

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.

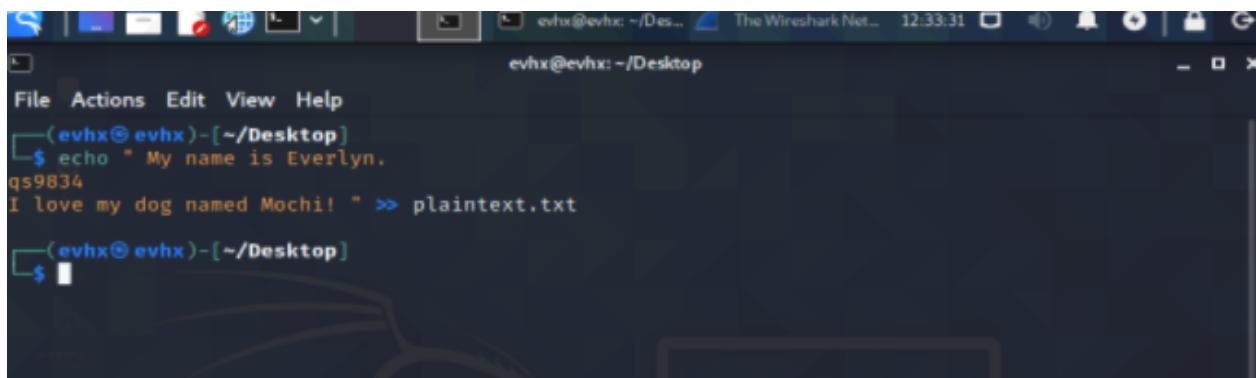
Introduction

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
```



```
File Actions Edit View Help
└─( evhx@ evhx )-[ ~/Desktop ]
$ echo " My name is Everlyn.
qs9834
I love my dog named Mochi! " >> plaintext.txt
└─( evhx@ evhx )-[ ~/Desktop ]
$
```

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.

```
evhx@evhx: ~/Desktop
File Actions Edit View Help
└──(evhx@evhx)-[~/Desktop]
    $ echo " My name is Everlyn.
    qs9834
    I love my dog named Mochi! " >> plaintext.txt

└──(evhx@evhx)-[~/Desktop]
    $ chmod 600 plaintext.txt
└──(evhx@evhx)-[~/Desktop]
    $
```

A screenshot of a terminal window on Kali Linux. The terminal window has a dark theme with light-colored text. It shows the user's session path as "evhx@evhx: ~/Desktop". The user has run the command "echo" to create a file named "plaintext.txt" containing the text " My name is Everlyn. I love my dog named Mochi!". After creating the file, the user runs the command "chmod 600 plaintext.txt" to change the file's permissions. The terminal window also displays the Kali logo in the background.

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)

```
(evhx㉿evhx) [~/Desktop]
$ gpg --help
gpg (GnuPG) 2.2.27
libgcrypt 1.8.8
Copyright (C) 2021 Free Software Foundation, Inc.
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/evhx/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2

Syntax: gpg [options] [files]
Sign, check, encrypt or decrypt
Default operation depends on the input data

Commands:

 -s, --sign           make a signature
 -c, --clear-sign    make a clear text signature
 -b, --detach-sign   make a detached signature
 -e, --encrypt        encrypt data
 -c, --symmetric      encryption only with symmetric cipher
 -d, --decrypt        decrypt data (default)
 -v, --verify          verify a signature
 -k, --list-keys      list keys
```

```
File Actions Edit View Help
-k, --list-keys      list keys
--list-signatures    list keys and signatures
--check-signatures   list and check key signatures
--fingerprint       list keys and fingerprints
-K, --list-secret-keys list secret keys
--generate-key       generate a new key pair
--quick-generate-key quickly generate a new key pair
--quick-add-uid      quickly add a new user-id
--quick-revoke-uid   quickly revoke a user-id
--quick-set-expire   quickly set a new expiration date
--full-generate-key  full featured key pair generation
--generate-revocation generate a revocation certificate
--delete-keys         remove keys from the public keyring
--delete-secret-keys remove keys from the secret keyring
--quick-sign-key     quickly sign a key
--quick-lsign-key    quickly sign a key locally
--quick-revoke-sig   quickly revoke a key signature
--sign-key            sign a key
--lsign-key           sign a key locally
--edit-key            sign or edit a key
--change-passphrase  change a passphrase
--export              export keys
--send-keys           export keys to a keyserver
--receive-keys        import keys from a keyserver
--search-keys         search for keys on a keyserver
--refresh-keys        update all keys from a keyserver
--import              import/merge keys
--card-status         print the card status
--edit-card           change data on a card
--change-pin          change a card's PIN
--update-trustdb     update the trust database
```

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)
```

```
(evhx@evhx) [~/Desktop]
$ gpg --symmetric plaintext.txt
File 'plaintext.txt.gpg' exists. Overwrite? (y/N) y

(evhx@evhx) [~/Desktop]
$ mv plaintext.txt.gpg ciphertext.txt.gpg

(evhx@evhx) [~/Desktop]
$ cat ciphertext.txt.gpg
-----BEGIN PGP MESSAGE-----
```

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)
```

```
(evhx@evhx) [~/Desktop]
$ gpg --symmetric -a plaintext.txt
(evhx@evhx) [~/Desktop]
$ mv plaintext.txt.asc ciphertext.txt.asc
(evhx@evhx) [~/Desktop]
$ cat ciphertext.txt.asc
-----BEGIN PGP MESSAGE-----
```

jA0ECQMCcfhjcjUG7Vj/0ncBMJsbZzBXDPGiXN6xLAlulAou6oE5hA4Gh6yt6M2g
lxED2u07pXgUzeqNY/xhFyldIC+Yzg+H+LyCnKX1rZKNVPb01Kun7KqLz5BCuLdL
3zDknnRFhAxe28uf74MiGbCzihzY/cuwTWtGcnseS2lvmMoFt3TRcg==
=TgNC
-----END PGP MESSAGE-----

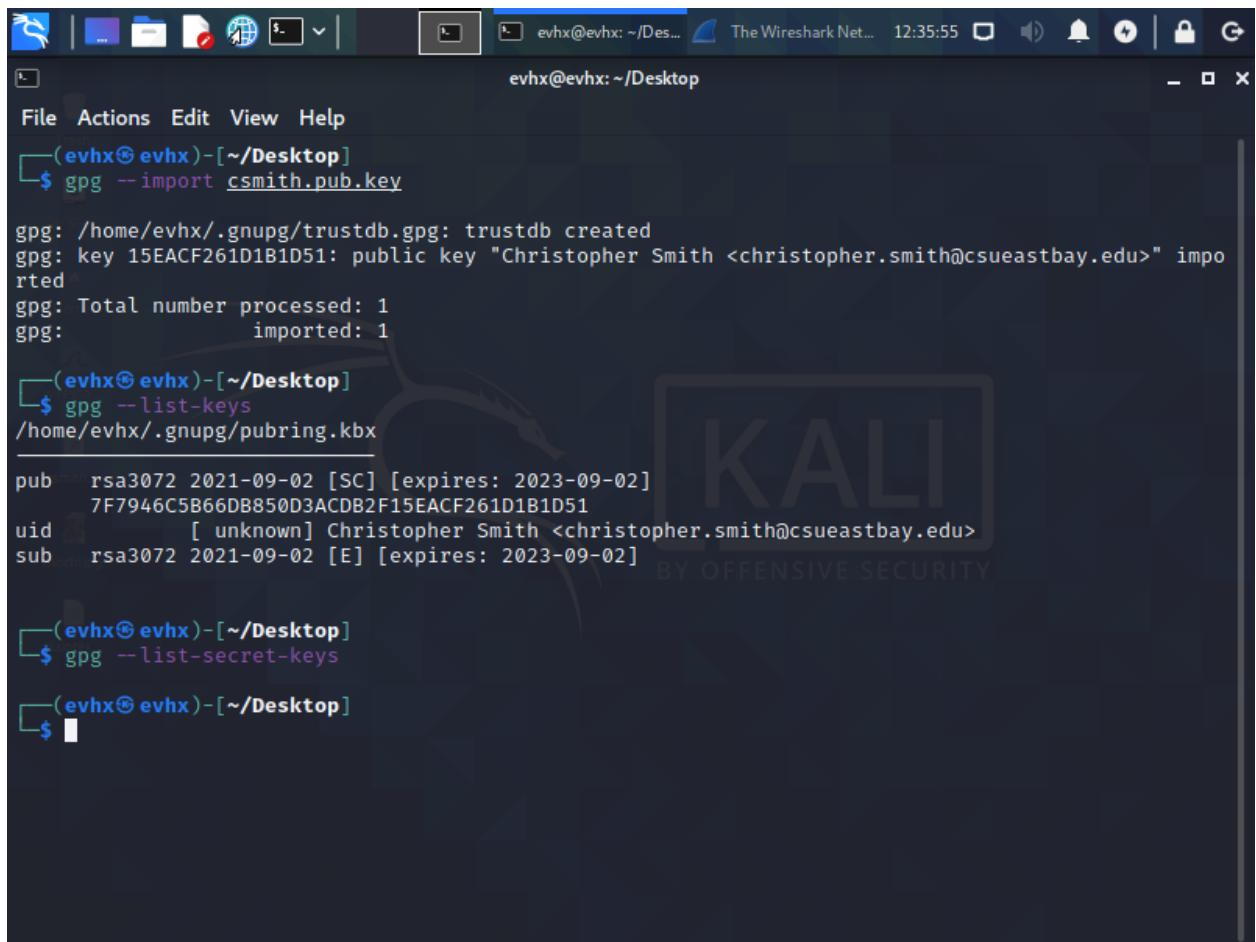
Import the provided public key called ‘csmith.pub.key’ into the gpg keyring using the command:
gpg --import csmith.pub.key

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.



The screenshot shows a terminal window on a Kali Linux desktop environment. The terminal session starts with the command `gpg --import csmith.pub.key`, which imports a public key from the file `csmith.pub.key`. The output shows the key was successfully added to the trustdb. Then, the command `gpg --list-keys` is run to list all keys in the keyring, showing the imported public key for Christopher Smith. Finally, `gpg --list-secret-keys` is run, which returns no results because there are no private keys present.

