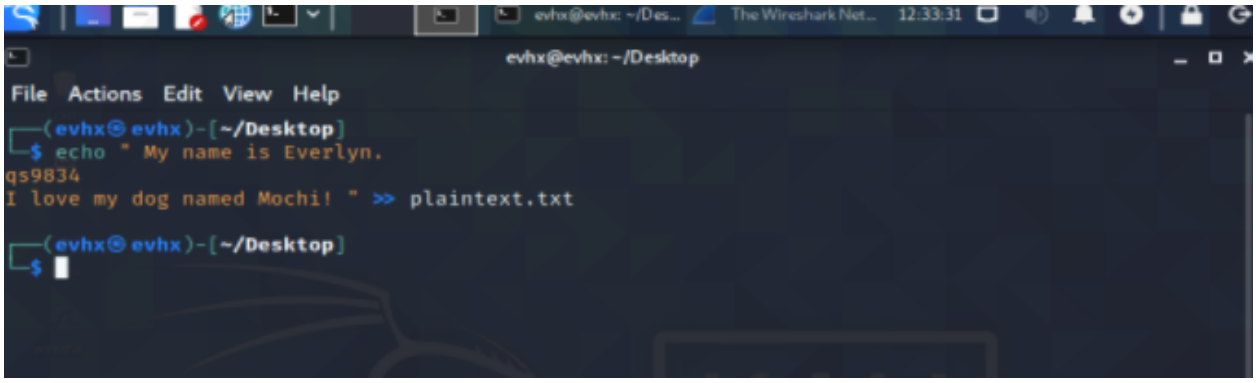
Everlyn Leon

Abstract
Analyzing the security mechanics in symmetric and asymmetric encryption utilizing Kali Linux, GPG, Netcat, Steghide and Md5sum to understand the constraints and use cases each provides.

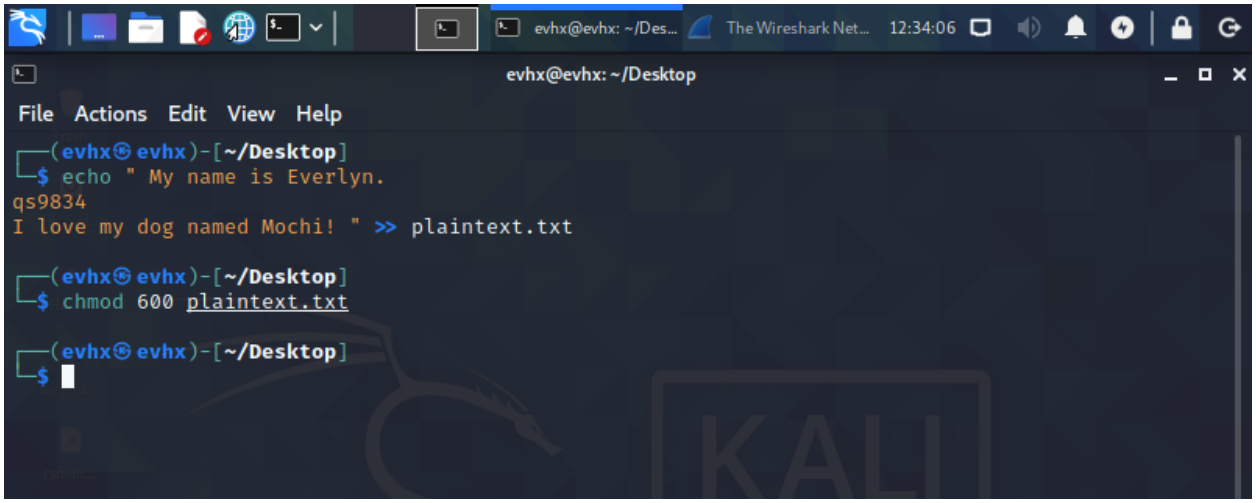
Introduction
Utilizing components used in a Cryptosystem, such as: Plaintext, Ciphertext, Encryption and Decryption Keys/Algorithms in symmetric and asymmetric encryption using numerous tools. GPG (GNU Privacy Guard) will be used to encrypt and decrypt files using public and private keys. Netcat will be used to transmit ASCII formatted encrypted text by creating a listener to receive data in a file. Steghide will be used to embed a text file into an image file, and using the same tool, the plaintext will be extracted from the image file as well. The Md5sum will be used to verify the integrity of the original image file and the image file with the embedded text file.

Summary of Results
On the Kali Linux machine, the terminal will be used for the commands used in creating the Cryptosystem components that will be used in symmetric and asymmetric encryption. To write text into a file use the command:
echo “[text]” >> plaintext.txt



The plaintext.txt file will appear on the desktop since the directory in the terminal was the desktop.

The permissions on the plaintext.txt file need to be changed so that only the owner of the file could have full read and write access. The ‘600’ number will be used because ‘6’ represents the permission to allow for both reading and writing under the owner, while ‘0’ represents the permissions given to the ‘group’ or ‘other’, which would be none. Full permissions would result in a ‘7’, by allowing for reading, writing and executing permissions, but since this is a text file, a permission to execute is not needed.



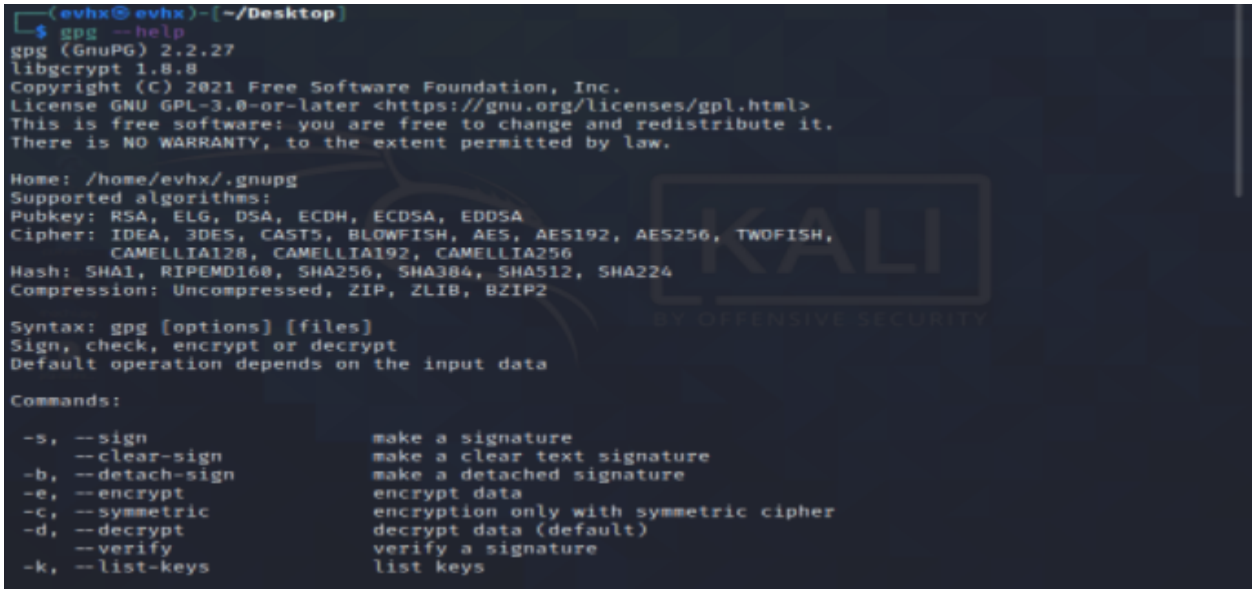
Utilization of the GNU Privacy Guard

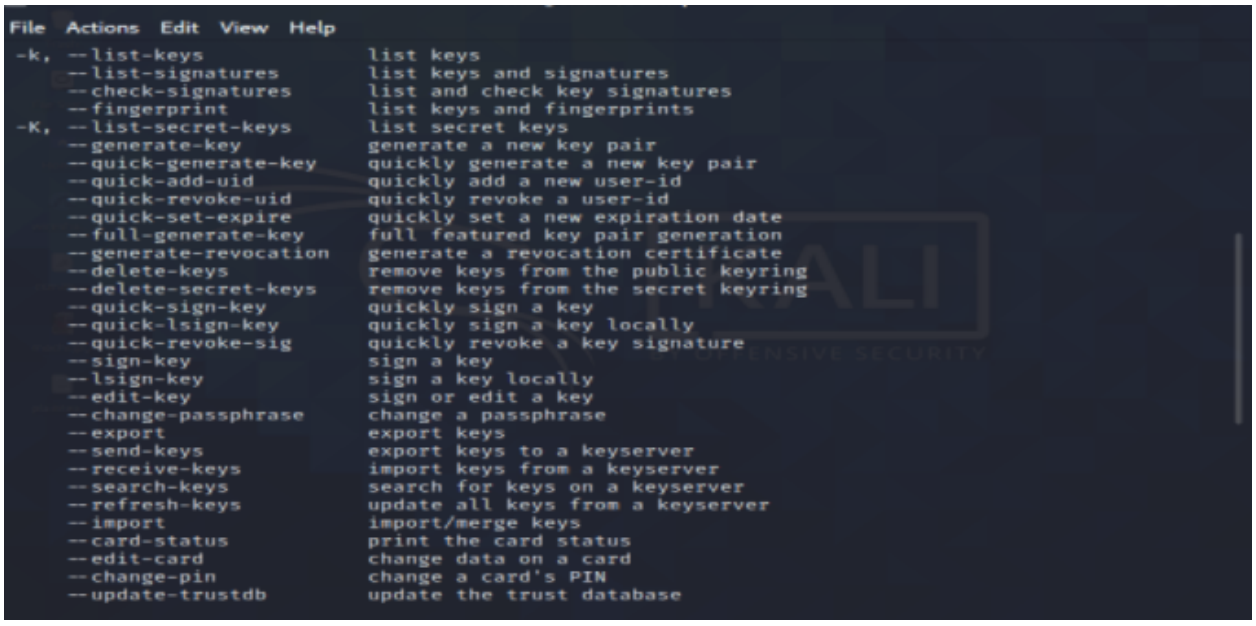
For a list of commands provided by the GPG, type the command:

gpg --help

The commands that will be used to perform symmetric and asymmetric tasks will be:

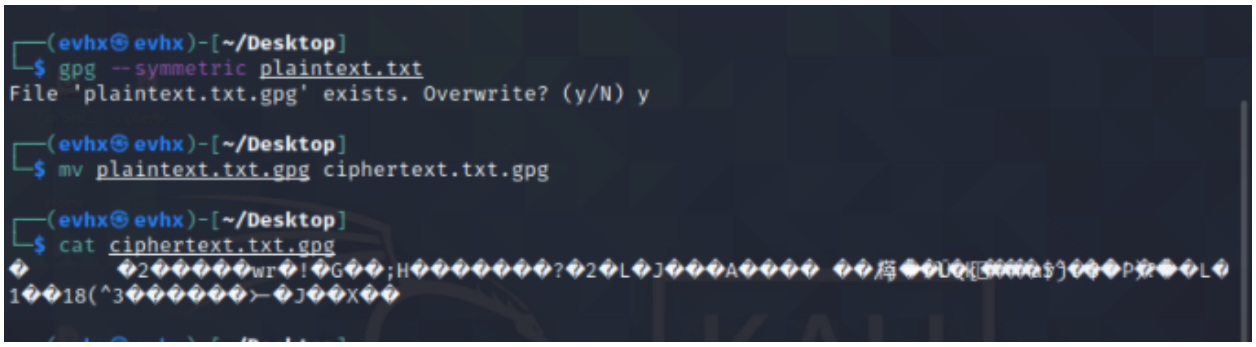
--import	(import/merge keys	)
--list-keys	(list keys	)
--list-secret-keys	(list secret keys	)
--full-generate-key	(full featured key pair generation	)
--sign	(make a signature	)
--export	(export keys	)
--symmetric	(encryption with symmetric cipher	)
-a	(create ASCII armored output	)
-e	(encrypt data	)
-u	(name used for the local user	)
-r	(encrypt for user id name	)





Symmetric encryption will be used on the plaintext.txt file to create a binary output into a ciphertext.txt file, with the file format of '.gpg', using the commands:

gpg --symmetric plaintext.txt (symmetric encryption)
mv plaintext.txt.gpg ciphertext.txt.gpg (creates new files)
cat ciphertext.txt.gpg (reads data from file and gives output)



Symmetric encryption will also be used for the ASCII armored output version of the ciphertext.txt file, with a file format of '.asc', using the commands:

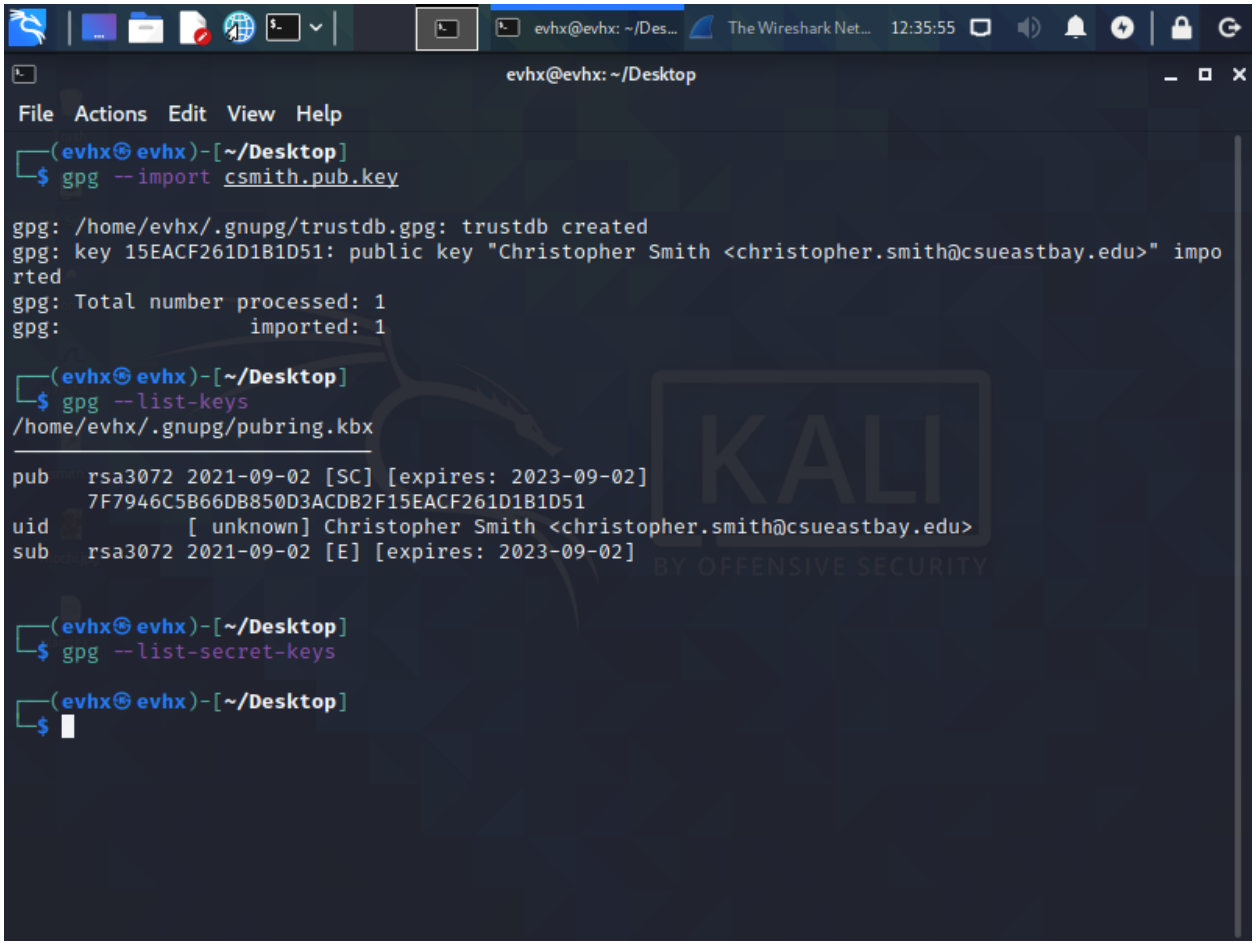
gpg --symmetric -a plaintext.txt (symmetric encryption)
mv plaintext.txt.asc ciphertext.txt.asc (creates new files)
cat ciphertext.txt.asc (reads data from file and gives output)

Import the provided public key called ‘csmith.pub.key’ into the gpg keyring using the command:
gpg --import csmith.pub.key

To check if the key was successfully imported, check the keys with the command:
gpg --list-keys

The key imported is a public key and not a private key. If private keys need to be viewed, use the command:
gpg --list-secret-keys

There are no private keys created, only the public key “Christopher Smith <christopher.smith@csueastbay.edu>” has been imported.

