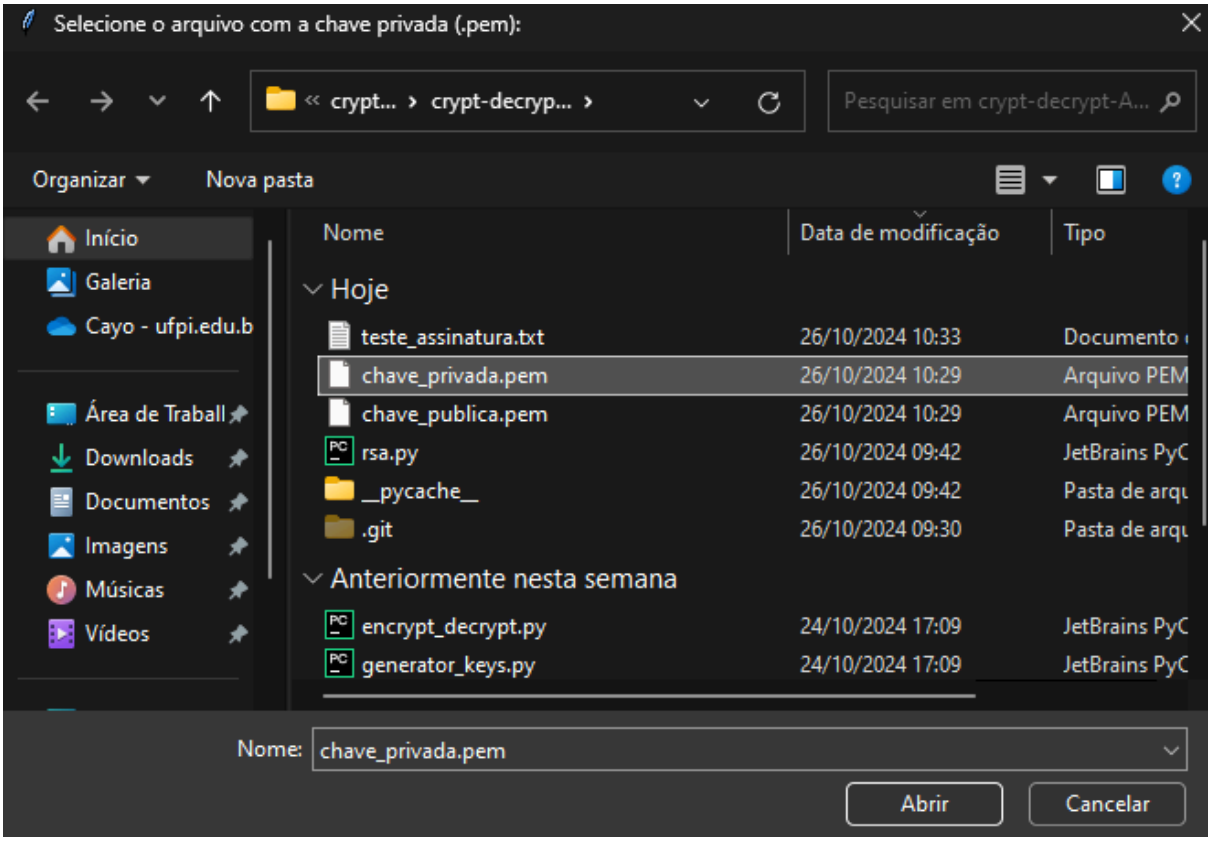
```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDgGgJuNjivHQMrs1eIUzBILvND
L4zaewtdXlPl7vIBUJG/QIG0BhwlZxNKa3WGNhs8sAxtFJmGePo96yCCRkahFmOs
vEAQIfIgsJ3+l/lphS4zHeFtuWtPAWTFDLx+7D5bQrTeCxgITWbGkUL066NJ1i24
5oNX+pAixvIyTGabPQIDAQAB
-----END PUBLIC KEY-----
```