```
evhx@evhx: ~/Desktop
$ gpg --import csmith.pub.key
gpg: /home/evhx/.gnupg/trustdb.gpg: trustdb created
gpg: key 15EACF261D1B1D51: public key "Christopher Smith <christopher.smith@csueastbay.edu>" imported
gpg: Total number processed: 1
gpg:                 imported: 1

evhx@evhx: ~/Desktop
$ gpg --list-keys
/home/evhx/.gnupg/pubring.kbx
pub    rsa3072 2021-09-02 [SC] [expires: 2023-09-02]
      7F7946C5B66DB850D3ACDB2F15EACF261D1B1D51
uid          [ unknown] Christopher Smith <christopher.smith@csueastbay.edu>
sub    rsa3072 2021-09-02 [E] [expires: 2023-09-02]

evhx@evhx: ~/Desktop
$ gpg --list-secret-keys
evhx@evhx: ~/Desktop
$
```

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.

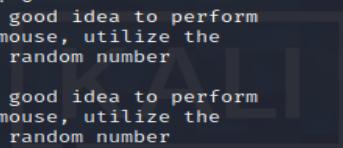


```
(evhx@evhx)-[~/Desktop]
$ gpg --full-generate-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
 (1) RSA and RSA (default)
 (2) DSA and Elgamal
 (3) DSA (sign only)
 (4) RSA (sign only)
 (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Everlyn
Email address: eleon15@horizon.csueastbay.edu
```



```
GnuPG needs to construct a user ID to identify your key.

Real name: Everlyn
Email address: eleon15@horizon.csueastbay.edu
Comment: Everlyns key
You selected this USER-ID:
  "Everlyn (Everlyns key) <eleon15@horizon.csueastbay.edu>"

Change (N)ame, (C)omment, (E)mail or (O)key/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key 4D9455E2397D4A10 marked as ultimately trusted
gpg: directory '/home/evhx/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/evhx/.gnupg/openpgp-revocs.d/15975E27CB4AFEC0CF8F109
24D9455E2397D4A10.rev'
public and secret key created and signed.

pub    rsa4096 2021-09-10 [SC]
      15975E27CB4AFEC0CF8F10924D9455E2397D4A10
uid            Everlyn (Everlyns key) <eleon15@horizon.csueastbay.edu>
sub    rsa4096 2021-09-10 [E]

(evhx@evhx)-[~/Desktop]
```

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.



```
(evhx@evhx) [~/Desktop]
$ gpg --list-keys
/home/evhx/.gnupg/pubring.kbx
pub    rsa3072 2021-09-02 [SC] [expires: 2023-09-02]
      7F7946C5B66DB850D3ACDB2F15EACF261D1B1D51
uid          [ unknown] Christopher Smith <christopher.smith@csueastbay.edu>
sub    rsa3072 2021-09-02 [E] [expires: 2023-09-02]

pub    rsa4096 2021-09-10 [SC]
      15975E27CB4AFEC0CF8F10924D9455E2397D4A10
uid          [ultimate] Everlyn (Everlyns key) <eleon15@horizon.csueastbay.edu>
sub    rsa4096 2021-09-10 [E]

( evhx@evhx ) [ ~/Desktop ]
$ gpg --list-secret-keys
/home/evhx/.gnupg/pubring.kbx
sec    rsa4096 2021-09-10 [SC]
      15975E27CB4AFEC0CF8F10924D9455E2397D4A10
uid          [ultimate] Everlyn (Everlyns key) <eleon15@horizon.csueastbay.edu>
ssb    rsa4096 2021-09-10 [E]

( evhx@evhx ) [ ~/Desktop ]
$
```

The public key will be exported into a file named [NETID].public.key, using the command:

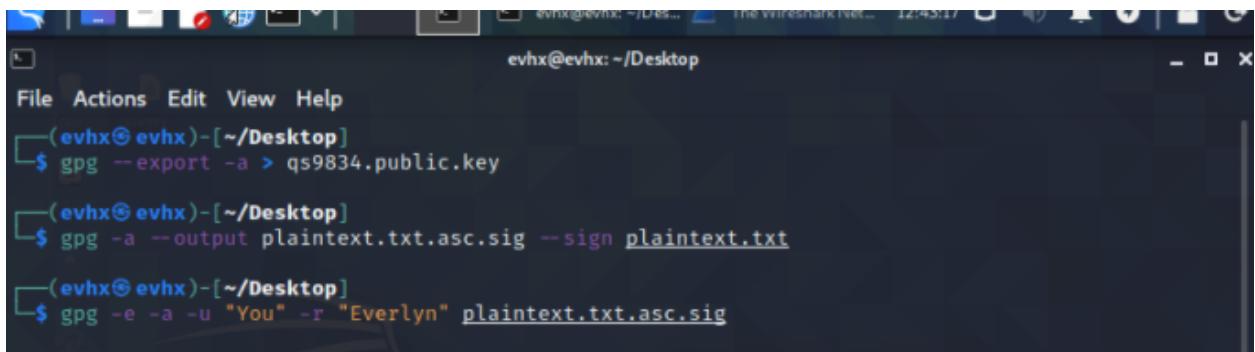
```
gpg --export -a > qs9834.public.key
```

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
```

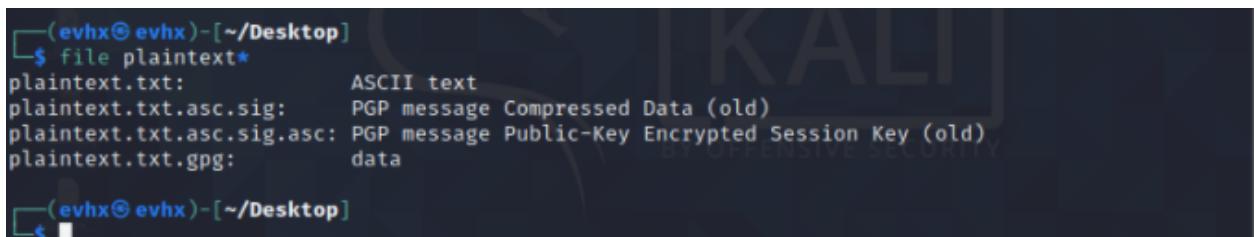


A screenshot of a terminal window titled "evhx@evhx: ~/Desktop". The window shows a command history with three entries:

- \$ gpg --export -a > qs9834.public.key
- \$ gpg -a --output plaintext.txt.asc.sig --sign plaintext.txt
- \$ gpg -e -a -u "You" -r "Everlyn" plaintext.txt.asc.sig