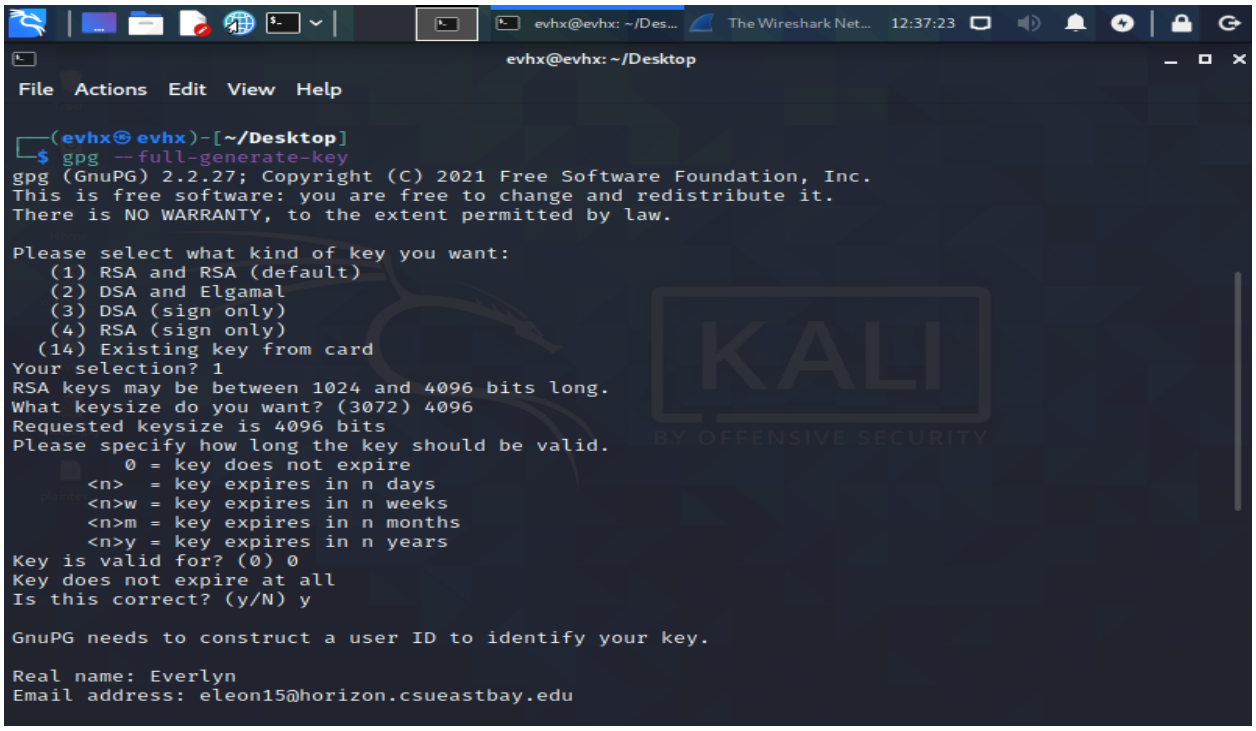


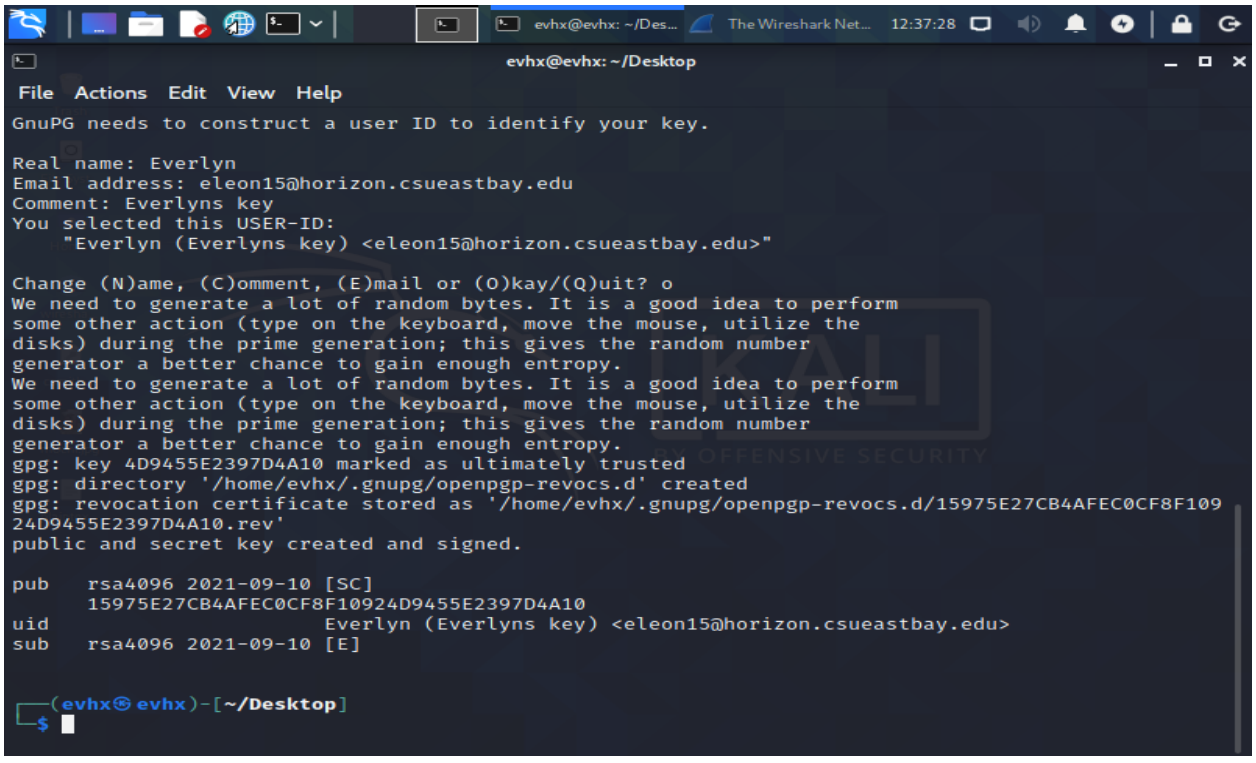
To create a new public/private key pair use the command:
gpg --full-generate-key

The prompted questions will be asked for key specifications such as:

Algorithm used	-> (1) RSA and RSA (default)	(has multi-use functionality)
Key size	-> 4096	(common key length)
Expiration date	-> 0 - key does not expire	(do not need the key to expire)

Then proceed with creating a user ID for identifying the key.

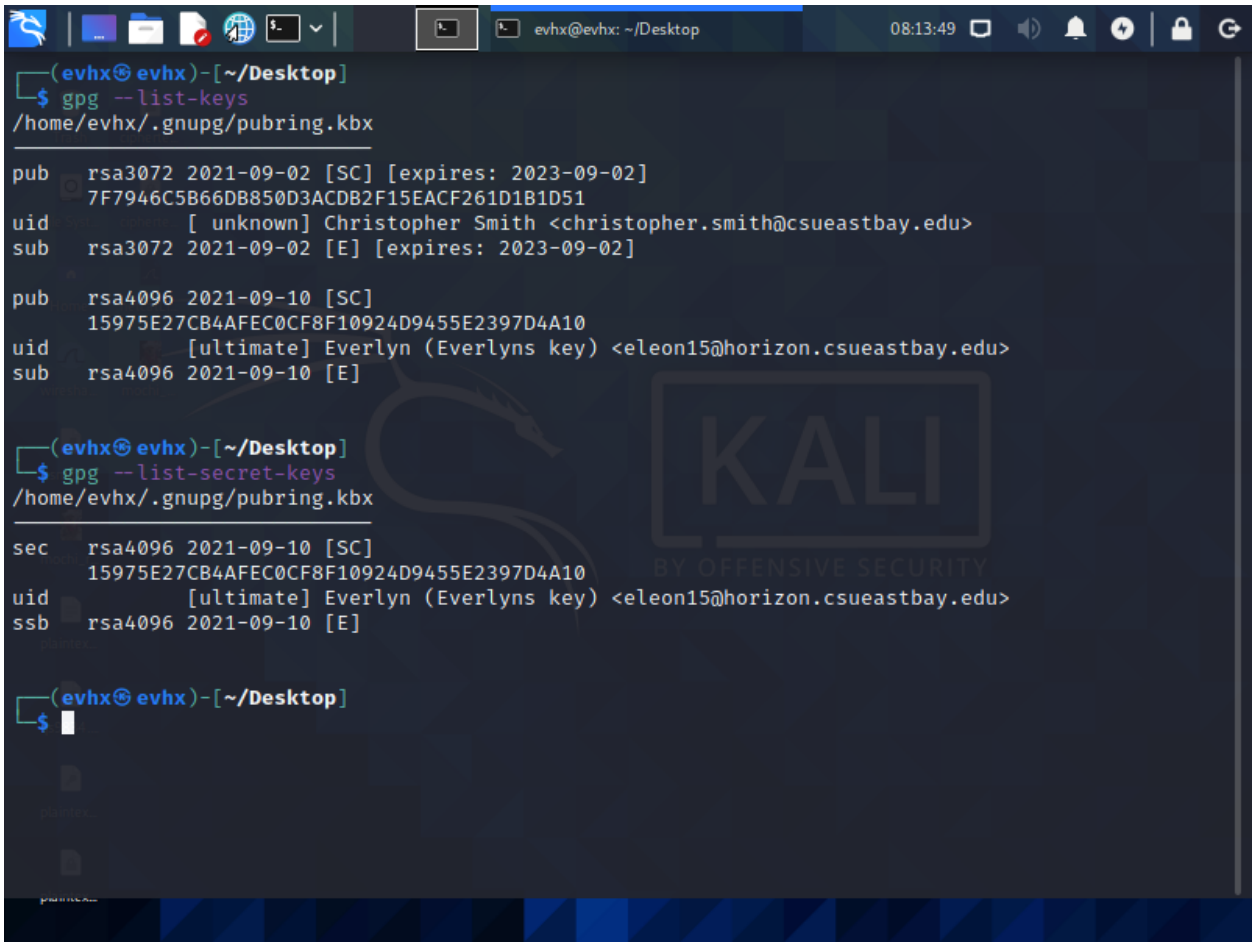




The key pair has been created. To view the keys use the command:
gpg --list-keys

The pair is a pair of a private and a public key, and to view the private key specifically, use the command:
gpg --list-secret-keys

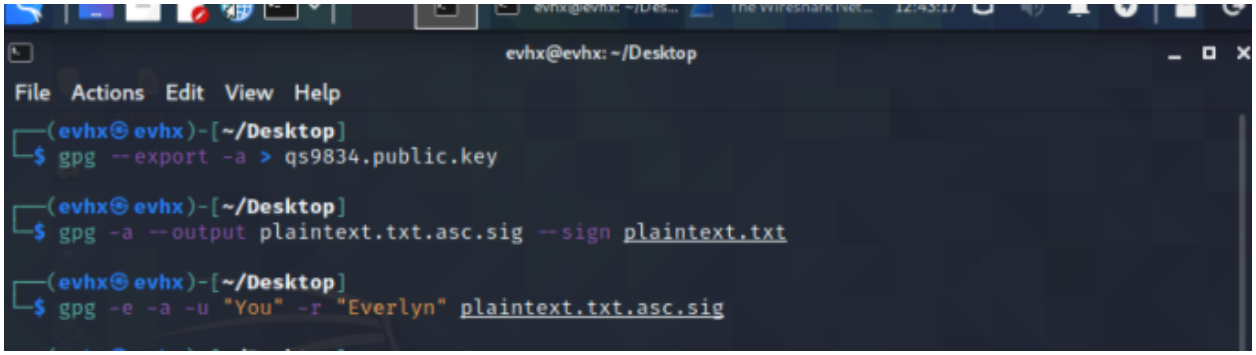
Now unlike before, there is a private key, under the ‘list secret keys’ command.



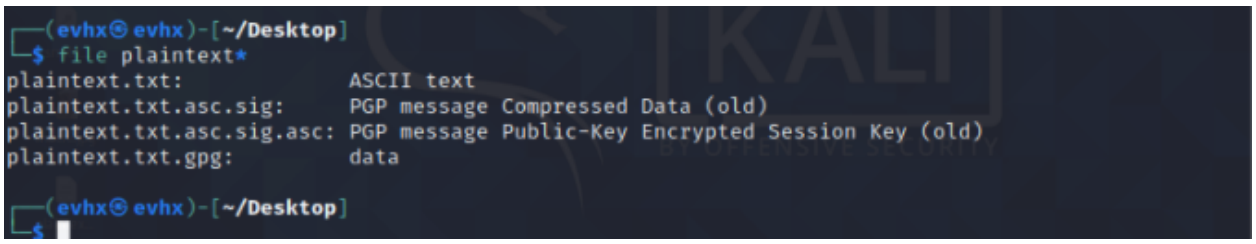
The public key will be exported into a file named [NETID].public.key, using the command:
gpg --export -a > qs9834.public.key

Then sign the plaintext.txt file with the private key using the command:
gpg -a --output plaintext.txt.asc.sig --sign plaintext.txt

Encrypt the signed file to ASCII output using the provided key, using the command:
gpg -e -a -u “[USERNAME]” -r “[NAME]” plaintext.txt.asc.sig

