



MINISTÉRIO DA EDUCAÇÃO
UTFPR – UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO



Professor: Lucas Hiera Sampaio

Aluno: Mateus Yi Muraoka

Atividade 1

EC38D – Segurança e Auditoria de Sistema – C81(2022/1)

Comunicação Autêntica, Íntegra e Confidencial

CORNÉLIO PROCÓPIO

ABRIL – 2022

Sumário

INTRODUÇÃO	1
MATERIAIS E METODOS	2
MATERIAIS	2
METODOS	5
CONCLUSÃO	11
REFERÊNCIA	11

1. INTRODUÇÃO

Nos dias atuais, informações devem ser manter sempre seguras e como muitas destas informações necessitam se locomover pelos meios de comunicação há necessidade de sua segurança e nisto entra a criptografia é forma de garantir que as estas informações que necessitam estrar protegidas sejam feitas de uma forma mais segura. Onde a criptografia aplica algoritmos que codificam estas informações e faz com que elas estejam em formatos de algoritmos que não sejam decifrados de maneira fácil.

Onde neste trabalho será buscado alterar o código fonte disponibilizado de maneira que as informações que serão transmitidas sejam criptografadas e não sejam capturadas pelo Wireshark.

2. MATERIAIS E METODOS

Foi disponibilizado para este trabalho 2 códigos fontes, como demonstrado na figura 1, e um documentos com instruções a serem seguidas, estes 2 códigos fontes citado anteriormente são um servidor (*Server.py*) e um cliente (*Client.py*), onde no servidor foi utilizado o IP 127.0.0.1 e foi operado na porta 5535.

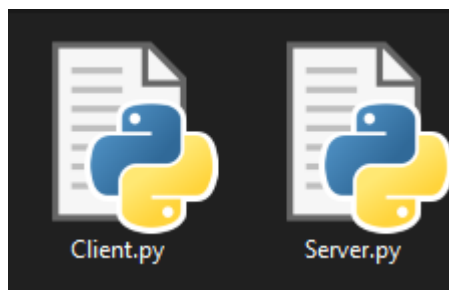


Figura 1 – Arquivos disponibilizados.

2.1. MATERIAIS

Foi utilizado linguagem Python na versão 3.10.4 no sistema operacional Windows 10 e o analisador de protocolos Wireshark para que fosse capturado a troca de mensagens entre 2 clientes e que passaram pelo servidor. Assim como descrito nas instruções, foi inicializado o Wireshark (Figura 2) antes do início dos testes com os arquivos em Python.