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

```
file plaintext*
```

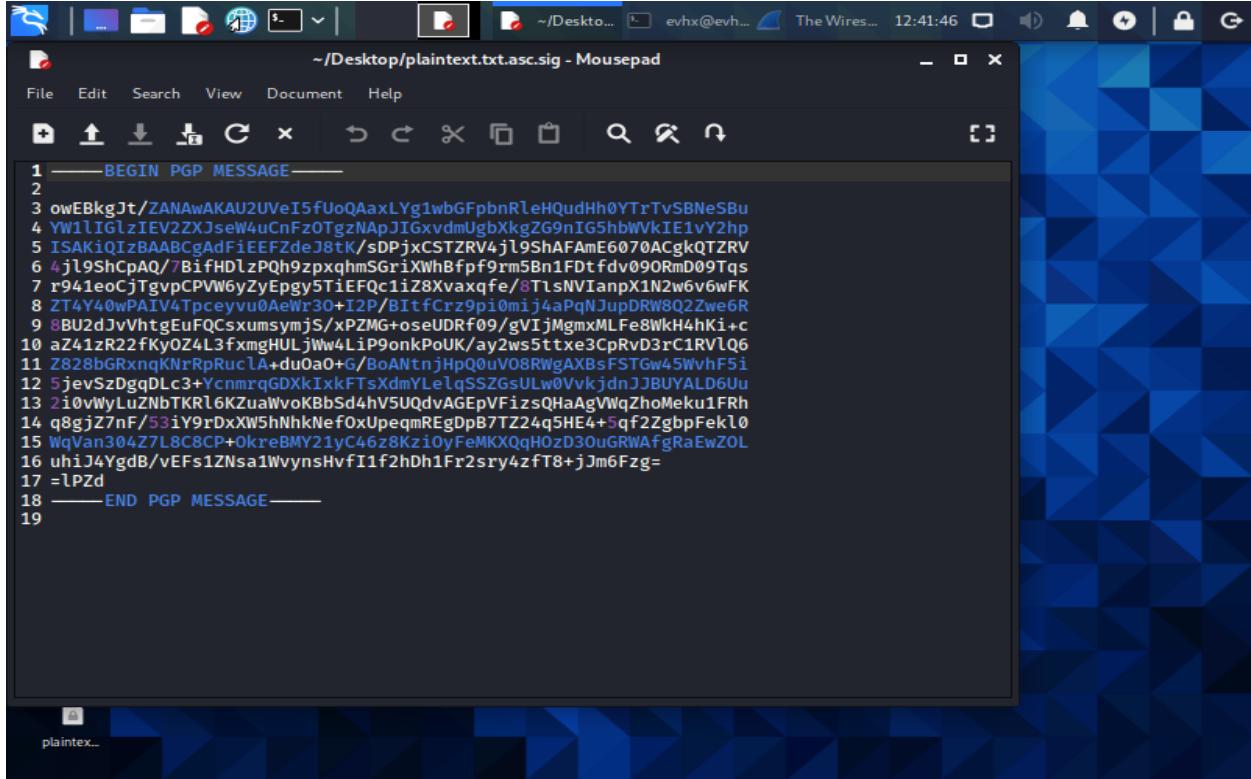


A screenshot of a terminal window titled "evhx@evhx: ~/Desktop". The window shows the output of the "file" command:

```
plaintxt.txt: ASCII text
plaintxt.txt.asc.sig: PGP message Compressed Data (old)
plaintxt.txt.asc.sig.asc: PGP message Public-Key Encrypted Session Key (old)
plaintxt.txt.gpg: data
```

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

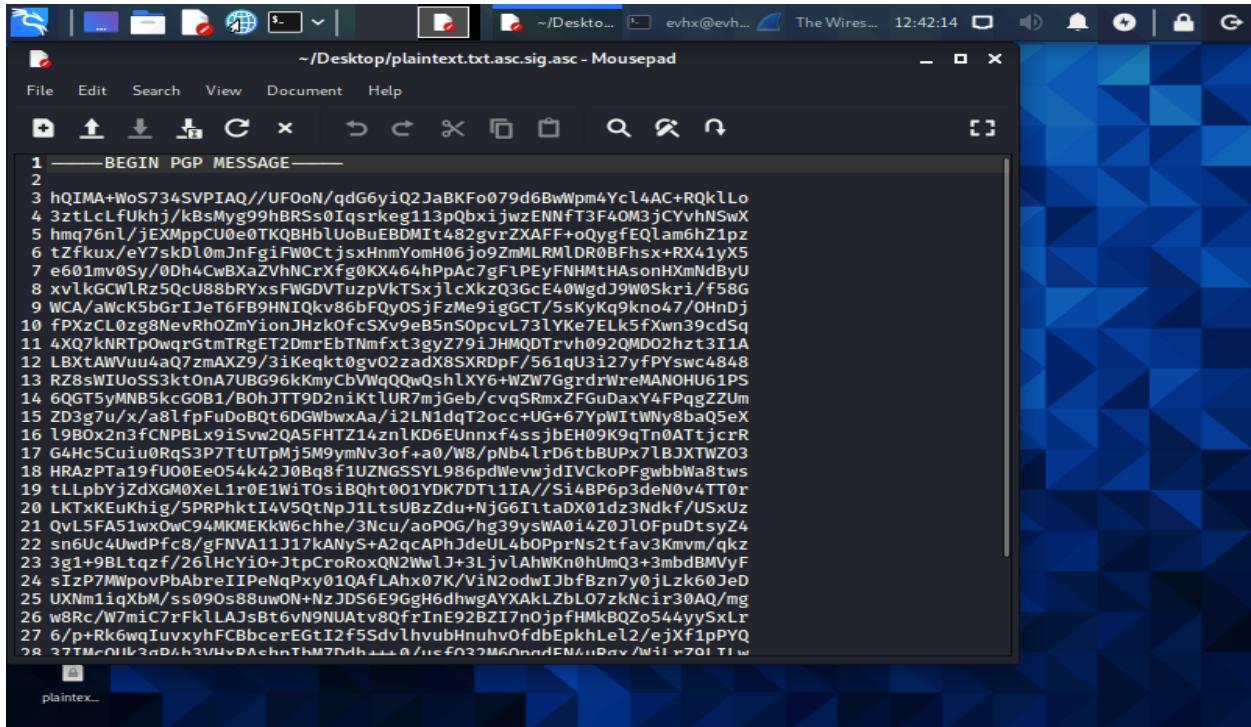
Public Key:



```
1 -----BEGIN PGP MESSAGE-----  
2  
3 owEBkgJt/ZANAwAKAU2UVeI5fuQAxaxLYg1wbGFpbRleHQudHh0YTrTvsBNeSbu  
4 YW1lIGlZIEV2ZXJseW4uCnFzOTgzNaPjGxvdmUgbXkgZG9nIG5hbWVkIE1vY2hp  
5 ISAKiQiZBaABCgAdfIEFzdeJ8tK/sDPjxCSTZRv4jlShAFAMeE6070AcgkQTZRV  
6 4jl9ShCpAQ/?BifHdlzPQh9zpxqhmSGriXWhBfpf9rm5Bn1FDtfdv090RmD09Tqs  
7 r941eoCjTgvpCPW6yZyEpgy5TiEFQc1iZ8Xvaxqf8/TlsNViinpX1N2w6vWFK  
8 ZT4Y40wPAIV6Tpceyyu0eWr30+I2P/BItfCrz9pi0mj4aPqNJuupDRW8Q2Zwe6R  
9 8BU2dJvVhtgEuFQCsxumsymjs/xPZMG+oseUDRf09/gVIjMgmxiMLFe8WkH4hKi+c  
10 aZ41zR22fKyO24L3fxmgHULjW4L1p0nkPoUK/ay2ws5txe3CprvD3rC1RVLQ6  
11 Z828bGRxnqKNrRpRucla+du0ao+G/BoANtnjhpo0uVO8RWgAXBsFSTGw45WvhF5i  
12 5jevSzDggQLC3+YcnmrqGDxkIxkFTsXdmYLelqSSZGzULw0VvkjdnJJBUYALD6uu  
13 2i0WwLuZnbTKRl6KZuaWvoKBbsd4hv5UQdvaGEpVFizsQHaAgVwqZhoMeku1Frh  
14 q8gjZ7nF/53iY9rDxXW5hNhkNef0xUpeqmREgDpB7TZ24q5HE4+5qf2ZgbpFekl0  
15 WqVan304Z7L8C8CP+OkreBMY21yC46z8KziOyFeMKXQh0zD3ouGRWAfgRaEwZOL  
16 uhiJ4YgdB/vEFs1ZNsaiWvynsHvfI1f2hDh1Fr2sry4zft8+jJm6Fzg=
```

