Assignment 2

Issiah Deleon

Abstract

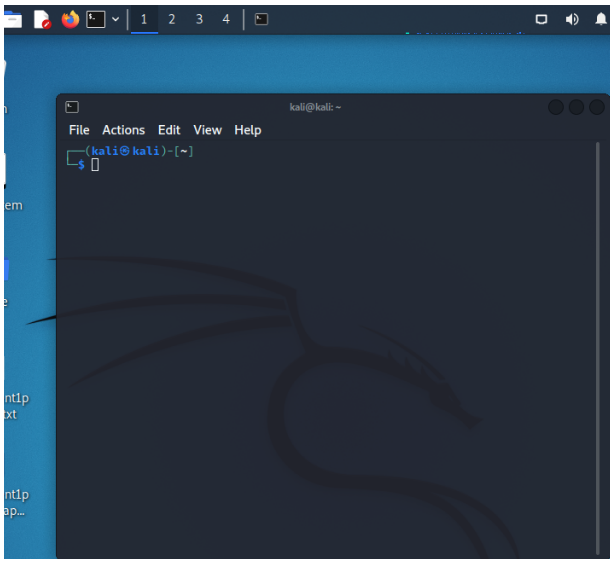
The intent of this lab is to work with symmetric and asymmetric encryption using the following tools: Kali Linux virtual machine, gpg, md5sum, and steghide. These tools will aid in our understanding and implementation of symmetric and asymmetric encryption practices using pseudo data, and we will demonstrate the effectiveness of such practices.

Introduction

Kali Linux will be used within a virtual machine using Oracle VM VirtualBox Manager. Within this machine, we will use gpg, md5sum, and steghide tools within the VM terminal to encrypt a simple text file containing pseudo secret information. We will then use netcat to transmit the encrypted version of this text file and intercept it using wireshark. Steghide is not included in this version of Kali Linux, and will need to be installed.

Summary

Begin by opening the Kali Linux virtual machine, and logging in using the "kali/kali" username and password. Open a terminal by clicking on the black box with a $ inside at the top left.

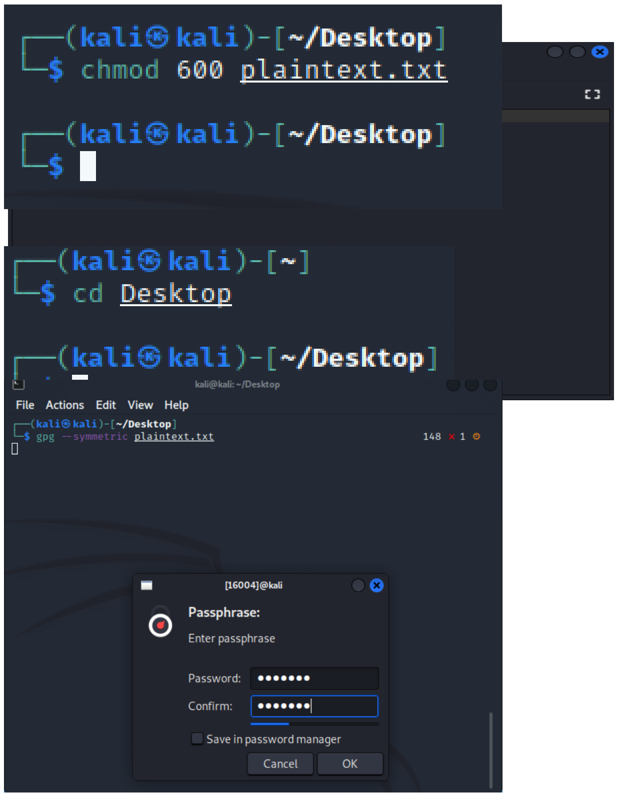


```
┌──(kali㉿kali)-[~]
└─$ 
```

Then, create a new plain text file called plaintext.txt by right clicking a blank space on the desktop and selecting “New Document”. Add any pseudo “secret” information we would like for demonstration purposes as shown below.