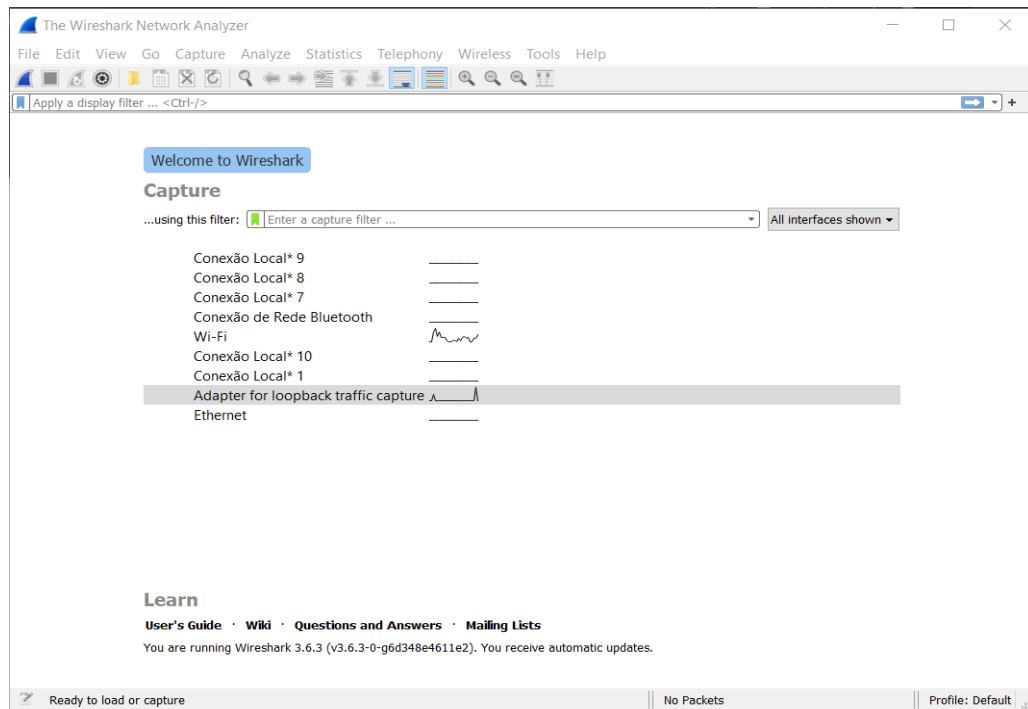


Figura 2 – Wireshark

Onde foi escolhido o modo “*Adapter for loopback traffic capture*”, que consiste na captura de pacotes na interface *loopback*. Após isto foi aberto o servidor e inicializado os clientes, como demonstrado na imagem 3.

<pre>PS C:\seg_aud_at1> python Server.py Server started on port 5535 [<socket.socket fd=376, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('0.0.0.0', 5535)>] ('127.0.0.1', 50481) <socket.socket fd=408, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 50481)> ('127.0.0.1', 50490) <socket.socket fd=412, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 50490)></pre>	<pre>PS C:\seg_aud_at1> pyth on Client.Py Starting client Enter the server IP >>127.0.0.1 Enter the server Destin ation Port >>5535 Connecting Connected Enter the User Name to be Used >>User 1 Starting service >></pre>	<pre>PS C:\seg_aud_at1> pyth on Client.Py Starting client Enter the server IP >>127.0.0.1 Enter the server Destin ation Port >>5535 Connecting Connected Enter the User Name to be Used >>User 2 Starting service >> </pre>
---	---	--

Figura 3 – Teste dos arquivos *Client* e *Server*.

```

PS C:\seg_aud_at1> python Server.py
Server started on port 5535
[<socket.socket fd=396, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('0.0.0.0', 5535)>]
('127.0.0.1', 50547)
<socket.socket fd=424, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 50547)>
('127.0.0.1', 50551)
<socket.socket fd=428, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 50551)>
Traceback (most recent call last):
  File "C:\seg_aud_at1\Server.py", line 56, in run
    if (str(s.getpeername()) == SENT_BY[items])
    :
KeyError: b'User 1: Salve'
Sending to ('127.0.0.1', 50551)
Sending to ('127.0.0.1', 50547)
Ignoring ('127.0.0.1', 50551)

PS C:\seg_aud_at1> python Client.py
Starting client
Enter the server IP
>>127.0.0.1
Enter the server Destination Port
>>5535
Connecting
Connected
Enter the User Name to be Used
>>User 1
Starting service
>>Salve
>>User 2: Ola
>>

PS C:\seg_aud_at1> python Client.py
Starting client
Enter the server IP
>>127.0.0.1
Enter the server Destination Port
>>5535
Connecting
Connected
Enter the User Name to be Used
>>User 2
Starting service
>>User 1: Salve
>>
Ola
>>

```

Figura 4 – Troca de mensagens entre o usuário 1 e usuário 2.

Então foi feita a troca de mensagens entre os usuários, como demonstrado na figura 4, e foi pausado a captura de dados no Wireshark, foi digitado na barra de filtro o código “tcp.port == 5535”, assim foi demonstrado somente os pacotes que continham o TCP 5535 (figura 5).