17 =lPZd
18 -----END PGP MESSAGE-----
19

Private Key:



```
1 -----BEGIN PGP MESSAGE-----  
2  
3 hQIMA+WoS734SVPIAQ//UF0oN/qdG6yiQ2JaBKFo079d6BwWpm4Ycl4AC+RQkLlo  
4 3ztLcLf0zg8NevRh0ZmYhZkOfcVnx9eB5n5OpvcvL73lYke7ELK5fxWn39cdSq  
5 hmq76nl/jEXMpCUe0tQKOBHbLUoBuEBDMIt482gvrxZXAFF+oQygfEQlam6hZlpz  
6 tZfkux/eY7skDl0mJnFgiFW0CtjxsxHnmYomH06jo9ZmLMRLDR0BFhsx+RX41yX5  
7 e601mv0sSy/0Dh4CwB9NCrxFgk464nPpAc7gFLPeYfNHMtHasonHxmNdbyU  
8 xv1kGCWLrz5QcU88bRyx5FWGDVTuzpVktSjlcKxzQ3gcE40WgdJ9W0Skri/f58G  
9 WCA/aWcK5bGr1JeT6FB9HN1Qkv86bfQy0sjFzMe9igGCT/5sKyKq9kn047/0HnDj  
10 fPxZcL0zg8NevRh0ZmYhZkOfcVnx9eB5n5OpvcvL73lYke7ELK5fxWn39cdSq  
11 4X07kNR7p0wqrGtmTrgeT2DmxEbTnmfxt3gyZ79iJHM0dTrvh0920MD02hzt31IA  
12 LBxtAWVu04aQ7zmAXZ9/3iKeqkt0gv02zadX8SXRdpF/561qU3i27yfPYswc4848  
13 RZ8sWIUoS3ktOnA7UBG96kKmyCbWqQwQshlxY6+WZW7GgrdrWreMANOHU61PS  
14 6QGT5yMNB5kcG0B1/B0hJTT9D2niKtLUR7mjGeb/cvqSrmxzZFuGdaxY4FPqgZZUm  
15 ZD3g7u/x/a81fpFuDoBqt6DGWbwxA/i2LN1dqT2occ+UG+67YpWttWny8baQ5eX  
16 l9BXo2n3fCNPBLx9i5vw2QA5FHTZ14zn1KD6EUnmx4ssjbEHO9KqTn0AttjcrR  
17 G4Hc5Cuio0RqS3P7tUTPmJ5MyNmNv3of+a0/w8/pNb4lrD6tbUPx71BJXTWZ03  
18 HRAzPTa19fu00Ee054k42J0Bg8f1UZNGLSYL986pdWevwjdIVCkoPfgwbbWa8twS  
19 tLLpbYjZdXGM0XeL1r0E1wiToSiBQht001YDK7DTl1IA//Si4BP6p3deN0v4TT0r  
20 LKTxKEuKhig/5PRPhktI4V5QtNpJ1ltsUbZdu+NjG61taoX01dz3Ndkf/USxUz  
21 QvL5FA51wx0wC94MKMEEKKW6chhe/3Ncu/aoPOG/hg39ysWA0i4Z0jlOfpuDtSYz4  
22 sn6UC4UwdPfc8/gFNVA11317kAnys+A2qcApHJdeUL4bOPprNs2tfav3Kmvn/qkz  
23 3gl+9BLtzqzf/26lHcYi0+JtpCroRoxN2WwlJ+3lJv1AhWKn0hUmQ3+3mbdBMVyF  
24 sIzP7MWpovPbAbreIIPeNgPxv01QAFLhx07K/Vin2odwI1jbfbzn7y0jLzk60jeD  
25 UXNm1iqXBm/ss090s88uwON+NzJDSE69Ggh6dhwgAYXakLzbLo7zKncir30AQ/mg  
26 w8Rc/W7miC7rFk1LAJsBt6vN9NUAtv8QfrInE92BZI7n0jpffHMkBQZo544yySxLr  
27 6/p+Rk6wqIuvxyhFCBbcerEGt12f55dvlhvubHnuhv0fdbEpkhLeL2/ejxf1pPYQ  
28 37TMc0lk3gD4h2VlvPAsnThM7Ddh.../u.../uf023M60ngdEM.../u.../Pw.../Wii.../zo1TLw
```

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

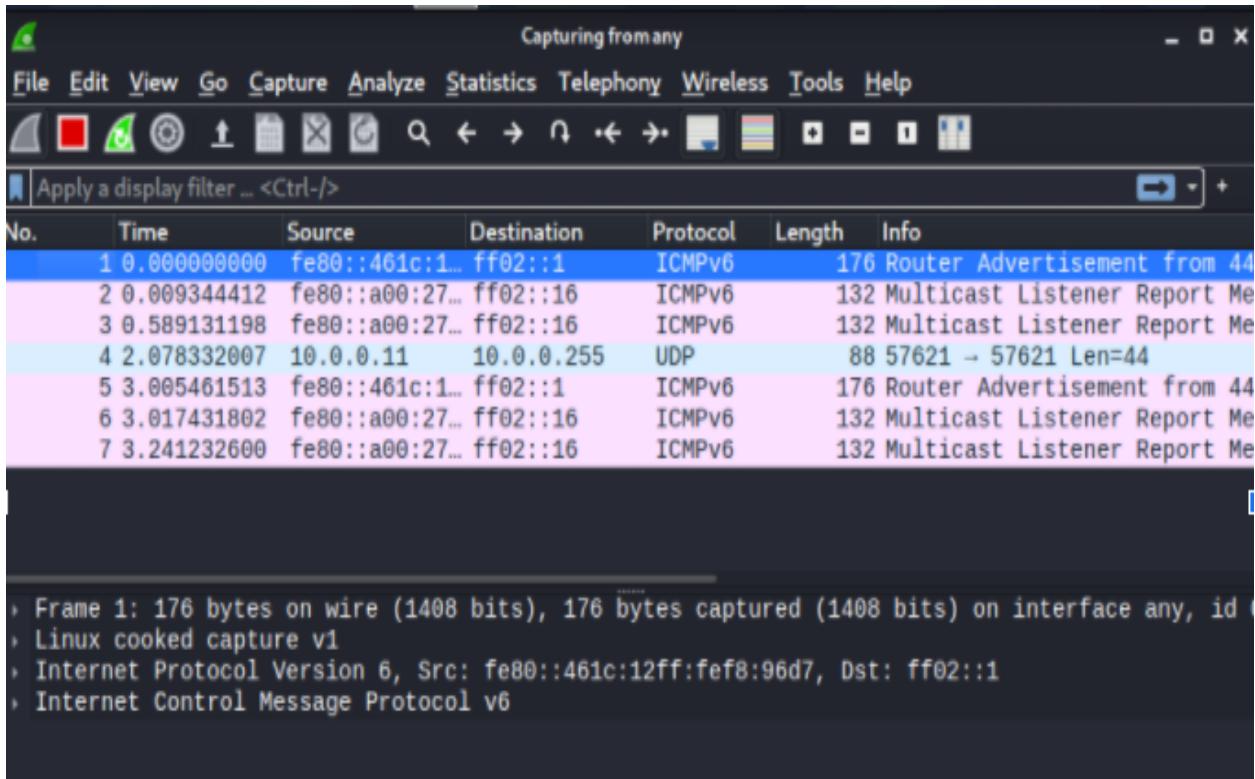
```
(evhx@evhx)-[~/Desktop]
$ ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            inet 10.0.0.136/24 brd 10.0.0.255 scope global dynamic noprefixroute eth0
            inet6 2601:644:203:1180::5acc/128 scope global dynamic noprefixroute
            inet6 2601:644:203:1180:c31e:93be:1eff:e20a/64 scope global temporary dynamic
            inet6 2601:644:203:1180:a00:27ff:fe1e:b91/64 scope global dynamic mngtmpaddr noprefixroute
            inet6 fe80::a00:27ff:fe1e:b91/64 scope link noprefixroute

(evhx@evhx)-[~/Desktop]
$
```

Ubuntu IP: 10.0.0.251

```
evhx@evhx-VirtualBox:~/Desktop$ ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            inet 10.0.0.251/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s3
            inet6 2601:644:203:1180::c6b7/128 scope global dynamic noprefixroute
            inet6 2601:644:203:1180:302e:e325:1cd1:5a9f/64 scope global temporary dynam
ic
            inet6 2601:644:203:1180:8c69:4aed:c4e7:ad1d/64 scope global dynamic mngtmpa
ddr noprefixroute
            inet6 fe80::cd7d:a7f5:53bd:4bd5/64 scope link noprefixroute
evhx@evhx-VirtualBox:~/Desktop$
```

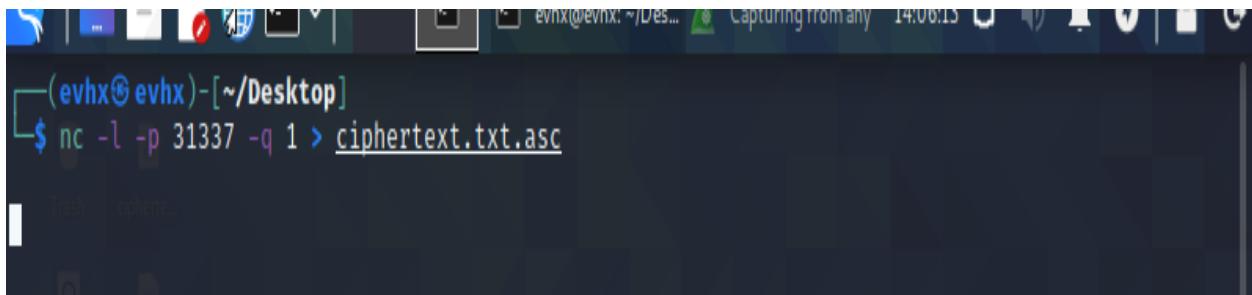
Wireshark will be used because it provides a variety of tools to make network packets easy to analyze, such as a display filter, the list of captured packets, the details to any selected packet, and the ASCII and hexadecimal contained within the packets. To start reading the packages, press the blue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