Next, go to the terminal window. Begin by changing the permissions of the plaintext.txt file by first changing the directory with the following command: cd Desktop. This allows us to access the plaintext.txt file.cd

Proceed by changing the permissions of the file using the following command: chmod 600 plaintext.txt.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ chmod 600 plaintext.txt

┌──(kali㉿kali)-[~/Desktop]
└─$
```

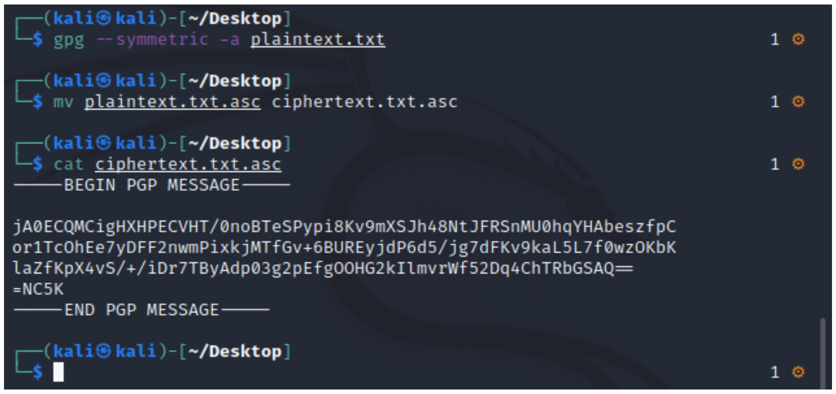
```
┌──(kali㉿kali)-[~]
└─$ cd Desktop

┌──(kali㉿kali)-[~/Desktop]
```

```
kali@kali: ~/Desktop
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg --symmetric plaintext.txt                                148 ✕ 1 ⚙
```

Then, begin the process of symmetric encryption by entering this command: gpg –symmetric -a plaintext.txt. we will then be prompted to enter a password. We will use “letmein”.

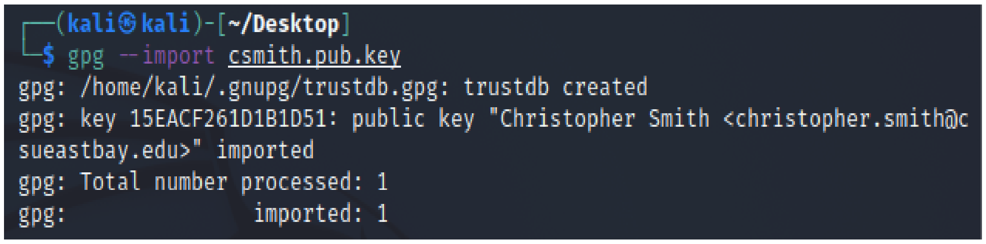
Now we will encrypt the file to ascii text so that we can copy and paste the encrypted text if needed. Using the commands: mv plaintext.txt.asc ciphertext.txt.asc will save the cipher text to a new file, ciphertext.txt, and cat ciphertext.txt.asc will display the cipher text as shown.

We will now prepare the public key used for asymmetric encryption. Begin by saving the csmith.pub.key file to the virtual machine's desktop. Then, type in the following command in the



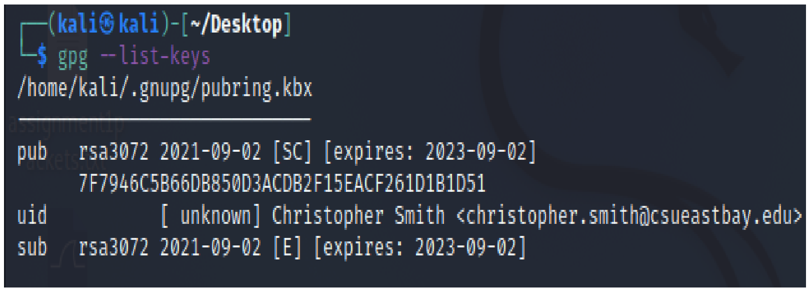
```
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg --symmetric -a plaintext.txt                                    1 ⚙

┌──(kali㉿kali)-[~/Desktop]
└─$ mv plaintext.txt.asc ciphertext.txt.asc                             1 ⚙