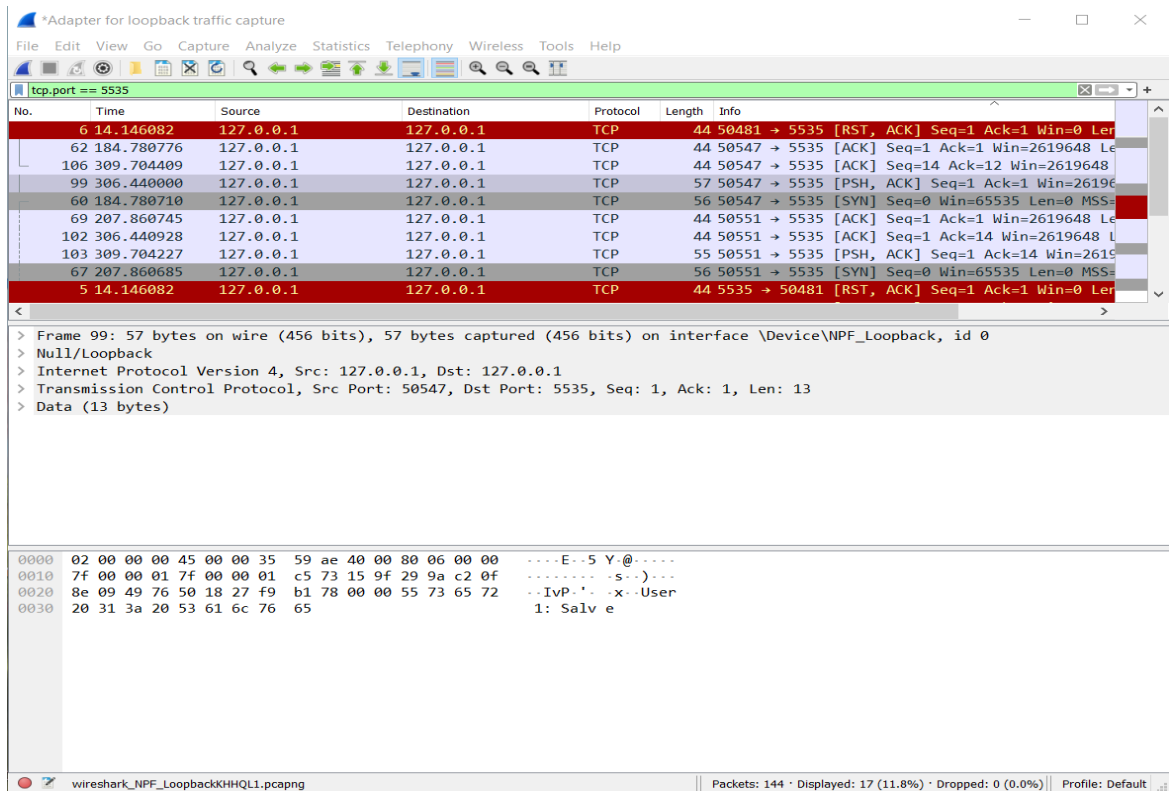


Figura 5 – Filtragem por TCP 5535.

Quando o pacote foi selecionado e a sequência de teclas foi digitada (CTRL + ALT + SHIFT + T), surgiu uma janela que demonstrava as mensagens que foram trocadas pelos usuários, figura 6.