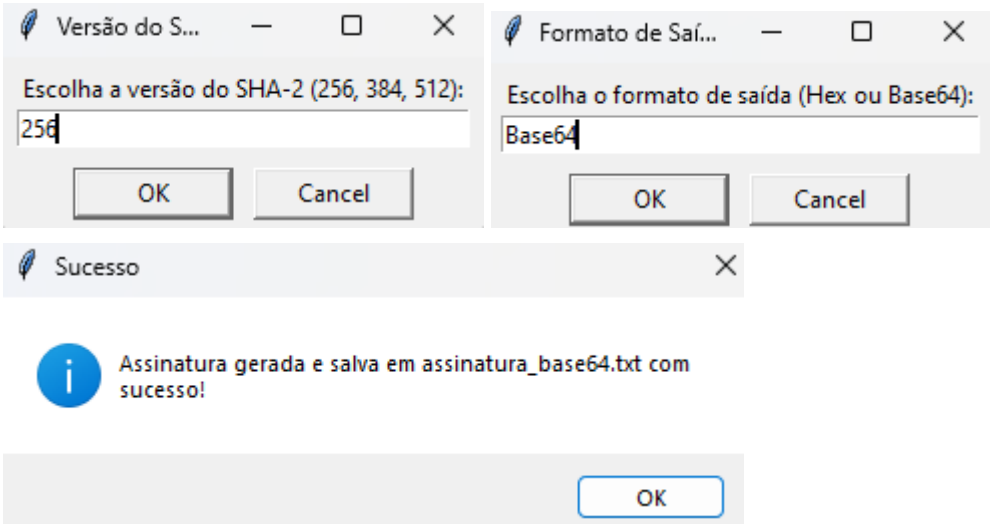
4. Assinatura/Verificação utilizando RSA

   A assinatura e verificação também tem um menu proprio, que permite vc escolher se quer fazer a assinatura ou a verificação da mesma, ela utiliza as chaves que são geradas na função de geração de chaves.