┌──(kali㉿kali)-[~/Desktop]
└─$ cat ciphertext.txt.asc                                              1 ⚙
-----BEGIN PGP MESSAGE-----

jA0ECQMCigHXHPECVHT/0noBTeSPypi8Kv9mXSJh48NtJFRSnMU0hqYHAbeszfpC
or1TcOhEe7yDFF2nwmPixkjMTfGv+6BUREyjdP6d5/jg7dFKv9kaL5L7f0wzOKbK
laZfKpX4vS/+/iDr7TByAdp03g2pEfgOOHG2kIlmvrWf52Dq4ChTRbGSAQ==
=NC5K
-----END PGP MESSAGE-----

┌──(kali㉿kali)-[~/Desktop]
└─$                                                                     1 ⚙
```

terminal: gpg –import csmith.pub.key.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg --import csmith.pub.key
gpg: /home/kali/.gnupg/trustdb.gpg: trustdb created
gpg: key 15EACF261D1B1D51: public key "Christopher Smith <christopher.smith@c
sueastbay.edu>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

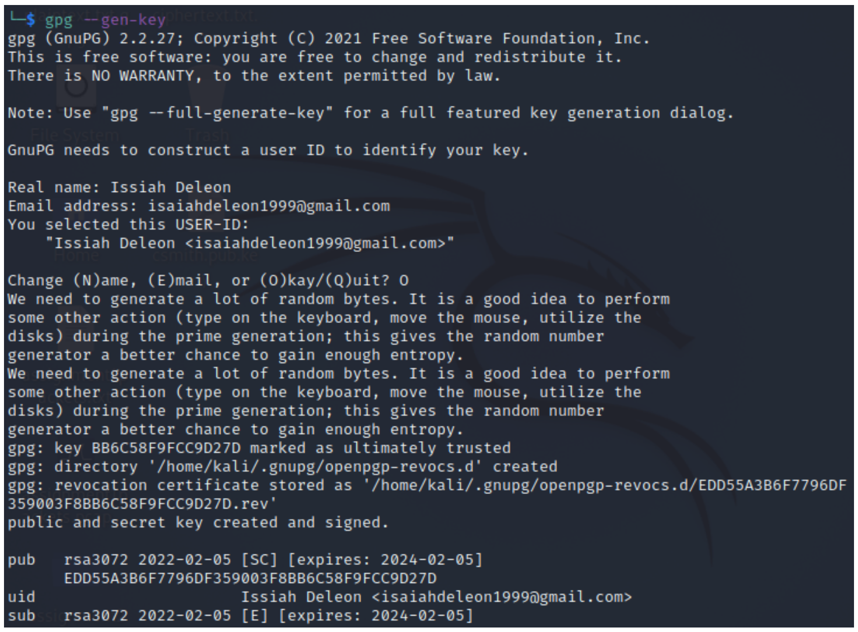
Now, list the keys by typing in: gpg –list-keys.



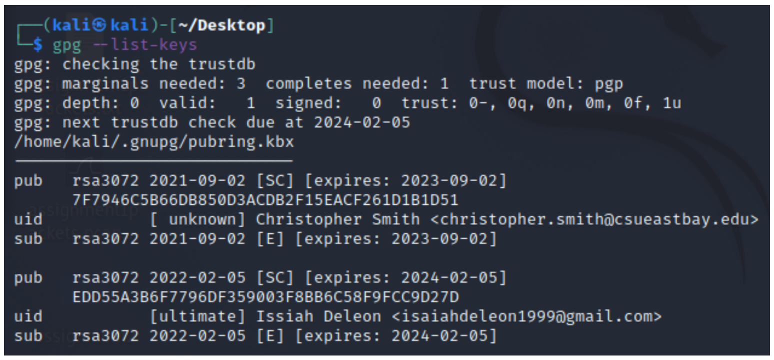
```
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg --list-keys
/home/kali/.gnupg/pubring.kbx
-----------------------------
pub   rsa3072 2021-09-02 [SC] [expires: 2023-09-02]
      7F7946C5B66DB850D3ACDB2F15EACF261D1B1D51
uid           [ unknown] Christopher Smith <christopher.smith@csueastbay.edu>
sub   rsa3072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

Now we will create a public/private key pair. Begin this process by typing in the command: gpg –gen-key. we will then be prompted to enter the real name and email address, as well as a password. Again use the password "letmein" and we will then finalize the process by typing in O for (O)kay. the key is provided on the line "gpg: key XXXXXXXXXXX".