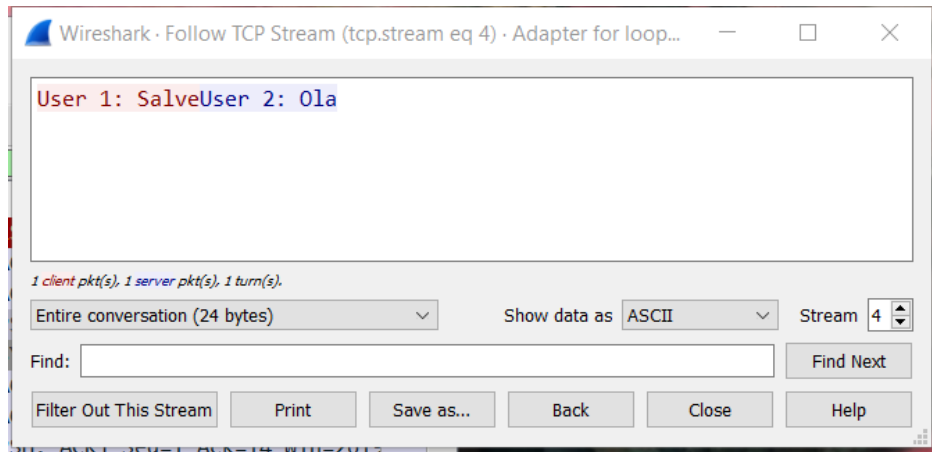


Figura 6 – Mensagem que foram captadas pelo Wireshark.

O intuito desta atividade 1 foi de garantir que todos os clientes pudessem ver as informações apresentadas na interface do seu terminal e que o fluxo TCP fosse completamente indiscernível para o Wireshark. Além disto foi necessário o compartilhamento das chaves simétricas utilizando a criptografia assimétrica, por meio de mensagens transmitidas e foi verificado a integridade e autenticidade das mensagens enviadas. O algoritmo simétrico que foi utilizado foi o Fernet e o algoritmo assimétrico foi Rsa ambos instalados no computador e importados nas bibliotecas iniciais dos arquivos necessários.

2.2. METODOS

A partir do arquivo “Client.py” foram criados 2 novos arquivos para os usuários sendo denominados “01Client.py” e “02Client.py”, onde o código fonte original foi alterado e configurado conforme será explicado abaixo no relatório.

01Client.py	05/05/2022 13:54	Arquivo Fonte Pyt...	10 KB
02Client.py	05/05/2022 13:54	Arquivo Fonte Pyt...	11 KB
Server.py	24/03/2022 15:12	Arquivo Fonte Pyt...	3 KB

Figura 7 – Arquivos criados e renomeados a partir do código fonte

Foi seguido os passos semelhantes ao citado em “materiais”, mas para que a conversa entre os clientes obtenha sucesso foi necessário que fossem executado o Server primeiro e após os códigos para cada cliente em curto período, visto que


```

# Cria o hash utilizando a chave simetrica com o algoritmo de SHA-1
hash1 = rsa.compute_hash(keySimetrica, 'SHA-1')
#Criando a assinatura do cliente
assinatura = rsa.sign_hash(hash1, privateKey1, 'SHA-1')
#Realiza o envio par o cliente 2
self.client(host,port,assinatura)
#Irá adiar o segmento de chamada que tiver em execução em segundos
time.sleep(5)

```

Figura 10 Criação de Hash.

O 02Client verificou se a assinatura é valida com utilização da chave pública do 01Client, e como a resposta foi positiva ele manda uma mensagem de confirmado e depois armazena em um arquivo, caso não fosse confirmado imprimiria uma mensagem de alerta avisando. No momento em o que 01Client recebeu este “confirmado”, demonstrado na figura 11, ele armazenou em um arquivo de texto e permitiu 01 compreendesse que a operação de “Handshake”(aperto de mão), foi finalizada e que a troca de mensagem fosse feita com segurança. O 02Client verificou se o “Handshake” foi finalizado de maneira segura com a utilização do arquivo “keySimetrica2.key”, onde esta armazenada a chave simétrica que foi recebida.

```

# criado um loop que tem como função receber e conferir a assinatura da chave simétrica
for item in read:
    try:
        b = item.recv(1024)
        if b != '':
            # Se recebe a assinatura do cliente 01
            assi = b
            # Ocorre a verificação se a assinatura corresponde a chave simétrica recebida
            # através da chave assimétrica publica do cliente 01
            if (rsa.verify(keySimetrica, assi, publicKey1)):
                # Se verificado a recepção da chave simétrica do cliente 01
                print('>> A chave simétrica foi recebida com sucesso << ')
                # Salva em um arquivo.key a chave simétrica
                sk = open('keySimetrica2.key', 'wb')
                sk.write(keySimetrica)
                sk.close()
                f = Fernet(keySimetrica)
                #Envia um "confirmado" para finalização do Handshake
                confirmado = "confirmado"
                self.receive.send(bytes(confirmado, encoding='utf8'))
                #Caso ocorra erro será imprimido um mensagem na tela
            else:
                print('>>Atenção! Chave simétrica não foi recebida com sucesso! <<')
                exit()

```

Figura 11 Handshake ente os entre o servidor e os usuários.