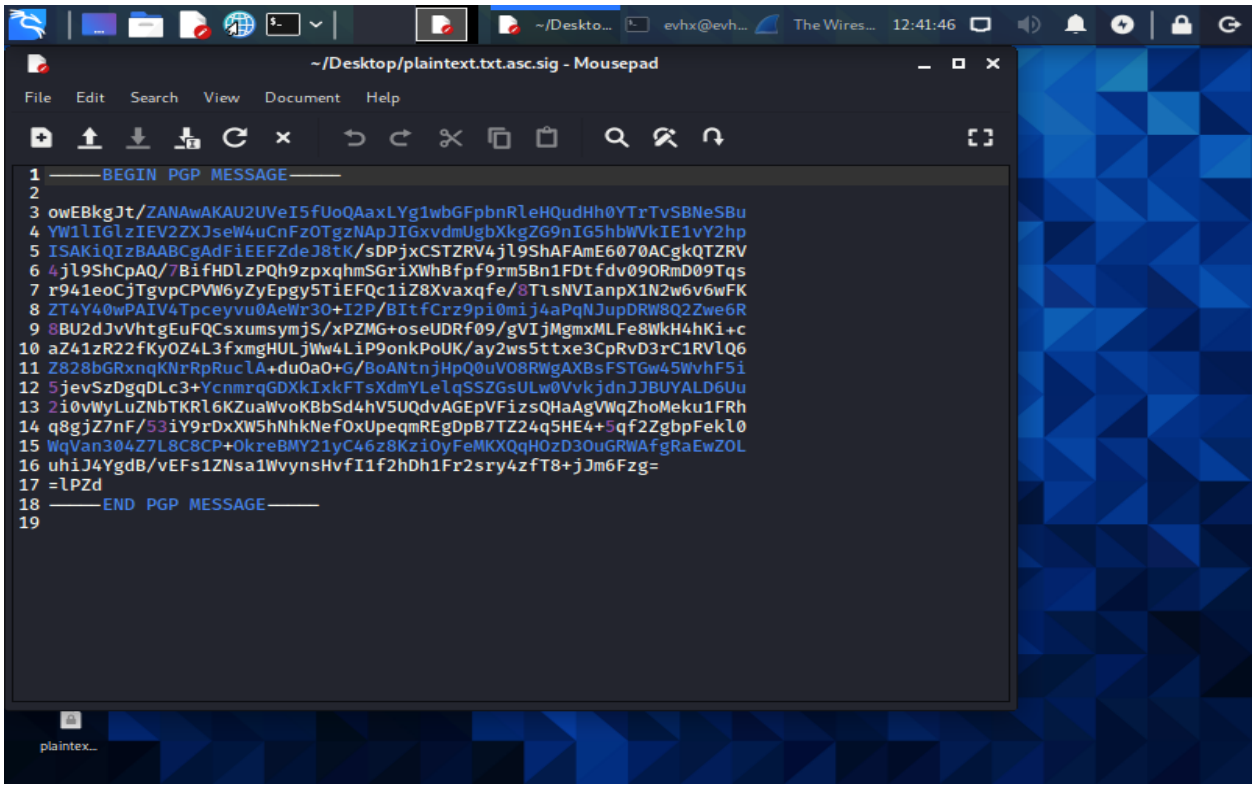


To display all the file types of ‘plaintext’ use the ‘ * ’ option, such as the command:
file plaintext*

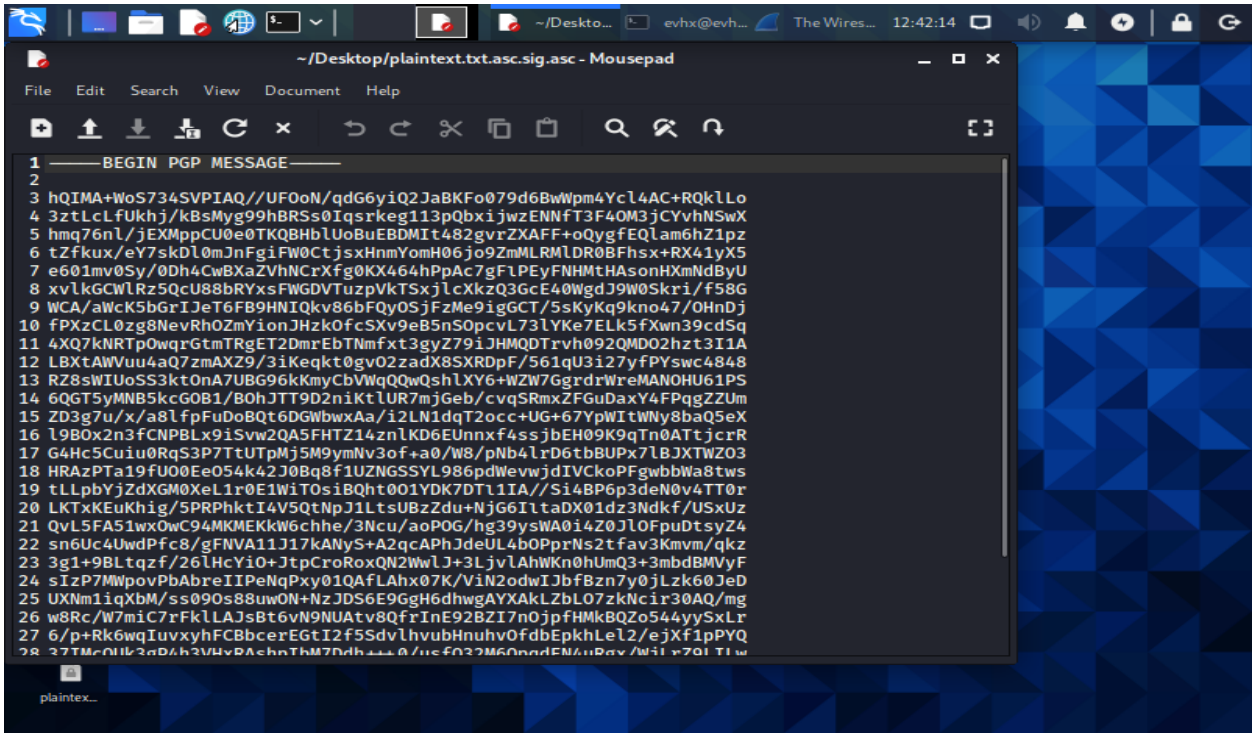


The differences between the ‘plaintext’ file signed with a public or private key can be viewed within the file.

Public Key:



Private Key:

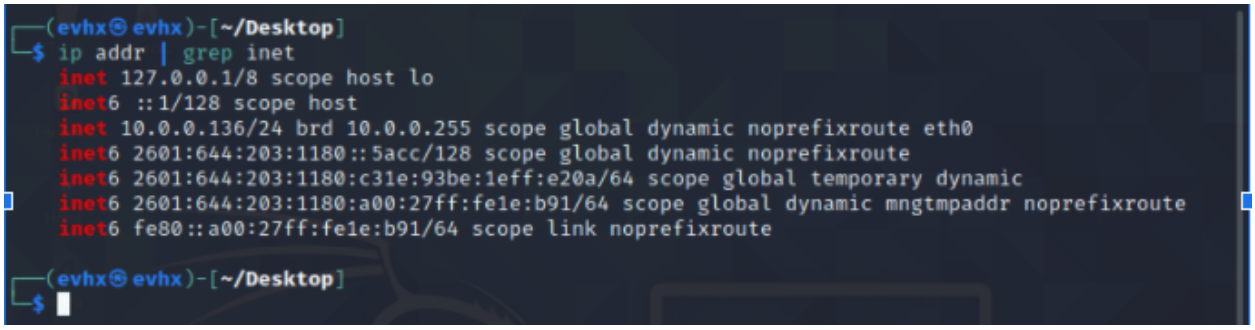


Utilization of Netcat

Netcat will be used to transmit an ASCII formatted encrypted text from an Ubuntu Machine to a Kali Linux machine. Wireshark will be used to analyze the network packets being transmitted between these two machines.

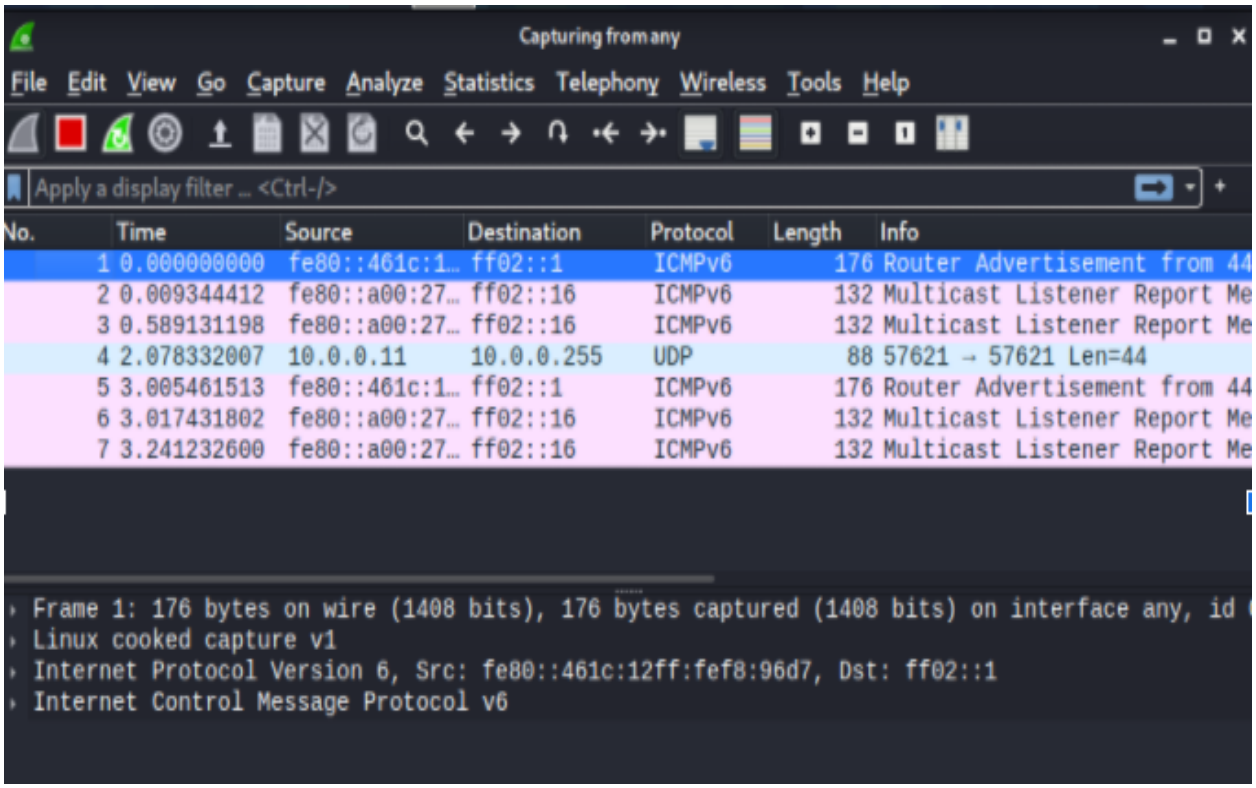
In the Ubuntu and Kali Linux machines, retrieve the IP address using the command:
ip addr | grep inet.

Kali Linux IP: 10.0.0.136



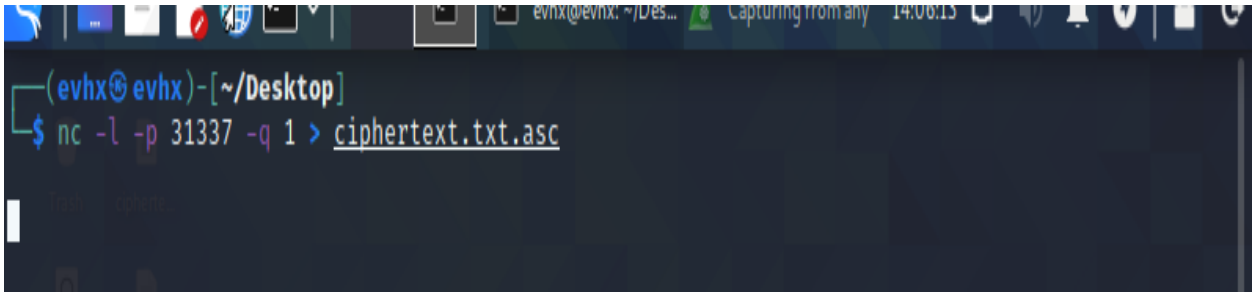
Ubuntu IP: 10.0.0.251

Wireshark will be used because it provides a variety of tools to make network packets easy to analyze, such as a display filter, the list of captured packets, the details to any selected packet, and the ASCII and hexadecimal contained within the packets. To start reading the packages, press the blue shark fin button, and to stop the capturing, press the red square.