To demonstrate that new keys have been generated, type in the command gpg –list-keys again.

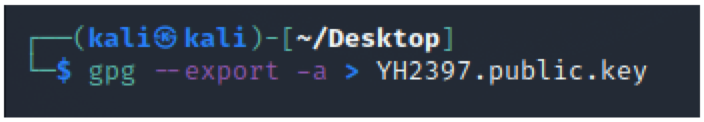


```
└─$ gpg --gen-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

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

Real name: Issiah Deleon
Email address: isaiahdeleon1999@gmail.com
You selected this USER-ID:
    "Issiah Deleon <isaiahdeleon1999@gmail.com>"

Change (N)ame, (E)mail, or (O)kay/(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 BB6C58F9FCC9D27D marked as ultimately trusted
gpg: directory '/home/kali/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/kali/.gnupg/openpgp-revocs.d/EDD55A3B6F7796DF
359003F8BB6C58F9FCC9D27D.rev'
public and secret key created and signed.

pub   rsa3072 2022-02-05 [SC] [expires: 2024-02-05]
      EDD55A3B6F7796DF359003F8BB6C58F9FCC9D27D
uid                      Issiah Deleon <isaiahdeleon1999@gmail.com>
sub   rsa3072 2022-02-05 [E] [expires: 2024-02-05]
```

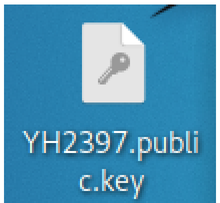


```
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2024-02-05
/home/kali/.gnupg/pubring.kbx
─────────────────────────────
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   rsa3072 2022-02-05 [SC] [expires: 2024-02-05]
      EDD55A3B6F7796DF359003F8BB6C58F9FCC9D27D
uid           [ultimate] Issiah Deleon <isaiahdeleon1999@gmail.com>
sub   rsa3072 2022-02-05 [E] [expires: 2024-02-05]
```

Export the newly generated key to the desktop by using the following command: gpg –export -a > theNETID.public.key

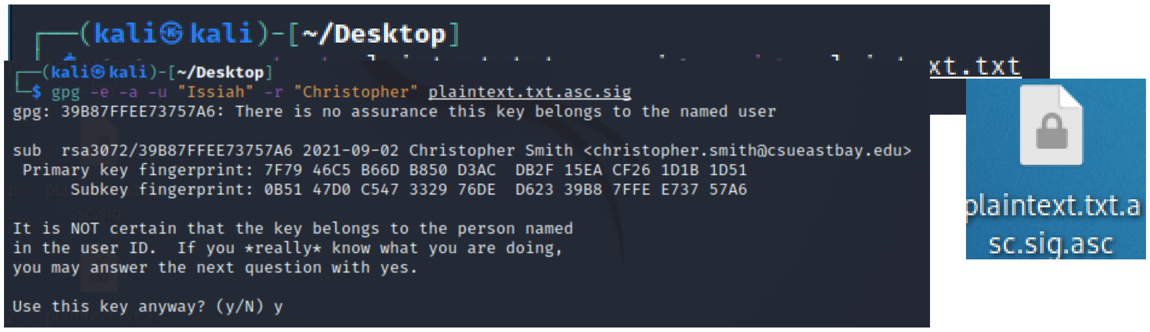


```
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg --export -a > YH2397.public.key
```



Now we will sign the plaintext file with the newly generated private key. To do so, type in the following command into the terminal: gpg -a –output plaintext.txt.asc.sig –sign plaintext.txt. we will be prompted to enter the password we chose when generating the private key.