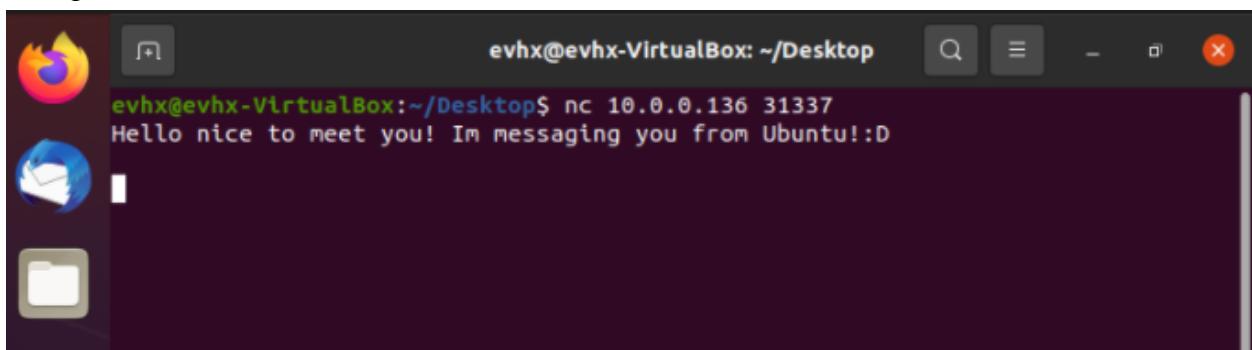


A screenshot of a terminal window on Kali Linux. The terminal title is '(evhx@evhx)-[~/Desktop]'. The command entered is '\$ nc -l -p 31337 -q 1 > ciphertext.txt.asc'. The terminal shows the command being typed and then executed.

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:

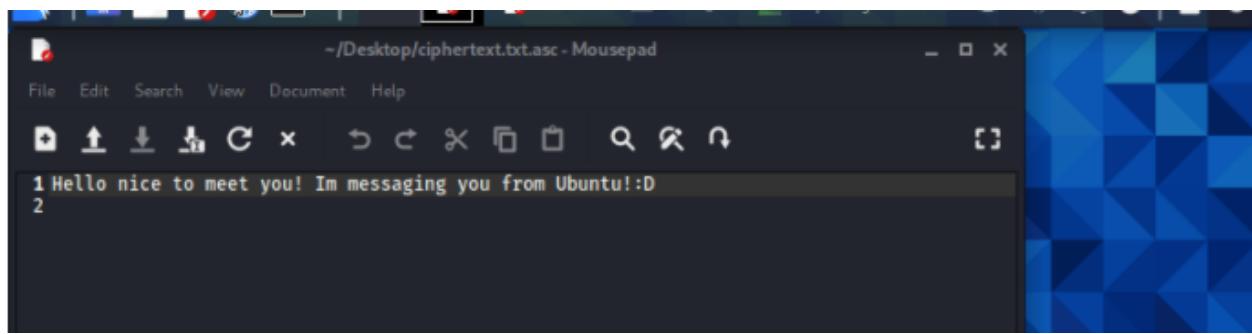
```
nc [IP address] [port number]
```

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



A screenshot of a terminal window on Ubuntu. The terminal title is 'evhx@evhx-VirtualBox: ~/Desktop'. The command entered is '\$ nc 10.0.0.136 31337'. The message 'Hello nice to meet you! Im messaging you from Ubuntu!:D' is typed into the terminal and sent.

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

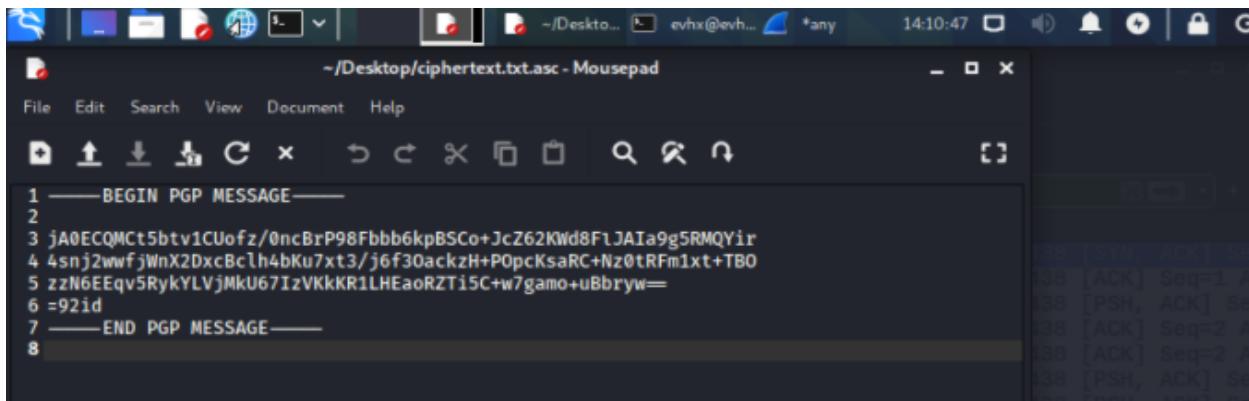


A screenshot of a Mousepad application window on Kali Linux. The file path is '/Desktop/ciphertext.txt.asc'. The content of the file is displayed as two lines of text: '1 Hello nice to meet you! Im messaging you from Ubuntu!:D' and '2'. The application interface includes a toolbar with various icons and a menu bar.

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

```
evhx@evhx-VirtualBox: ~/Desktop$ cat ciphertext.txt.asc | netcat 10.0.0.136 31337  
evhx@evhx-VirtualBox: ~/Desktop$
```

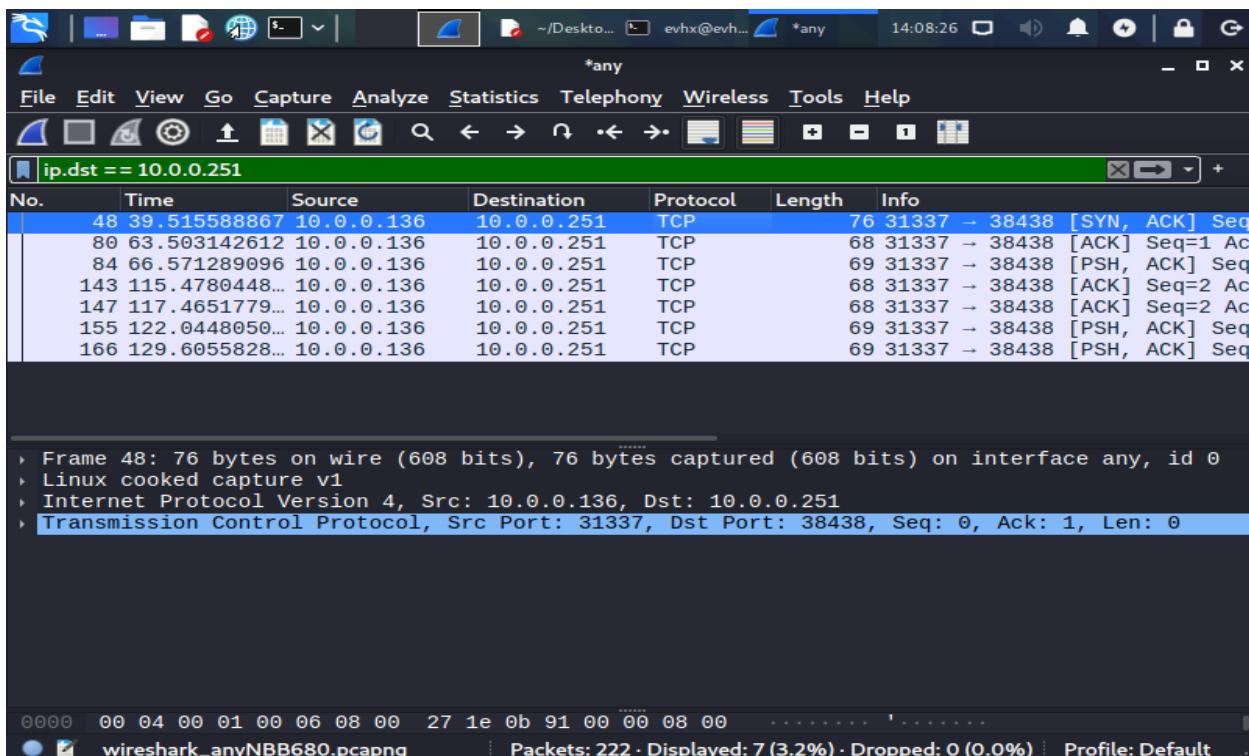
On the Kali Linux machine, view the ciphertext.txt.asc, which now contains the ASCII armored encrypted file.