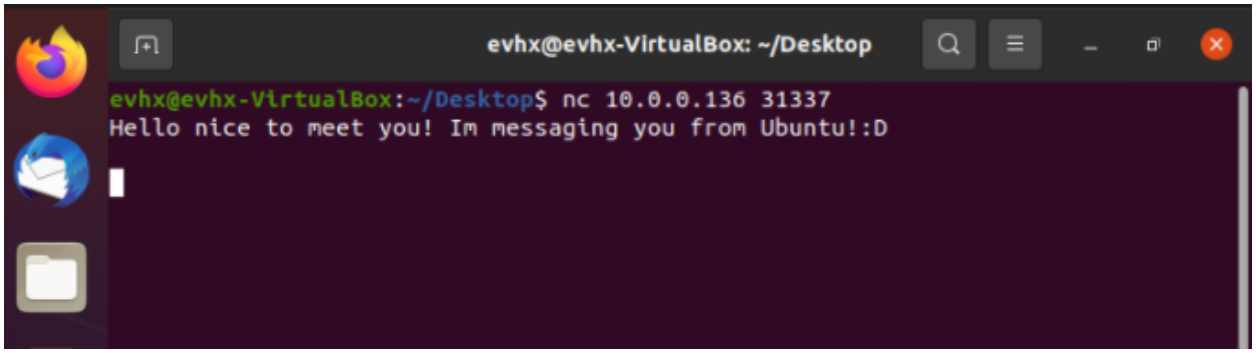
Set up a prearranged listener on the Kali Linux machine, that will write whatever it listens to into the ciphertext.txt.asc file. Use the command:
nc -l -p [socket number] -q 1 > ciphertext.txt.asc

The ‘nc’ command represents ‘Netcat’, and is able to read and write network connections in relation to TCP, UDP and UNIX sockets.

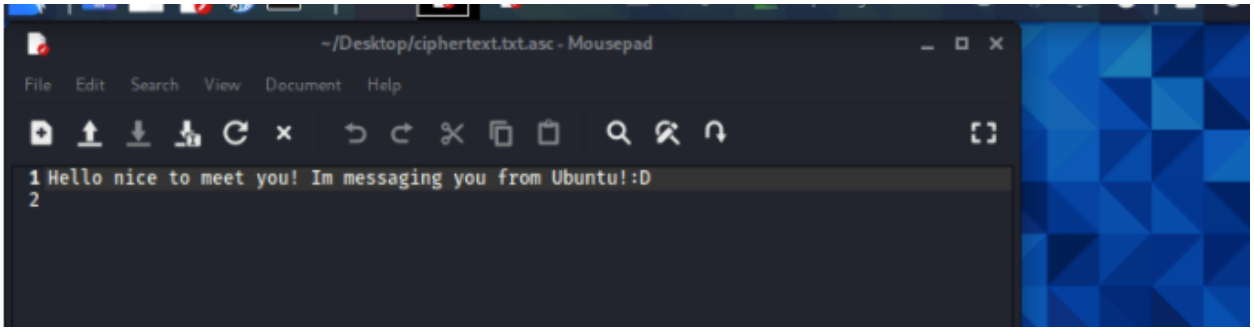


The Ubuntu machine will communicate to the Kali Linux machine listener, using the machine's IP address and the port number used for the listener. Use the command:
nc [IP address] [port number]

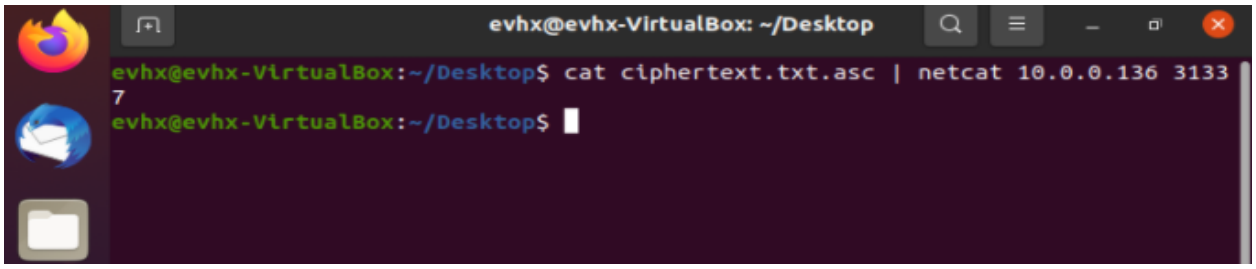
Then type a message that will be sent to the Kali Linux machine, which then is concatenated into the ciphertext.txt.asc file.



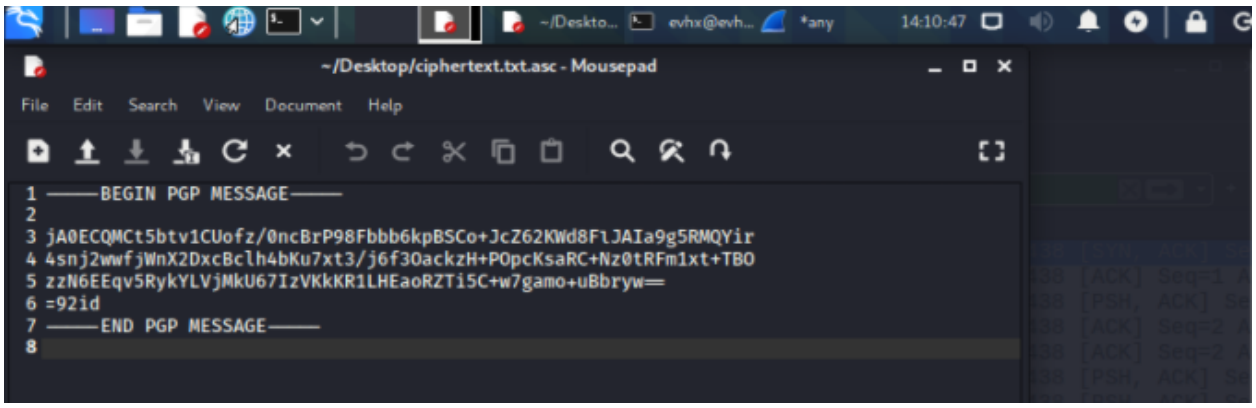
In the Kali Linux machine, open the ciphertext.txt.asc file to view the message sent from the Ubuntu machine.



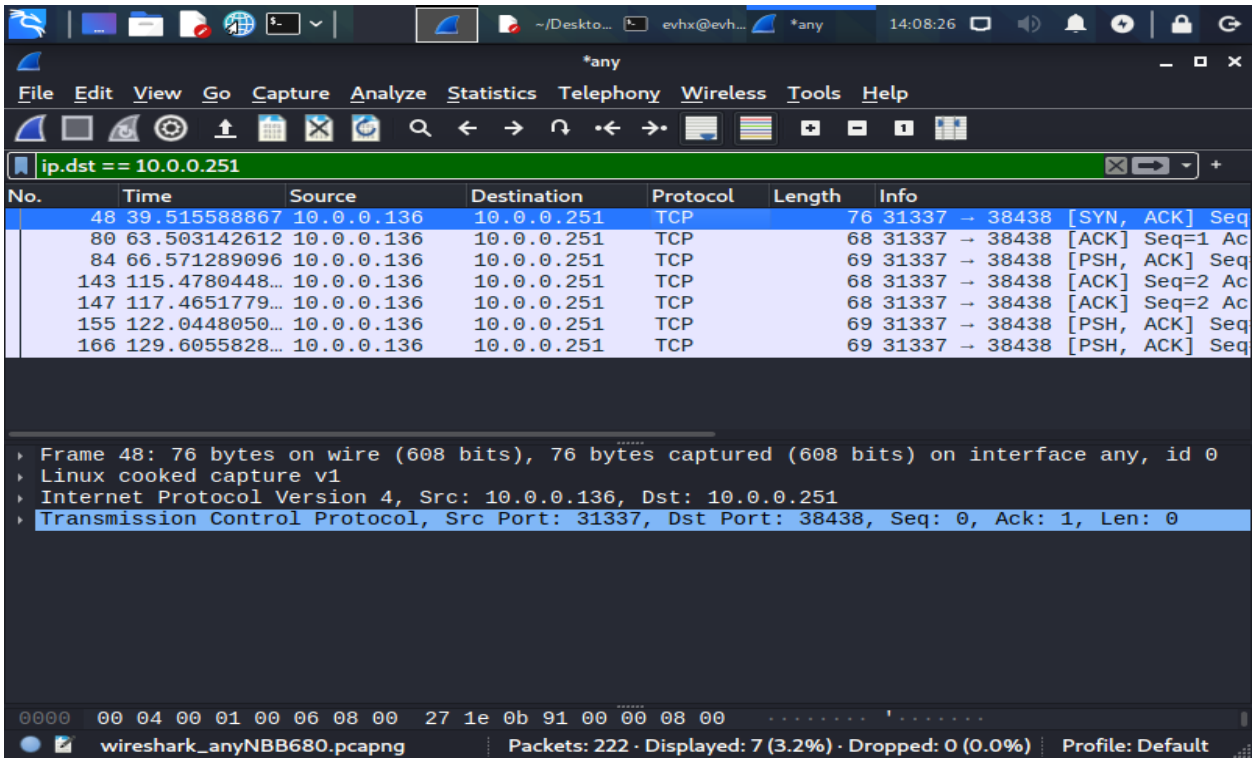
Now send the ASCII armored encrypted file to the prearranged listener using the command:
cat ciphertext.txt.asc | netcat 10.0.0.136 31337



On the Kali Linux machine, view the ciphertext.txt.asc, which now contains the ASCII armored encrypted file.