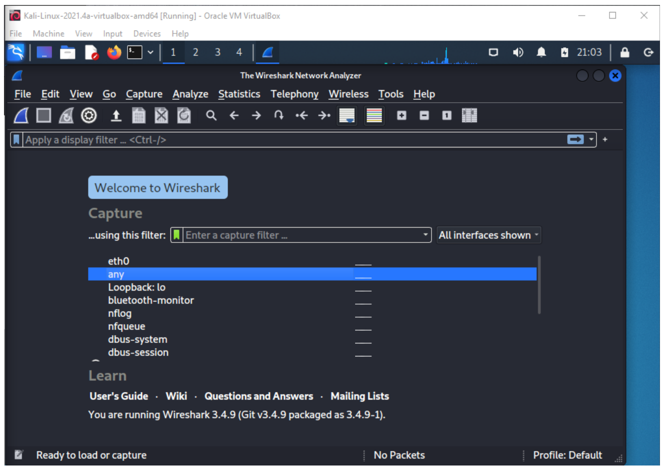
Now we will encrypt the signed file to Ascii output using the provided key by entering the



```
┌──(kali㉿kali)-[~/Desktop]
└─$ gpg -e -a -u "Issiah" -r "Christopher" plaintext.txt.asc.sig
gpg: 39B87FFEE73757A6: There is no assurance this key belongs to the named user

sub  rsa3072/39B87FFEE73757A6 2021-09-02 Christopher Smith <christopher.smith@csueastbay.edu>
 Primary key fingerprint: 7F79 46C5 B66D B850 D3AC  DB2F 15EA CF26 1D1B 1D51
      Subkey fingerprint: 0B51 47D0 C547 3329 76DE  D623 39B8 7FFE E737 57A6

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

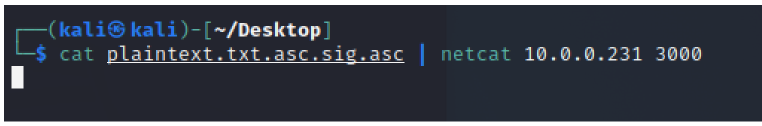
following command: gpg -e -a -u “thename” -r “Christopher” plaintext.txt.asc.sig. we should see this signed file appear on the desktop titled “plaintext.txt.a.sc.sig.asc”

Now we will demonstrate the interception of ascii encrypted text using two terminals and wireshark. Begin by opening WireShark and listening to “any" channels. Next, open another terminal in addition to the first one. In this new terminal, enter the command: nc -l -p 3000. This will establish the terminal as the “listener” terminal. In the second terminal that we have been using, enter the command: cat plaintext.txt.asc.sig.asc | netcat 10.0.0.231 3000.

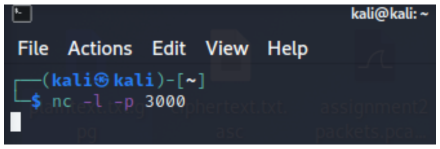


Kali-Linux-2021.4a-virtualbox-amd64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
21:03
The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
eth0
any
Loopback: lo
bluetooth-monitor
nflog
nfqueue
dbus-system
dbus-session
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists
Ready to load or capture
No Packets
Profile: Default

You are running Wireshark 3.4.9 (Git v3.4.9 packaged as 3.4.9-1).

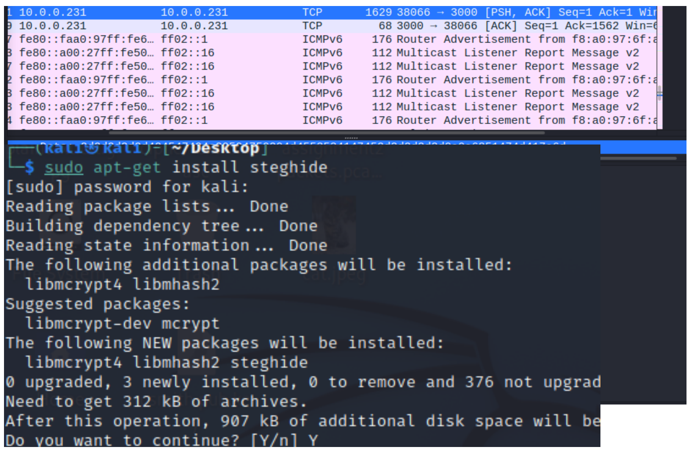