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:

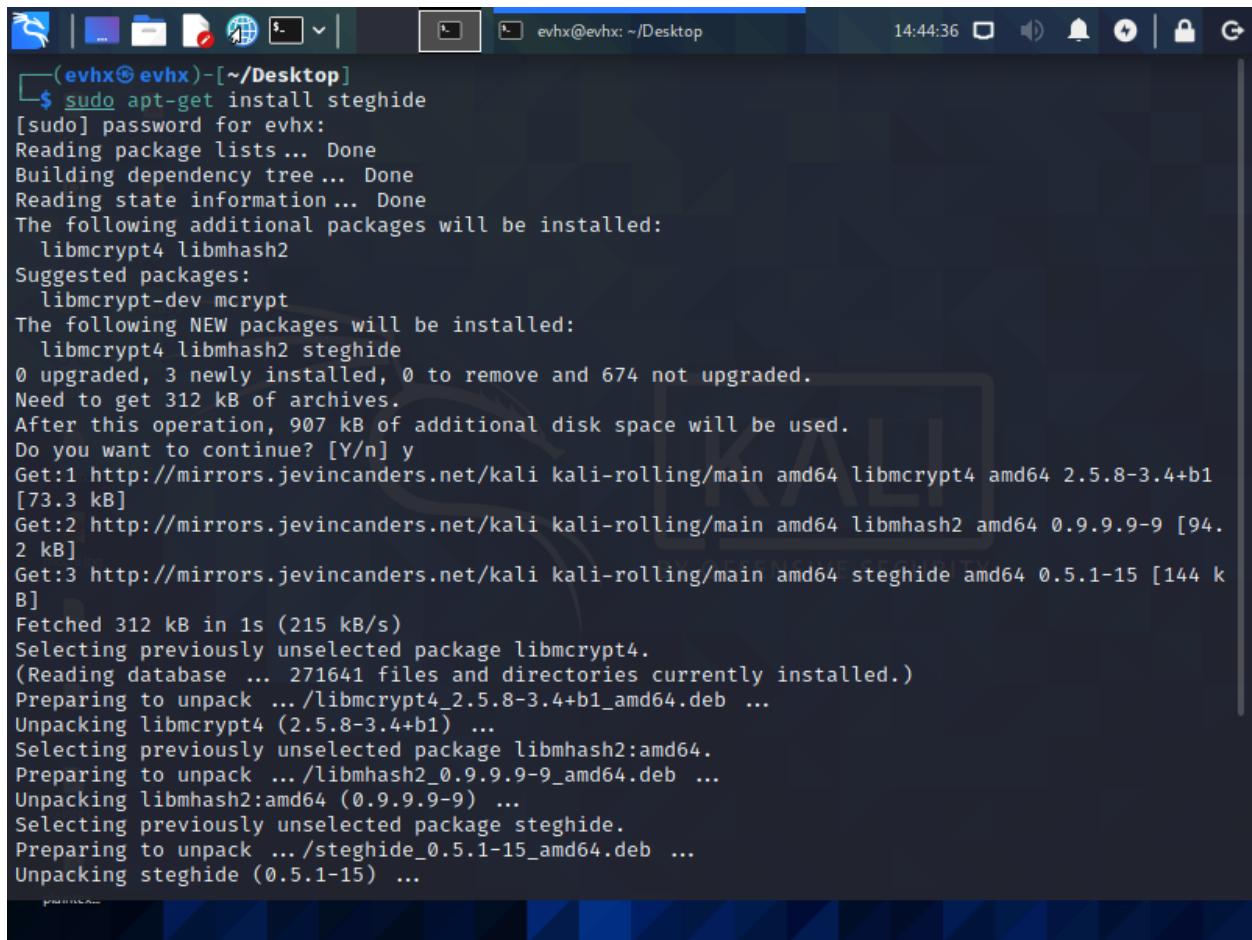
ip.dst==10.0.0.251 OR ip.addr==10.0.0.251

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

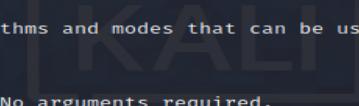


```
(evhx@evhx) [~/Desktop]
$ sudo apt-get install steghide
[sudo] password for evhx:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libmcrypt4 libmhash2
Suggested packages:
  libmcrypt-dev mcrypt
The following NEW packages will be installed:
  libmcrypt4 libmhash2 steghide
0 upgraded, 3 newly installed, 0 to remove and 674 not upgraded.
Need to get 312 kB of archives.
After this operation, 907 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://mirrors.jevincanders.net/kali kali-rolling/main amd64 libmcrypt4 amd64 2.5.8-3.4+b1 [73.3 kB]
Get:2 http://mirrors.jevincanders.net/kali kali-rolling/main amd64 libmhash2 amd64 0.9.9.9-9 [94.2 kB]
Get:3 http://mirrors.jevincanders.net/kali kali-rolling/main amd64 steghide amd64 0.5.1-15 [144 kB]
Fetched 312 kB in 1s (215 kB/s)
Selecting previously unselected package libmcrypt4.
(Reading database ... 271641 files and directories currently installed.)
Preparing to unpack .../libmcrypt4_2.5.8-3.4+b1_amd64.deb ...
Unpacking libmcrypt4 (2.5.8-3.4+b1) ...
Selecting previously unselected package libmhash2:amd64.
Preparing to unpack .../libmhash2_0.9.9.9-9_amd64.deb ...
Unpacking libmhash2:amd64 (0.9.9.9-9) ...
Selecting previously unselected package steghide.
Preparing to unpack .../steghide_0.5.1-15_amd64.deb ...
Unpacking steghide (0.5.1-15) ...
```

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

The Steghide commands that will be used 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)



evhx@evhx: ~/Desktop 14:44:52

COMMANDS

In this section the commands for steghide are listed. The first argument must always be one of these commands. You can supply additional arguments to the **embed**, **extract** and **info** commands. The other commands do not take any arguments.

embed, --embed
Embed secret data in a cover file thereby creating a stego file.

extract, --extract
Extract secret data from a stego file.

info, --info
Display information about a cover or stego file.

encinfo, --encinfo
Display a list of encryption algorithms and modes that can be used. No arguments required.

version, --version
Display short version information. No arguments required.

license, --license
Display steghide's license. No arguments required.

help, --help
Display a help screen. No arguments required.

EMBEDDING

You should use the **embed** command if you want to embed secret data in a cover file. The following arguments can be used with the **embed** command:

-ef, --embedfile filename
Manual page steghide(1) line 43 (press h for help or q to quit)

EMBEDDING

You should use the **embed** command if you want to embed secret data in a cover file. The following arguments can be used with the **embed** command:

-ef, --embedfile filename
Specify the file that will be embedded (the file that contains the secret message). Note that steghide embeds the original file name in the stego file. When extracting data (see below) the default behaviour is to save the embedded file into the current directory under its original name. If this argument is omitted or **filename** is **-**, steghide will read the secret data from standard input.

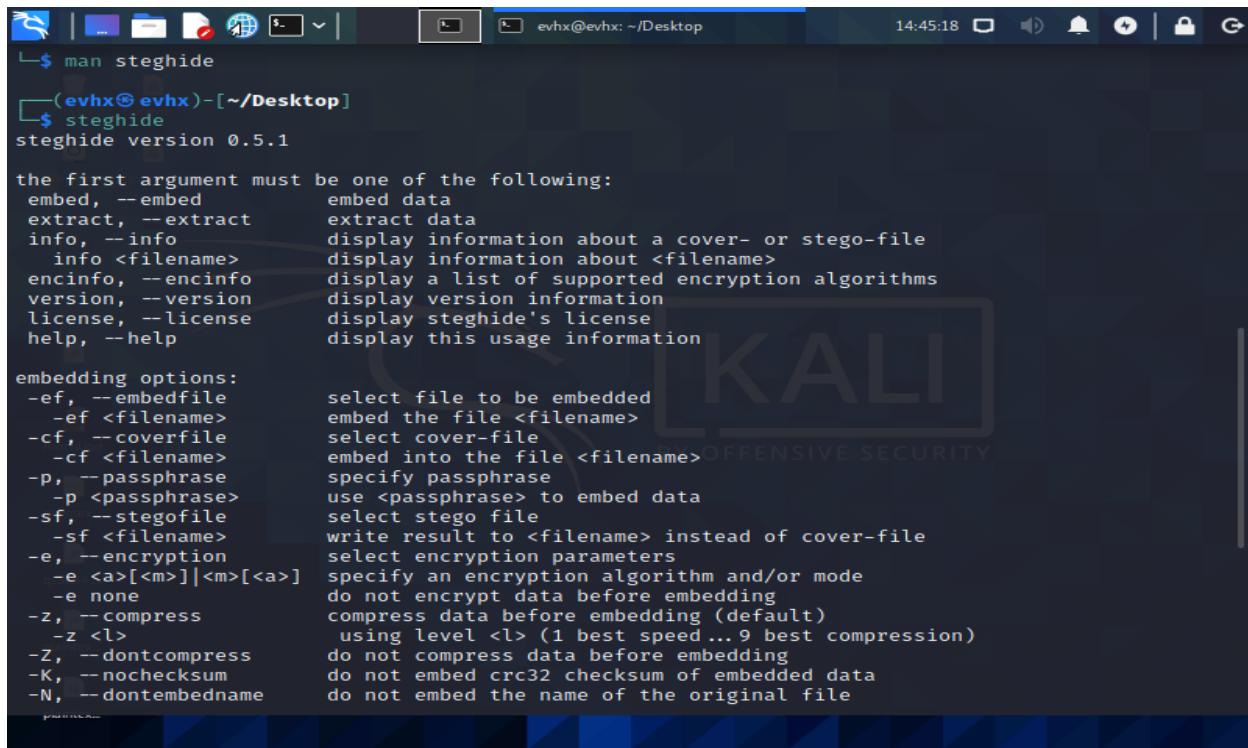
-cf, --coverfile filename
Specify the cover file that will be used to embed data. The cover file must be in one of the following formats: AU, BMP, JPEG or WAV. The file-format will be detected automatically based on header information (the extension is not relevant). If this argument is omitted or **filename** is **-**, steghide will read the cover file from standard input.