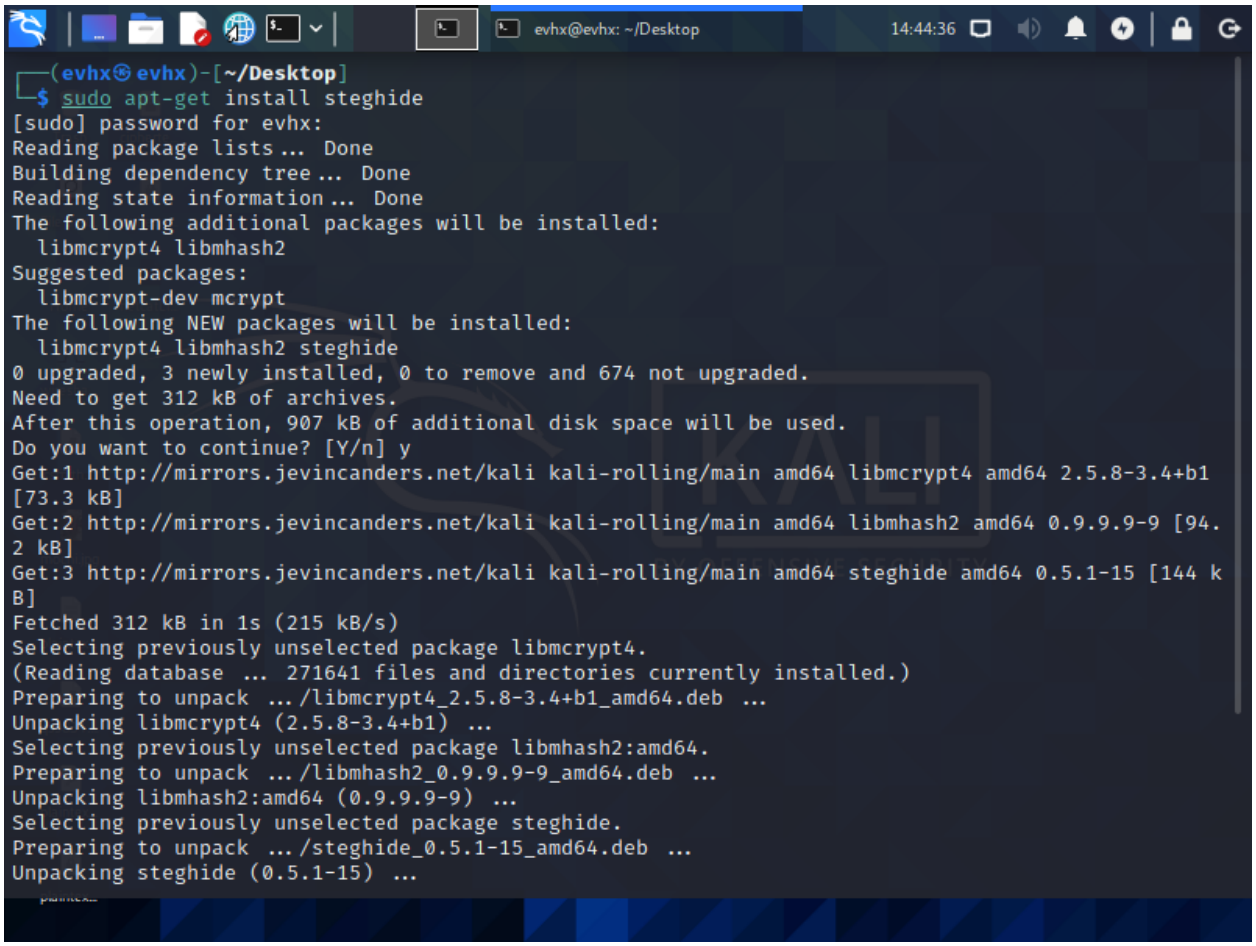


Stop the packet capturing on Wireshark and filter the packets to the ones being communicated between the Ubuntu and Kali Linux machine by using the filter command:
ip.dst==10.0.0.251 OR ip.addr==10.0.0.251
These packets contain data from the messages communicated between machines.



Utilization of Steghide

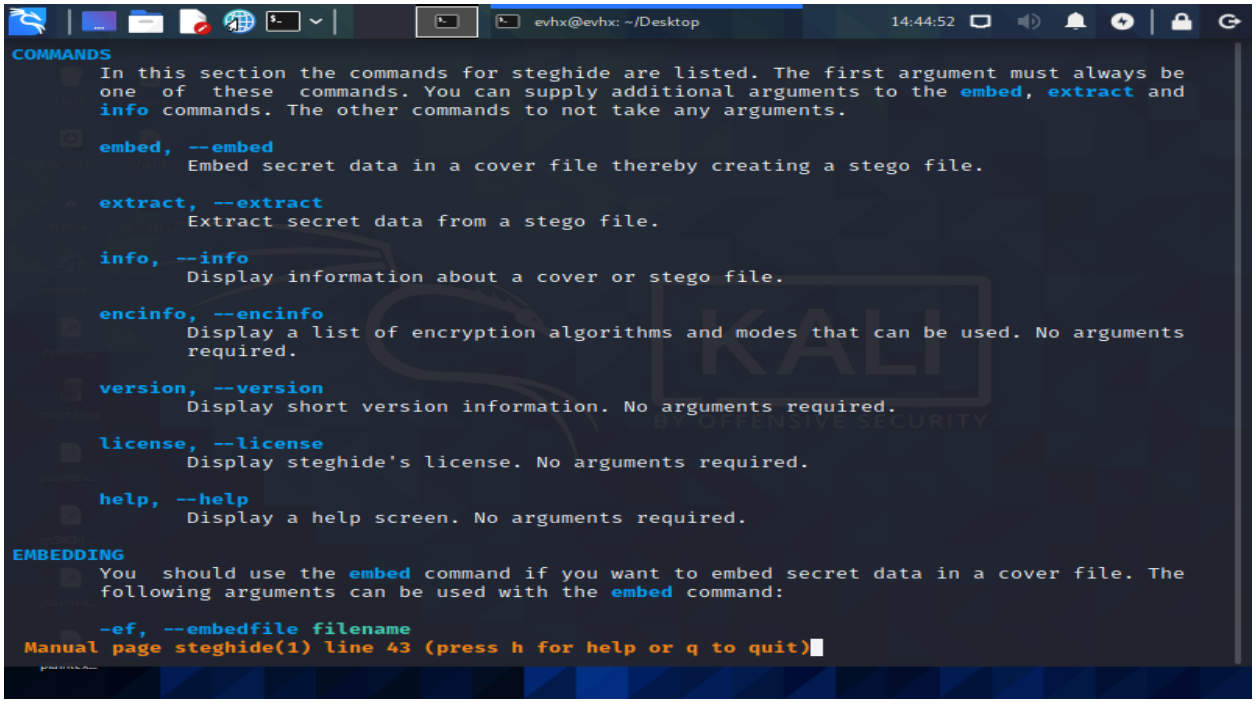
A Steganography tool named Steghide will be used to embed a text file into a jpeg image file. First Steghide needs to be installed onto the machine using the command:
sudo apt-get install steghide