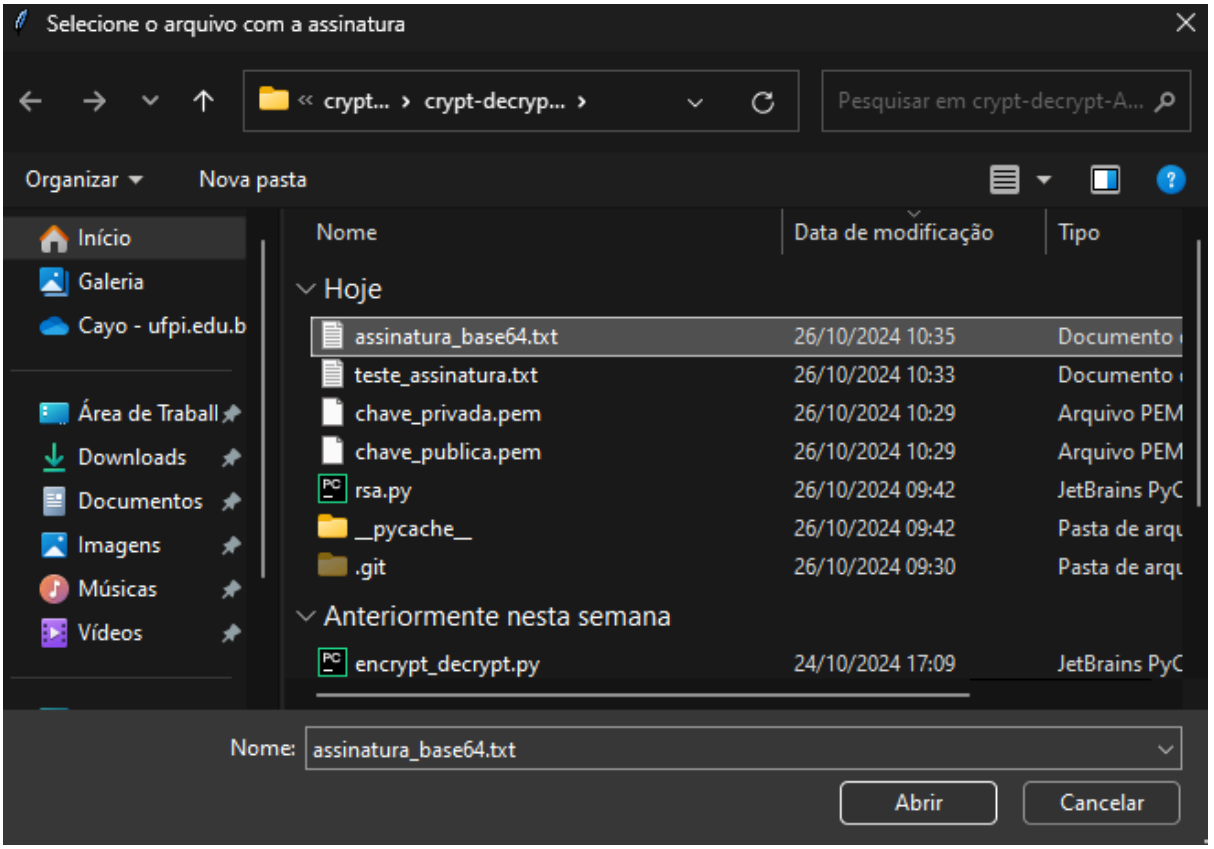


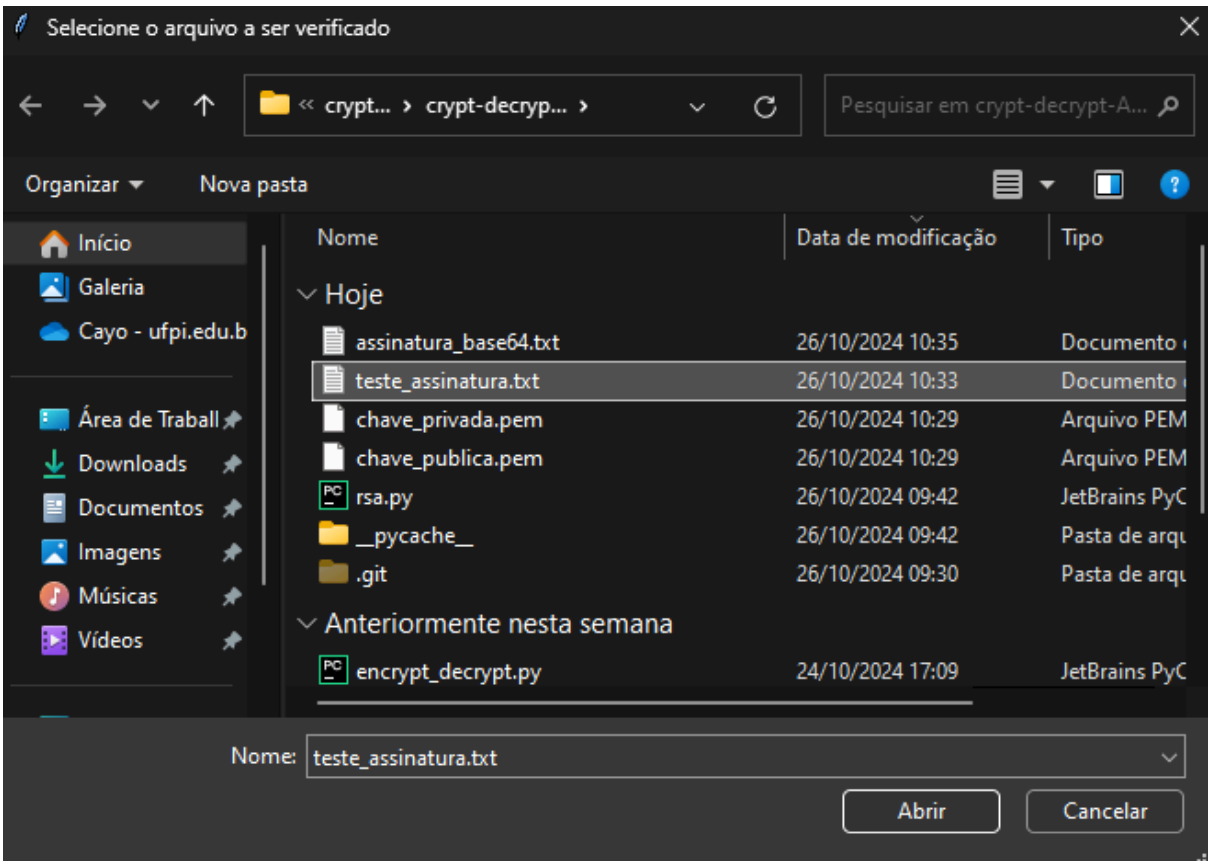

Selecione o arquivo a ser assinado
crypt... › crypt-decrypt-...
Pesquisar em crypt-decrypt-A...
Organizar
Nova pasta
Início
Galeria
Cayo - ufpi.edu.b
Área de Traball
Downloads
Documentos
Imagens
Músicas
Vídeos
Nome
Data de modificação
Tipo
Hoje
teste_assinatura.txt 26/10/2024 10:33 Documento
chave_privada.pem 26/10/2024 10:29 Arquivo PEM
chave_publica.pem 26/10/2024 10:29 Arquivo PEM
rsa.py 26/10/2024 09:42 JetBrains PyC
__pycache__ 26/10/2024 09:42 Pasta de arqu
.git 26/10/2024 09:30 Pasta de arqu
Anteriormente nesta semana
encrypt_decrypt.py 24/10/2024 17:09 JetBrains PyC
generator_keys.py 24/10/2024 17:09 JetBrains PyC
Nome: teste_assinatura.txt
Abrir
Cancelar