A seguir foi feita uma demonstração entre 2 usuárias fictícias em que elas estavam tendo uma conversa rotineira, como pode ser visto na figura 14 e com uma definição melhor na figura 15 e 16.

A quantidade de caracteres que foi definida para cada mensagem foi de no máximo 150 caracteres para este experimento, como demonstrado na figura 12. Caso a quantidade de caracteres seja maior que permitido o usuário receberá um aviso. Quando um dos clientes recebeu a mensagem, esta mensagem foi descriptografada pela chave simétrica recebida, sendo salva em outra variável, depois recebeu a assinatura e foi conferida utilizando a chave pública do cliente que enviou a mensagem.

Como durante a execução do teste a assinatura foi compatível, o nome do usuário e a mensagem foram demonstrados corretamente, caso não fosse compatível seria executado uma mensagem de erro avisando que a mensagem não foi recebida de forma integral.

```
# É criado um loop para envio de mensagem definida até 150 caracteres
while 1:
    msg = input('>>')
    #Verifica se a mensagem possui mais de 150 caracteres
    if(len(msg)>151):#151 pois começa em 0
        print("Atenção! Não é permitido enviar mensagem com mais de 150 caracteres!")
        break
    if msg == 'exit':
        break
    if msg == '':
        continue
    # Se concatena o valor do input de msg com o user_name definido e realiza o encode da msg
    msg = user_name + ":" + msg
    msgh = msg.encode()
```

Figura 12 Definição de quantidade de caracteres

No final da execução dos arquivos “Serve.py”, “01Client.py” e “02Client.py” na mesma pasta apareceu 3 novos arquivos comprovando que o “Handshake” ocorreu corretamente, como demonstrado na figura 13. Estes 3 novos arquivos deverão ser excluídos antes de uma nova execução, visto que podem causar erros caso estiverem na pasta ainda.




	Handshake.txt	05/05/2022 20:26	Documento de Te...	1 KB
	keySimetrica.key	05/05/2022 20:26	Arquivo KEY	1 KB
	keySimetrica2.key	05/05/2022 20:26	Arquivo KEY	1 KB

Figura 13 Arquivos criados e com informações armazenadas

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\seg_aud_at1> python Server.py
Server started on port 5535
[socket.socket fd=420, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('0.0.0.0', 5535)]

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\seg_aud_at1> python 01Client.py
>> Starting client <<
Digite o IP do servidor:
>> 127.0.0.1
Digite a porta de destino do Servidor:
>> 5535
>> Conectando... <<
>> Conectado <<

Entre com o nome de usuario que será usado:
>> Maria
>> Iniciando serviço... <<
>>Ola
>>Carol:Tudo bem?
>>
Tudo sim e vc?
>>Carol:To bem tbm?
>>

Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\seg_aud_at1> python 02Client.py
>> Starting client <<
Digite o IP do servidor:
>> 127.0.0.1
Digite a porta de destino do Servidor:
>> 5535
>> Conectando... <<
>> Conectado <<

Entre com o nome de usuario que será usado:
Carol
>> Iniciando serviço... <<
Aguardando Fim do Handshake...
>> A chave simétrica foi recebida com sucesso <<
>>Maria:Ola
>>
Tudo bem?
>>Maria:Tudo sim e vc?
>>
To bem tbm?

```

Figura 14 – Arquivo de conversa criptografada

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\seg_aud_at1> python 01Client.py
>> Starting client <<
Digite o IP do servidor:
>> 127.0.0.1
Digite a porta de destino do Servidor:
>> 5535
>> Conectando... <<
>> Conectado <<

Entre com o nome de usuario que será usado:
>> Maria
>> Iniciando serviço... <<
>>Ola
>>Carol:Tudo bem?
>>
Tudo sim e vc?
>>Carol:To bem tbm?
>>