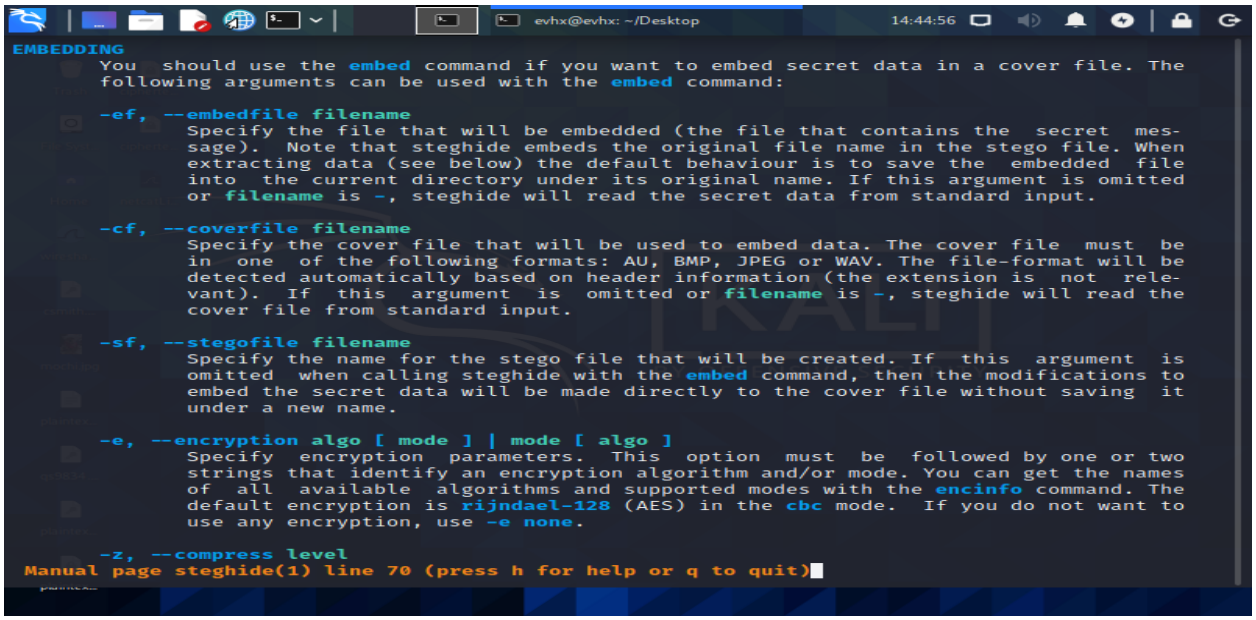


The Steghide manual page can be used with the command:
man steghide

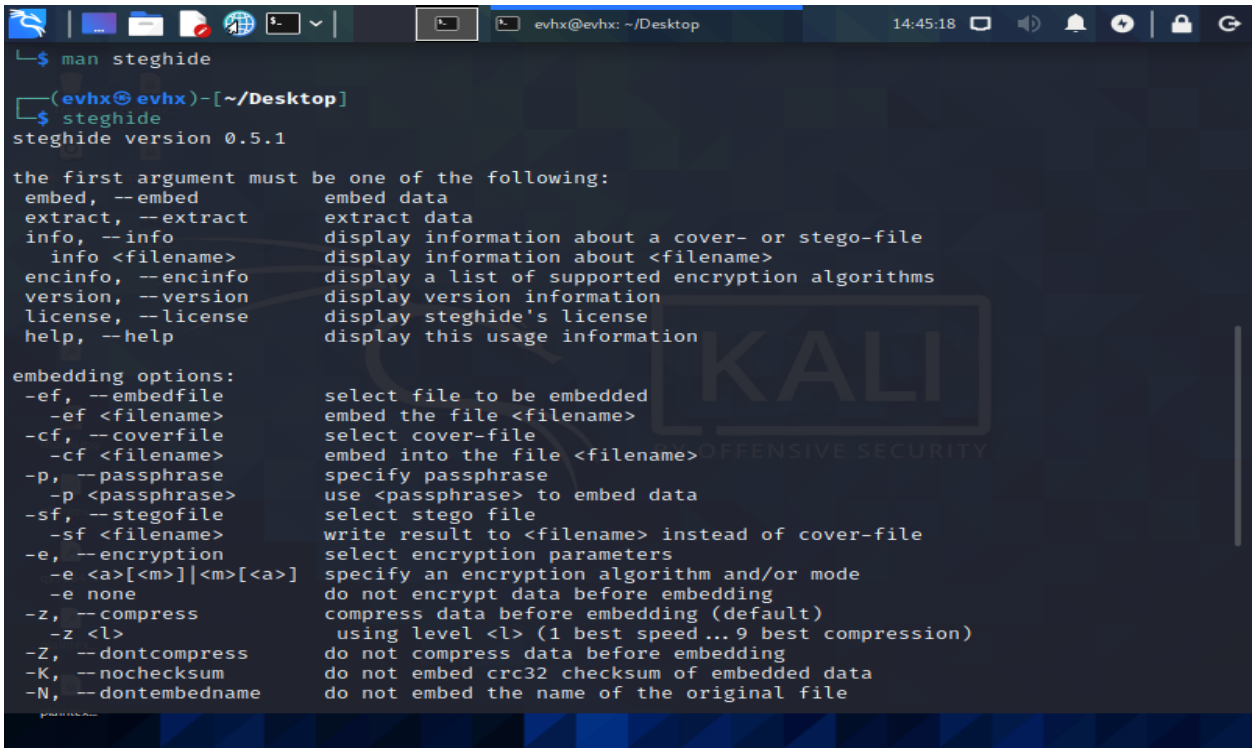
The Steghide commands that will be use will be:

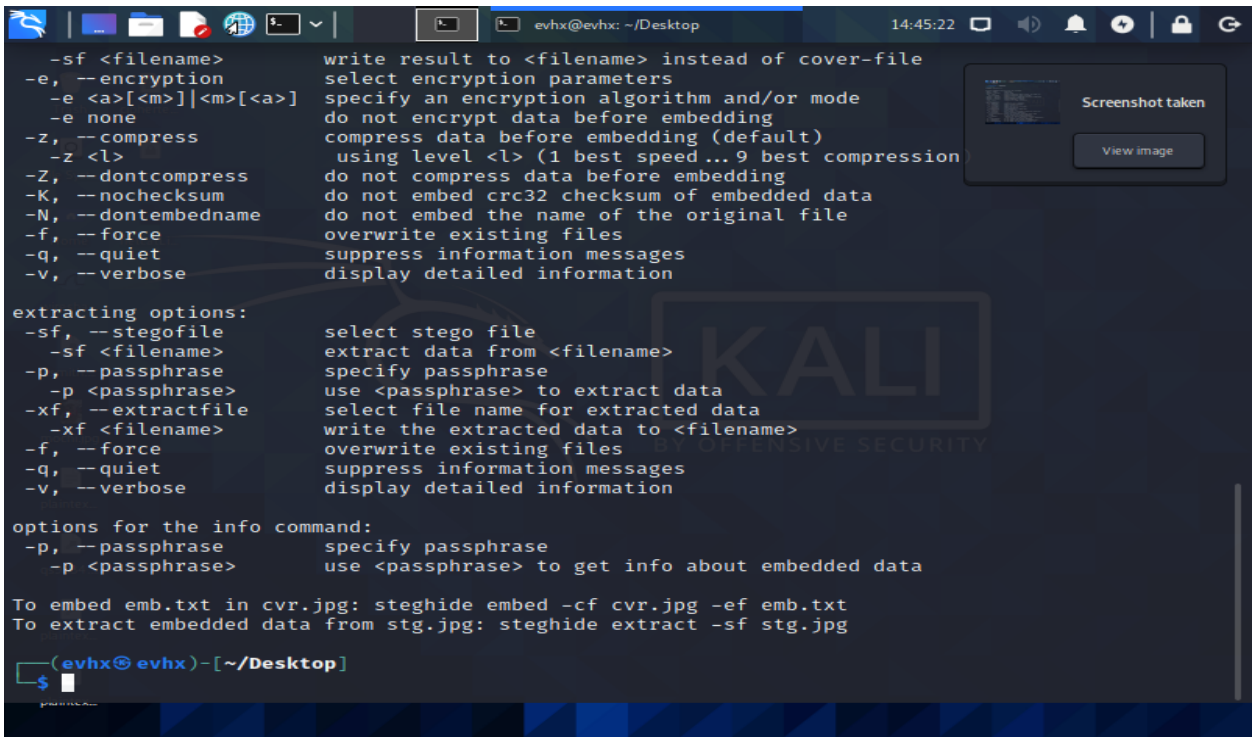
embed	(embed data into the cover(jpeg) file, creating a stego file)
-ef	(specify file that will be embedded into the cover file)
-cf	(specify the cover file that will be used to embed data)
-sf	(specify the name for the stego file (embedded file))
-p	(passphrase used to embed data)
-xf	(extract data from the stego file into another filename)
extract	(extract secret data from a stego file)



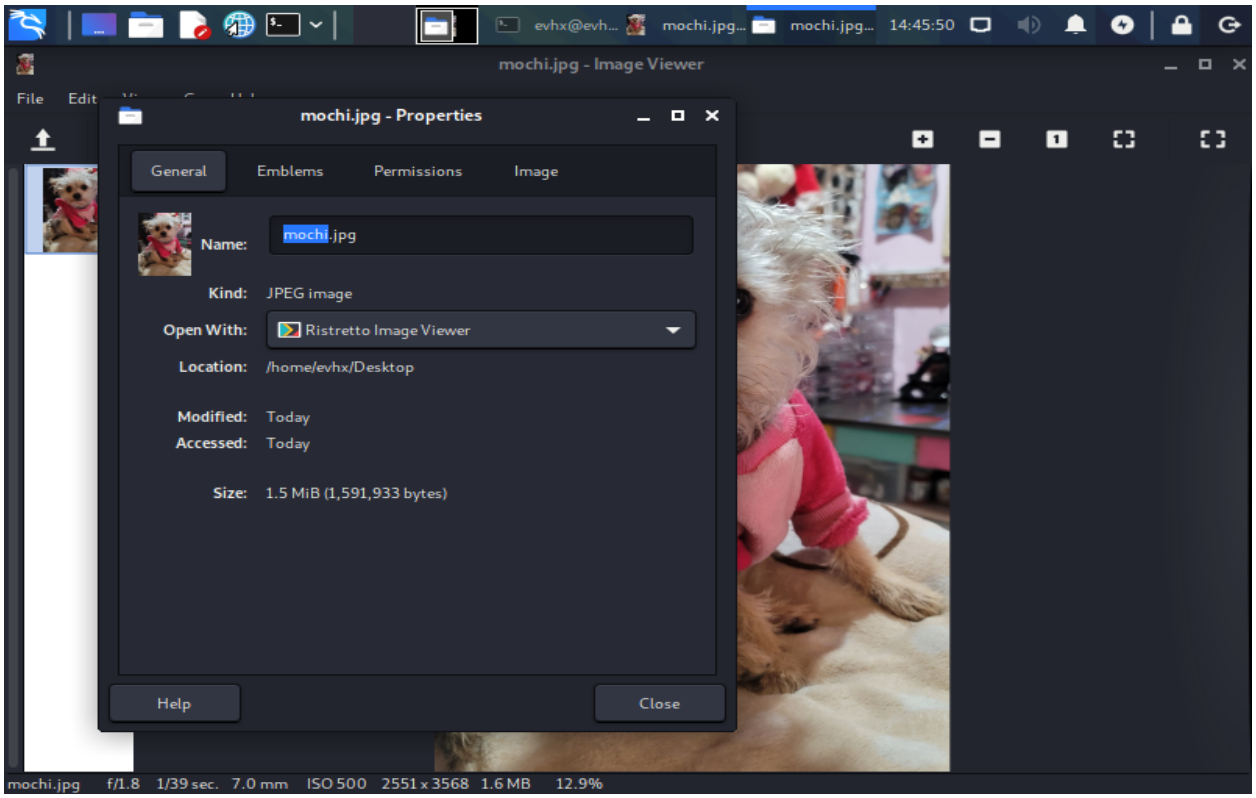


Another way to view the Steghide commands is to simply type the command:
steghide





A jpeg image named ‘mochi.jpg’ will be used to demonstrate the Steghide commands. This mochi.jpg is 1,591,933 bytes, which will change after the image has been embedded with the plaintext.txt file.



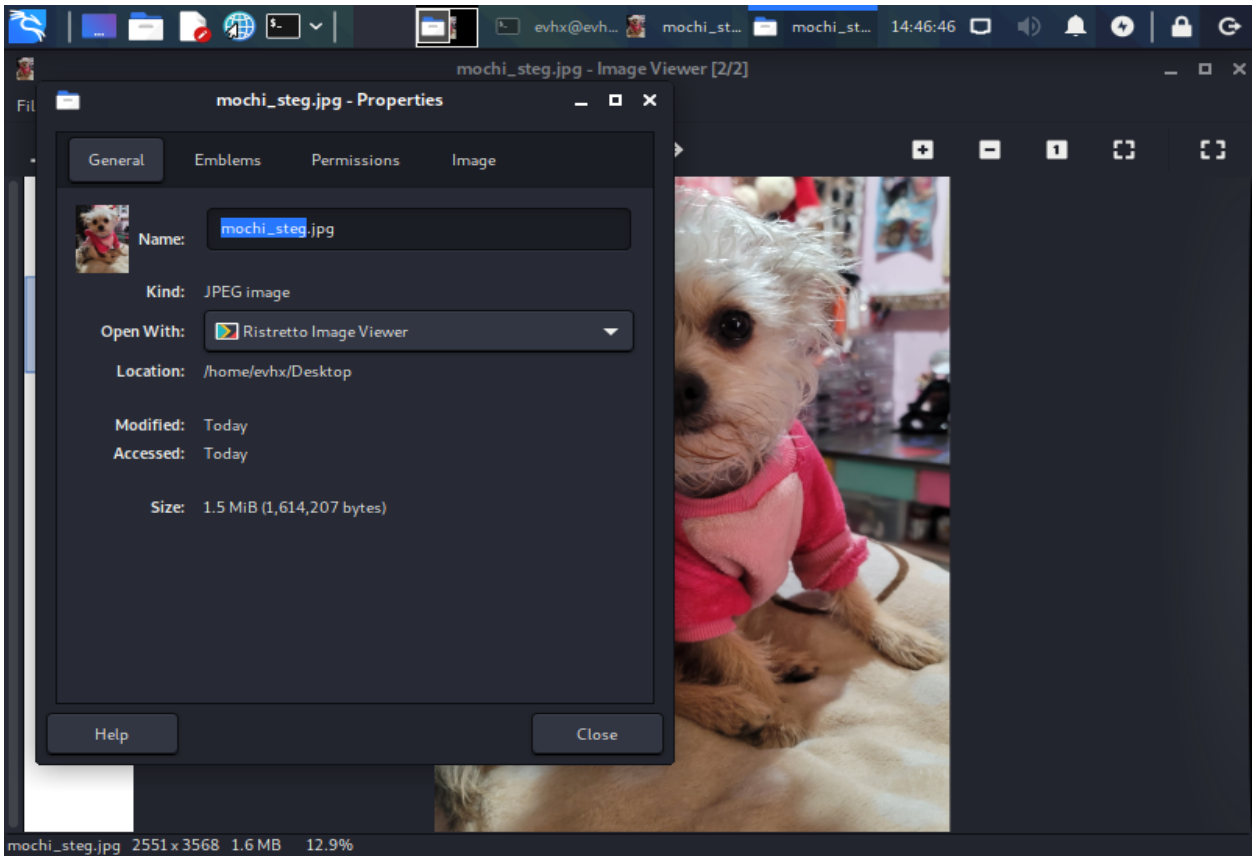
To embed the plaintext.txt file into the mochi.jpg file, to create a new embedded image file named mochi_steg.jpg, use the command:
steghide embed -ef plaintext.txt -cf mochi.jpg -sf mochi_steg.jpg -p letmein