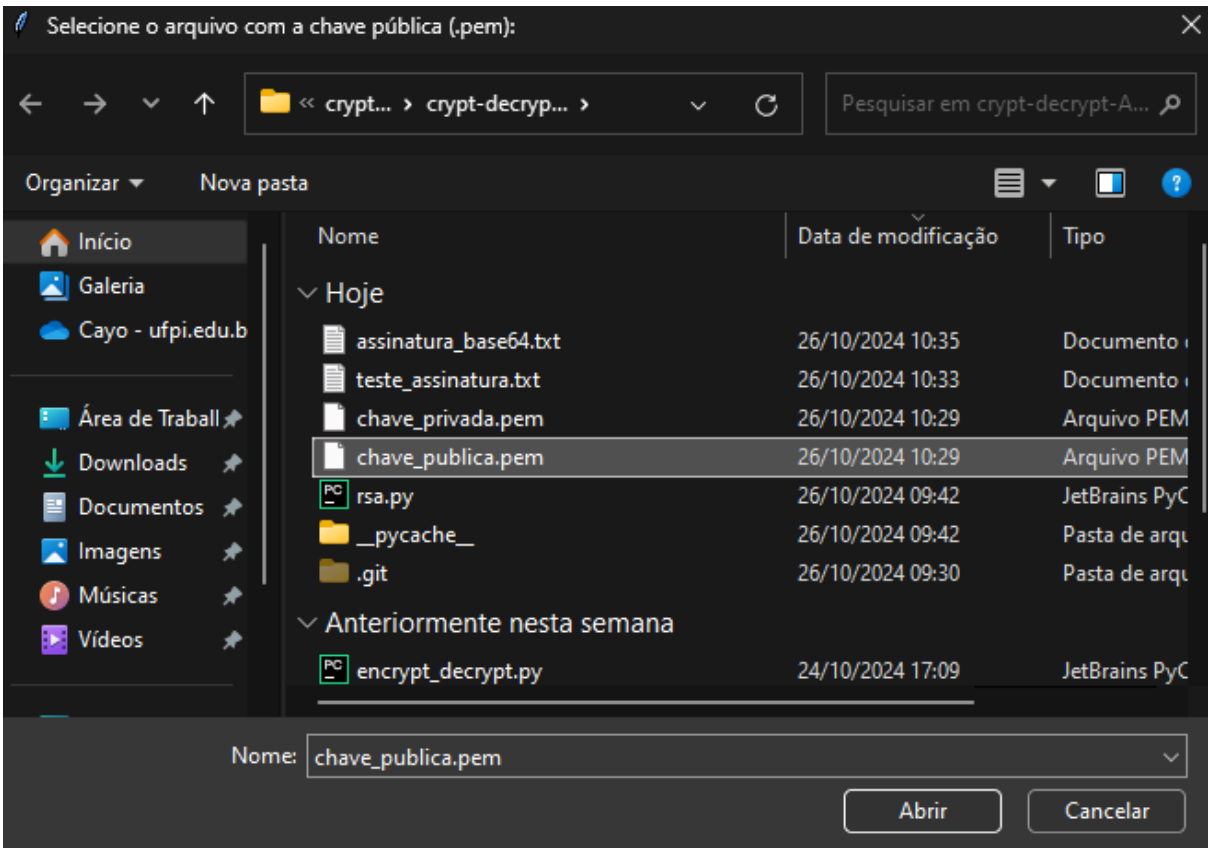

Selecione o arquivo com a chave privada (.pem):
crypt... › crypt-decryp... ›
Pesquisar em crypt-decrypt-A...
Organizar
Nova pasta
Início
Galeria
Cayo - ufpi.edu.b
Área de Traball
Downloads
Documentos
Imagens
Músicas
Vídeos
Nome
Data de modificação
Tipo
Hoje
teste_assinatura.txt 26/10/2024 10:33 Documento
chave_privada.pem 26/10/2024 10:29 Arquivo PEM
chave_publica.pem 26/10/2024 10:29 Arquivo PEM
rsa.py 26/10/2024 09:42 JetBrains PyC
__pycache__ 26/10/2024 09:42 Pasta de arqu
.git 26/10/2024 09:30 Pasta de arqu
Anteriormente nesta semana
encrypt_decrypt.py 24/10/2024 17:09 JetBrains PyC
generator_keys.py 24/10/2024 17:09 JetBrains PyC
Nome: chave_privada.pem
Abrir
Cancelar