-sf, --stegofile filename
Specify the name for the stego file that will be created. If this argument is omitted when calling steghide with the **embed** command, then the modifications to embed the secret data will be made directly to the cover file without saving it under a new name.

-e, --encryption algo [mode] | mode [algo]
Specify encryption parameters. This option must be followed by one or two strings that identify an encryption algorithm and/or mode. You can get the names of all available algorithms and supported modes with the **encinfo** command. The default encryption is **rijndael-128** (AES) in the **cbc** mode. If you do not want to use any encryption, use **-e none**.

-z, --compress level
Manual page steghide(1) line 70 (press h for help or q to quit)

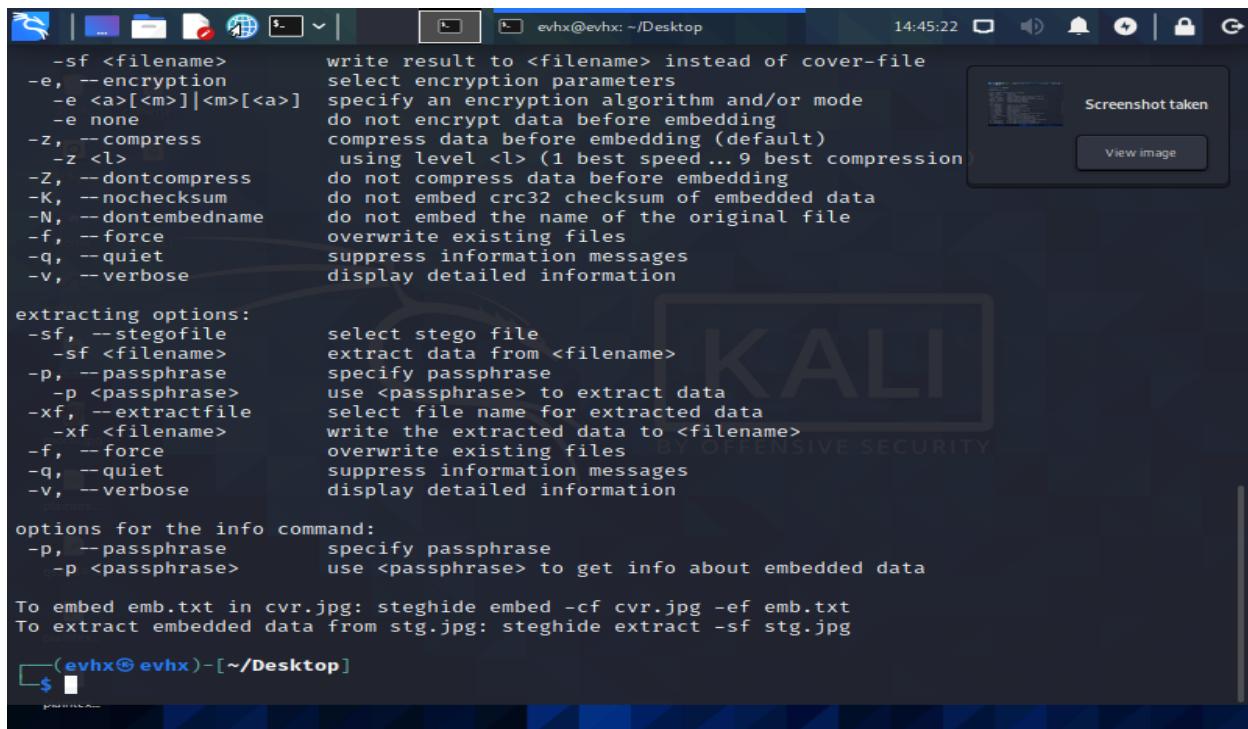
Another way to view the Steghide commands is to simply type the command:
steghide



(evhx@evhx)-[~/Desktop]\$ man steghide
(evhx@evhx)-[~/Desktop]\$ steghide
steghide version 0.5.1

the first argument must be one of the following:
embed, --embed embed data
extract, --extract extract data
info, --info display information about a cover- or stego-file
info <filename> display information about <filename>
encinfo, --encinfo display a list of supported encryption algorithms
version, --version display version information
license, --license display steghide's license
help, --help display this usage information

embedding options:
-ef, --embedfile select file to be embedded
-ef <filename> embed the file <filename>
-cf, --coverfile select cover-file
-cf <filename> embed into the file <filename>
-p, --passphrase specify passphrase
-p <passphrase> use <passphrase> to embed data
-sf, --stegofile select stego file
-sf <filename> write result to <filename> instead of cover-file
-e, --encryption select encryption parameters
-e <a>[<m>]|<m>[<a>] specify an encryption algorithm and/or mode
-e none do not encrypt data before embedding
-z, --compress compress data before embedding (default)
-z <l> using level <l> (1 best speed...9 best compression)
-Z, --dontcompress do not compress data before embedding
-K, --nochecksum do not embed crc32 checksum of embedded data
-N, --dontembedname do not embed the name of the original file



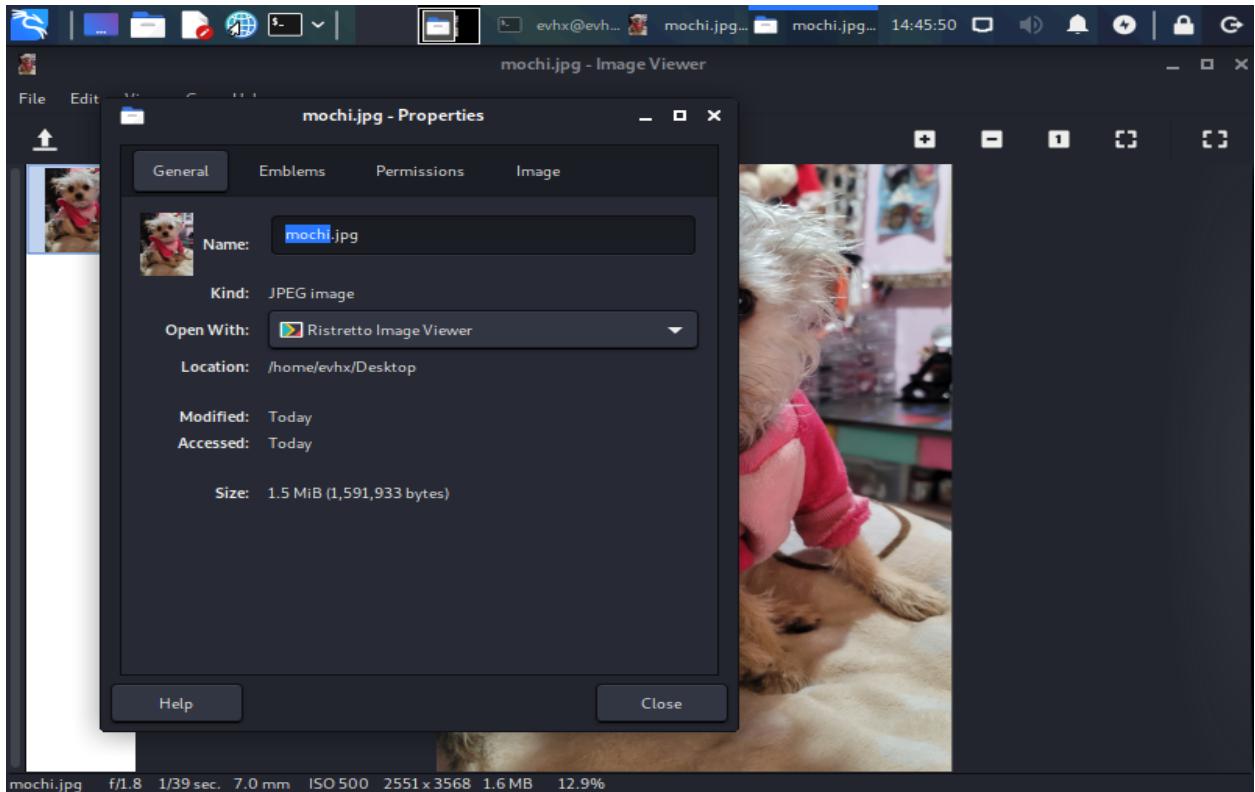
-sf <filename> write result to <filename> instead of cover-file
-e, --encryption select encryption parameters
-e <a>[<m>]|<m>[<a>] specify an encryption algorithm and/or mode
-e none do not encrypt data before embedding
-z, --compress compress data before embedding (default)
-z <l> using level <l> (1 best speed...9 best compression)
-Z, --dontcompress do not compress data before embedding
-K, --nochecksum do not embed crc32 checksum of embedded data
-N, --dontembedname do not embed the name of the original file
-f, --force overwrite existing files
-q, --quiet suppress information messages
-v, --verbose display detailed information

extracting options:
-sf, --stegofile select stego file
-sf <filename> extract data from <filename>
-p, --passphrase specify passphrase
-p <passphrase> use <passphrase> to extract data
-xf, --extractfile select file name for extracted data
-xf <filename> write the extracted data to <filename>
-f, --force overwrite existing files
-q, --quiet suppress information messages
-v, --verbose display detailed information

options for the info command:
-p, --passphrase specify passphrase
-p <passphrase> use <passphrase> to get info about embedded data