This new file called ‘mochi_stego.jpg’ will be embedded with the plaintext.txt file, with a passphrase that is ‘letmein’.

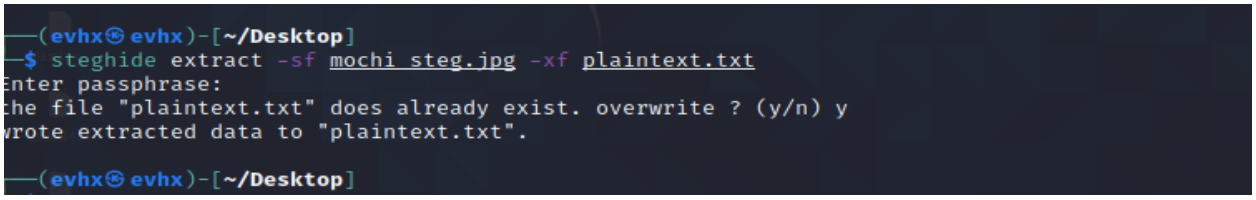
```
┌──(evhx㉿evhx)-[~/Desktop]
└─$ steghide embed -ef plaintext.txt -cf mochi.jpg -sf mochi_steg.jpg -p letmein
embedding "plaintext.txt" in "mochi.jpg"... done
writing stego file "mochi_steg.jpg"... done
```

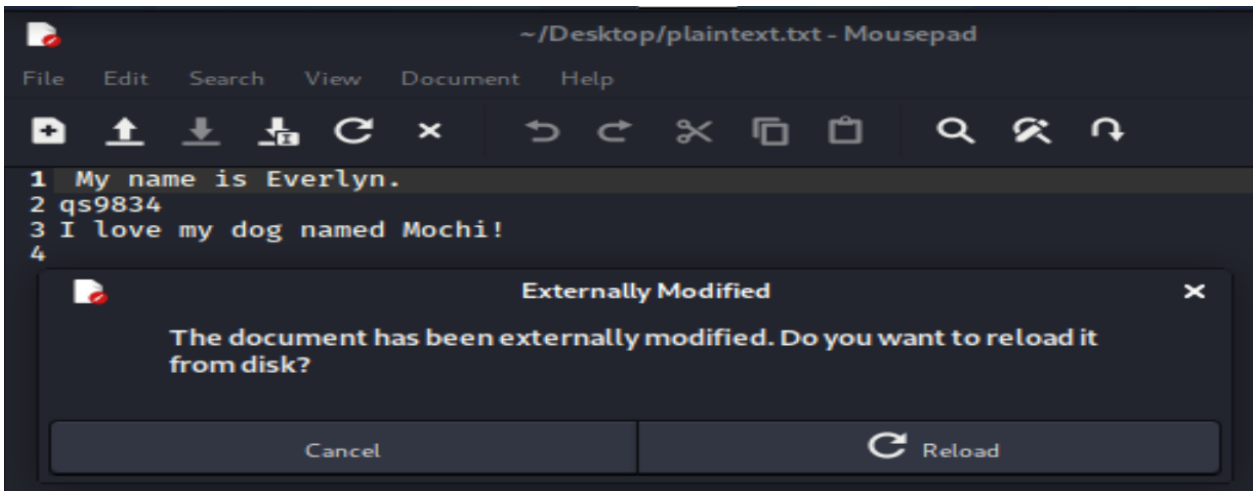
The mochi_stego.jpg file still looks exactly the same as the original mochi.jpg file, but the number of bytes was increased to 1,614,207 bytes.

mochi.jpg - 1,591,933 bytes
mochi_stego.jpg - 1,614,207 bytes



To extract from the embedded image and modify the plaintext.txt file use the command:
steghide extract -sf mochi_steg.jpg -xf plaintext.txt



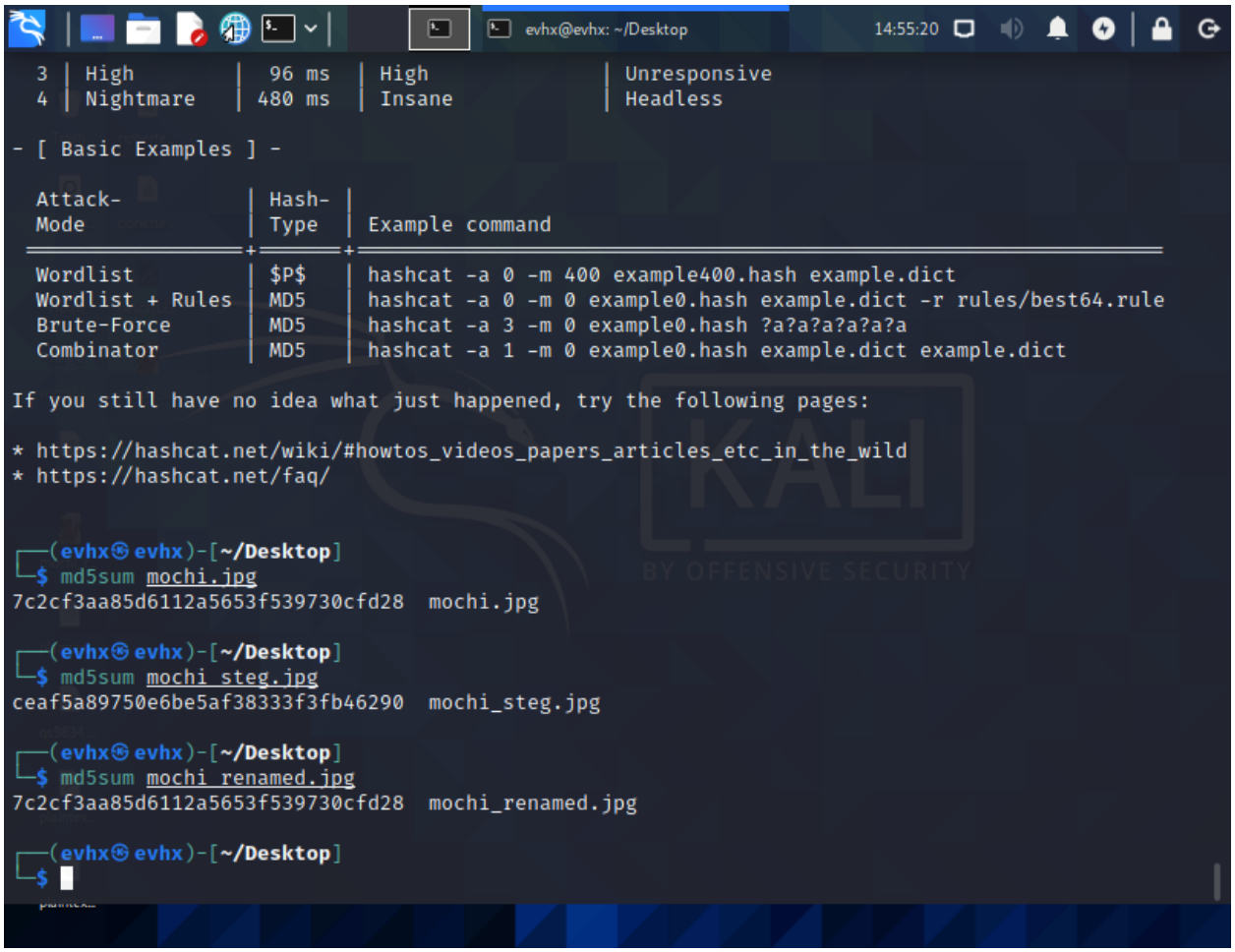


Utilization of the Md5 Checksums

An Md5 Checksum is an algorithm that can check the digital integrity and security of a file to ensure that it is the accurate copy of the original file. To check the integrity of the jpeg images used, type the command:
md5sum [jpeg image]

The integrity for the original image, embedded image file and re-named image file will be checked:
md5sum mochi.jpg
(results: 7c2cf3aa85d6112a5653f53970cfd28)
md5sum mochi_steg.jpg
(results: ceaf5a89750e6be5af38333f3fb46290)
md5sum mochi_renamed.jpg
(results: 7c2cf3aa85d6112a5653f53970cfd28)

The original mochi.jpg file and renamed mochi_renamed.jpg file have the same checksum, meaning the checksum of a file will not change if the file is renamed. The mochi_steg.jpg file on the other hand does have a different checksum, since it was embedded with a file.



Conclusion

GPG, Netcat, Steghide and Md5sum were used to show the various ways components in a Cryptosystem could be utilized in symmetric and asymmetric encryption. GPG is able to perform a variety of symmetric and asymmetric encryption tasks with the use of keys, but these same tasks can easily be replicated with symmetric algorithms. As shown in the Netcat activity, communication using Netcat can easily be seen with a package reader like Wireshark. Concealing information using Steganography, such as embedding data into images or videos is advantages when used with groups of people who wouldn't even understand the idea of embedding information in an image without seeing the image change, otherwise there are too many tools that can check the integrity of these images, such as the checksums Md5 uses.