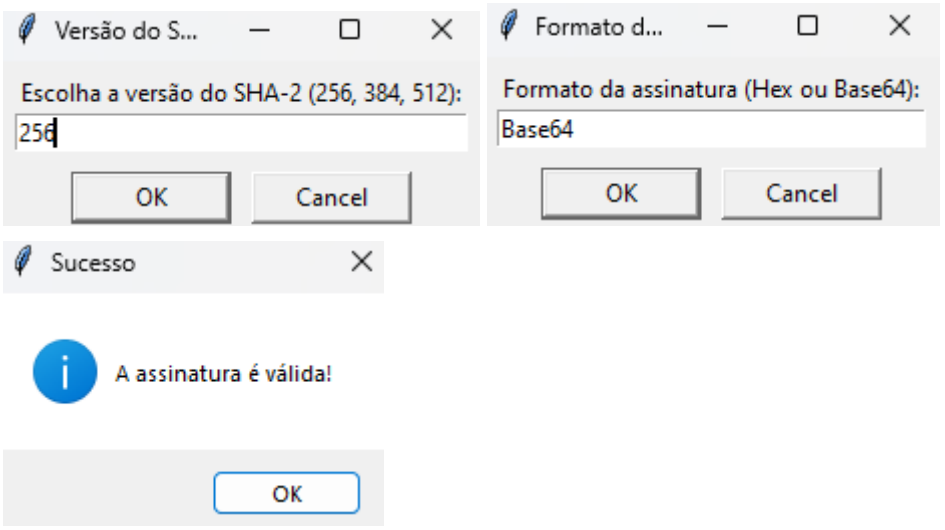


Para verificar a assinatura é necessário entrar com a assinatura, o arquivo assinado, a chave pública e as mesmas configurações de parâmetros que foram feitos na assinatura.




Selecione o arquivo a ser verificado
crypt... › crypt-decryp... ›
Pesquisar em crypt-decrypt-A...
Organizar
Nova pasta
Início
Galeria
Cayo - ufpi.edu.b
Área de Traball
Downloads
Documentos
Imagens
Músicas
Vídeos
Nome
Data de modificação
Tipo
Hoje
assinatura_base64.txt 26/10/2024 10:35 Documento
teste_assinatura.txt 26/10/2024 10:33 Documento
chave_privada.pem 26/10/2024 10:29 Arquivo PEM
chave_publica.pem 26/10/2024 10:29 Arquivo PEM
rsa.py 26/10/2024 09:42 JetBrains PyC
__pycache__ 26/10/2024 09:42 Pasta de arqu
.git 26/10/2024 09:30 Pasta de arqu
Anteriormente nesta semana
encrypt_decrypt.py 24/10/2024 17:09 JetBrains PyC
Nome: teste_assinatura.txt
Abrir
Cancelar


Selecione o arquivo com a chave pública (.pem):
crypt... › crypt-decryp... ›
Pesquisar em crypt-decrypt-A...
Organizar
Nova pasta
Início
Galeria
Cayo - ufpi.edu.b
Área de Traball
Downloads
Documentos
Imagens
Músicas
Vídeos
Nome
Data de modificação
Tipo
Hoje
assinatura_base64.txt 26/10/2024 10:35 Documento
teste_assinatura.txt 26/10/2024 10:33 Documento
chave_privada.pem 26/10/2024 10:29 Arquivo PEM
chave_publica.pem 26/10/2024 10:29 Arquivo PEM
rsa.py 26/10/2024 09:42 JetBrains PyC
__pycache__ 26/10/2024 09:42 Pasta de arqu
.git 26/10/2024 09:30 Pasta de arqu
Anteriormente nesta semana
encrypt_decrypt.py 24/10/2024 17:09 JetBrains PyC
Nome: chave_publica.pem
Abrir
Cancelar



Nesse caso de uso, a assinatura está válida, porém se houver qualquer mudança em algum parâmetro ou no arquivo de assinatura a verificação de assinatura é acusada como inválida.