To embed emb.txt in cvr.jpg: steghide embed -cf cvr.jpg -ef emb.txt
To extract embedded data from stg.jpg: steghide extract -sf stg.jpg

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:

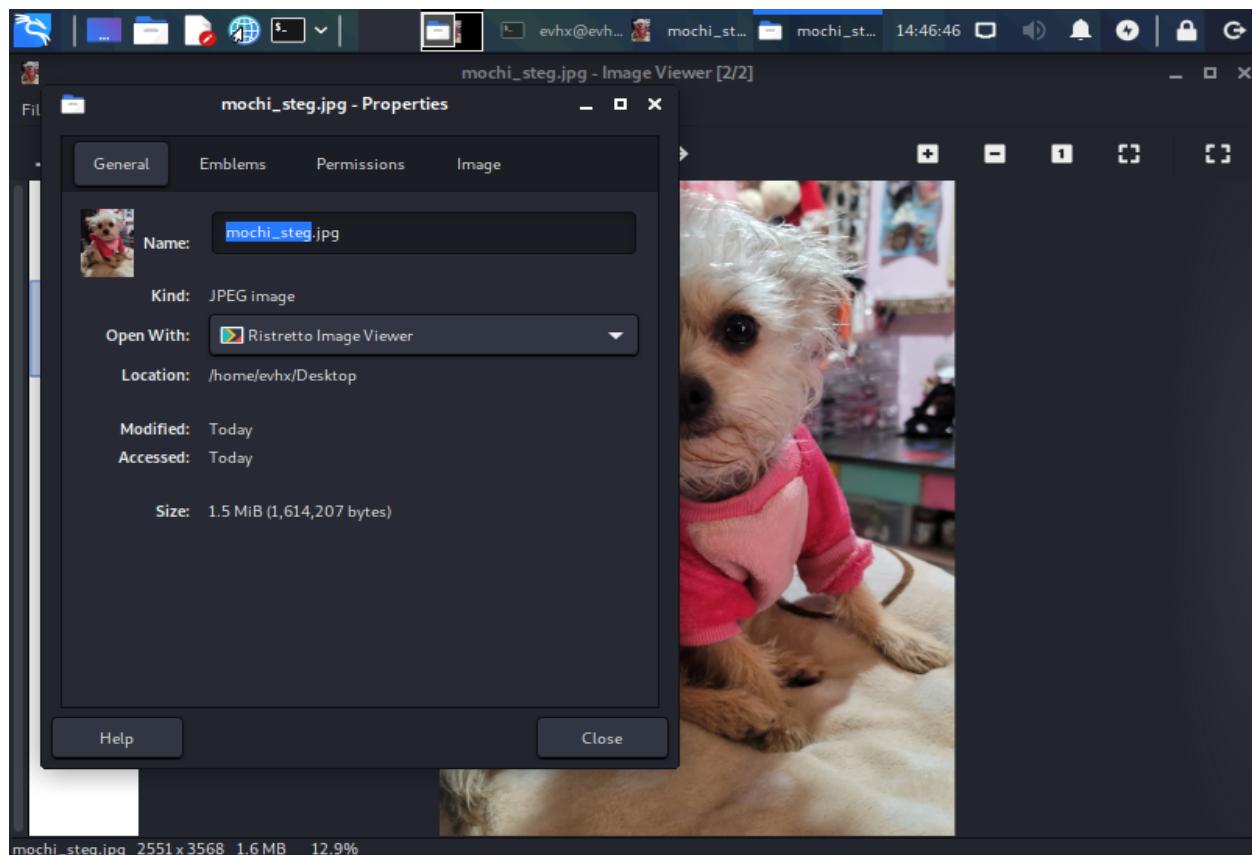
```
steghide embed -ef plaintext.txt -cf mochi.jpg -sf mochi_steg.jpg -p letmein
```

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
```

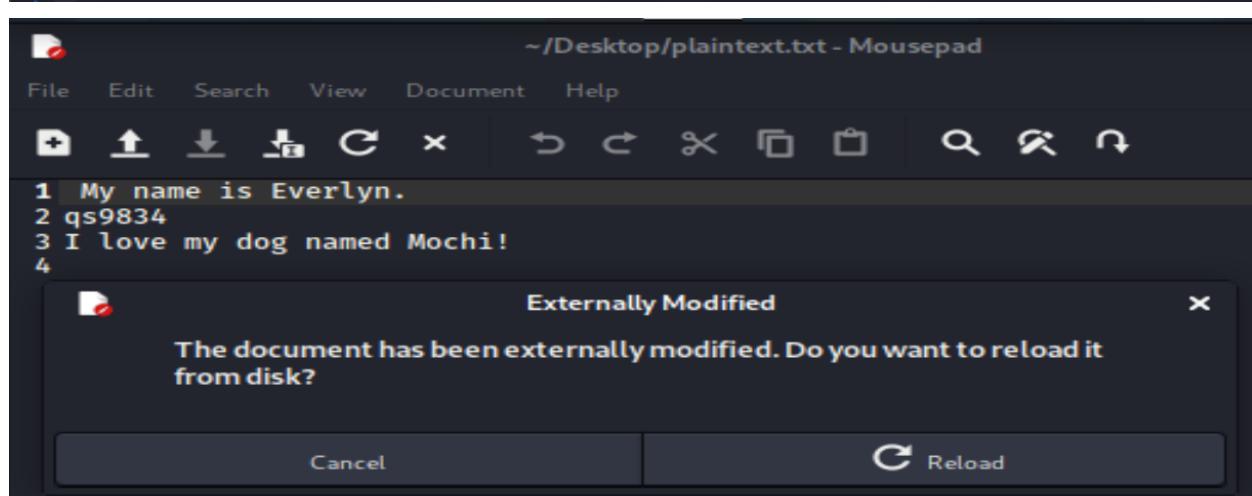
The mochi_stego.jpg file still looks exactly the same as the original mochi.jpg file, but the number of bytes was increased to 1,614,207 bytes.

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_stego.jpg -xf plaintext.txt

```
(evhx@evhx)@[~/Desktop]
$ steghide extract -sf mochi_stego.jpg -xf plaintext.txt
Enter passphrase:
the file "plaintext.txt" does already exist. overwrite ? (y/n) y
wrote extracted data to "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: 7c2cf3aa85d6112a5653f53970cf28 )  
md5sum mochi_steg.jpg  
( results: ceaf5a89750e6be5af38333f3fb46290 )  
md5sum mochi_renamed.jpg  
( results: 7c2cf3aa85d6112a5653f53970cf28 )
```

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.

```
evhx@evhx: ~/Desktop
```

3	High	96 ms	High	Unresponsive
4	Nightmare	480 ms	Insane	Headless

- [Basic Examples] -

Attack-Mode	Hash-Type	Example command
Wordlist	\$P\$	hashcat -a 0 -m 400 example400.hash example.dict
Wordlist + Rules	MD5	hashcat -a 0 -m 0 example0.hash example.dict -r rules/best64.rule
Brute-Force	MD5	hashcat -a 3 -m 0 example0.hash ?a?a?a?a?a
Combinator	MD5	hashcat -a 1 -m 0 example0.hash example.dict example.dict

If you still have no idea what just happened, try the following pages:

- * https://hashcat.net/wiki/#howtos_videos_papers_articles_etc_in_the_wild
- * <https://hashcat.net/faq/>

```
(evhx@evhx)-[~/Desktop]
$ md5sum mochi.jpg
7c2cf3aa85d6112a5653f539730cf28 mochi.jpg

(evhx@evhx)-[~/Desktop]
$ md5sum mochi_steg.jpg
ceaf5a89750e6be5af38333f3fb46290 mochi_steg.jpg

(evhx@evhx)-[~/Desktop]
$ md5sum mochi_renamed.jpg
7c2cf3aa85d6112a5653f539730cf28 mochi_renamed.jpg

(evhx@evhx)-[~/Desktop]
$
```

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.