```

Figura 15 – 01Client.py conversa

```

PS C:\seg_aud_at1> python 02Client.py
>> Starting client <<
Digite o IP do servidor:
>> 127.0.0.1
Digite a porta de destino do Servidor:
>> 5535
>> Conectando... <<

>> Conectado <<

Entre com o nome de usuario que será usado:
Carol
>> Iniciando serviço... <<
Aguardando Fim do Handshake...
>> A chave simétrica foi recebida com sucesso <<
>> Maria:Ola
>>
Tudo bem?
>> Maria:Tudo sim e vc?
>>
To bem tbm?
>>

```

Figura 16 – 02Client.py conversa

A ferramenta “Wireshark” foi utilizada para ver confirmar que as mensagens foram realmente criptografadas, como pode ser visto na figura 17 diferente da figura 6. Vale destacar que ambas as usuárias fictícias tiveram suas mensagens criptografadas de forma correta, e desta maneira o método utilizado para este trabalho ocorreu como planejado inicialmente.

The screenshot displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The main window is divided into three panes: Packet List, Packet Details, and Packet Bytes. The Packet List pane shows a list of captured packets, with packet 263 selected. The Packet Details pane shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The Packet Bytes pane shows the raw data of the selected packet, including a null terminator and a string 'Maria:Ola'.

Figura 17 – Wireshark e interceptação de pacote

3. CONCLUSÃO

Em virtude dos resultados obtidos ao final destes experimentos com os códigos fontes e documentos com instruções a serem seguidas, foi possível entender como é feita a criptografia que ocorre nos sistemas de comunicação de bate papos online como por exemplo o WhatsApp.

Neste trabalho, as dificuldades encontradas foram de utilizar uma linguagem de programação nunca antes mexido e também como incrementar e utilizar as chaves públicas privadas (criadas a partir da ajuda algoritmo Fernet) de forma conjunta com o Hash, além da utilização das chaves assimétricas a partir do algoritmo assimétrico Rsa. Logo foi necessário um desempenho maior de estudo do conteúdo de pesquisa e programação para que trabalho fosse concluído.

4. REFERÊNCIA

1. How to Encrypt and Decrypt Strings in Python. Disponível em: "<https://www.geeksforgeeks.org/how-to-encrypt-and-decrypt-strings-in-python/>". Acesso em: 09 de abr. 2021
2. CRIPTOGRAFIA. Guide to Python. Disponível em: "https://python-guide-pt-br.readthedocs.io/pt_BR/latest/scenarios/crypto.html" Acesso em: 14 de abr. 2021
3. STUMM JR, VALDIR. CALCULANDO O HASH DE STRINGS EM PYTHON. Pythonhelp. 2011. Disponível em: <https://pythonhelp.wordpress.com/tag/hashlib/>. Acesso em: 14 abr. 2022.
4. ESQUEMA DE ASSINATURA DIGITAL RSA USANDO PYTHON. Disponível em: "<https://acervolima.com/esquema-de-assinatura-digital-rsa-usando-python/#:~:text=O%20algoritmo%20RSA%20%C3%A9%20um,chave%20privada%20%C3%A9%20mantida%20privada.>". Acesso em: 15 abr. 2022.
5. How to Encryt and Decrypt Data in Python using Cryptography Library. Disponível em: "<https://devqa.io/encrypt-decrypt-data-python/>". Acesso em: 22 abr. 2022.
6. RSA Encryption and Decryption Disponível em: "<https://stackoverflow.com/questions/30056762/rsa-encryption-and-decryption-in-python>" Acesso em: 25 abr. 2022.