```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat plaintext.txt.asc.sig.asc | netcat 10.0.0.231 3000
```



```
┌──(kali㉿kali)-[~]
└─$ nc -l -p 3000
```

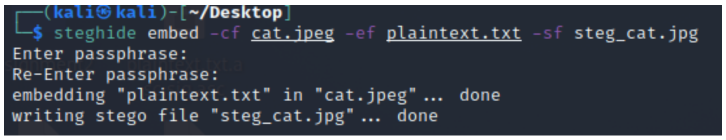
Now go to the wireshark window and locate the packet containing [PSH, ACK] in the info column. The packet should contain information that looks like the initial ascii encrypted text file.



```
1 10.0.0.231             10.0.0.231            TCP       1629 38066 → 3000 [PSH, ACK] Seq=1 Ack=1 Wir
9 10.0.0.231             10.0.0.231            TCP         68 3000 → 38066 [ACK] Seq=1 Ack=1562 Win=6
7 fe80::faa0:97ff:fe6… ff02::1              ICMPv6     176 Router Advertisement from f8:a0:97:6f:a
3 fe80::a00:27ff:fe50… ff02::16             ICMPv6     112 Multicast Listener Report Message v2
7 fe80::a00:27ff:fe50… ff02::16             ICMPv6     112 Multicast Listener Report Message v2
2 fe80::faa0:97ff:fe6… ff02::1              ICMPv6     176 Router Advertisement from f8:a0:97:6f:a
3 fe80::a00:27ff:fe50… ff02::16             ICMPv6     112 Multicast Listener Report Message v2
3 fe80::a00:27ff:fe50… ff02::16             ICMPv6     112 Multicast Listener Report Message v2
4 fe80::faa0:97ff:fe6… ff02::1              ICMPv6     176 Router Advertisement from f8:a0:97:6f:a
```

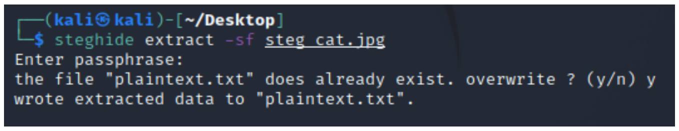
```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo apt-get install steghide
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libmcrypt4 libmhash2
Suggested packages:
  libmcrypt-dev mcrypt
The following NEW packages will be installed:
  libmcrypt4 libmhash2 steghide
0 upgraded, 3 newly installed, 0 to remove and 376 not upgrad
Need to get 312 kB of archives.
After this operation, 907 kB of additional disk space will be
Do you want to continue? [Y/n] Y
```

We will now encrypt the plaintext.txt file into an image of our choosing. But first, we will need to install steghide. To do this enter the following command: sudo apt-get install steghide. Then, ensure the image we would like to use is saved to the Kali Linux desktop for easy access. Now enter the command steghide embed -cf image.jpeg -ef plaintext.txt -sf steg_image.jpg into the terminal. we will be prompted to enter a password of the choosing twice. We will use: letmein.

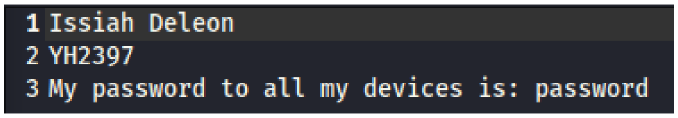


```
┌──(kali㉿kali)-[~/Desktop]
└─$ steghide embed -cf cat.jpeg -ef plaintext.txt -sf steg_cat.jpg
Enter passphrase:
Re-Enter passphrase:
embedding "plaintext.txt" in "cat.jpeg"... done
writing stego file "steg_cat.jpg"... done
```

If we wanted to extract the encrypted information from the image, we could enter this command along with the password when prompted: steghide extract -sf filename.jpg

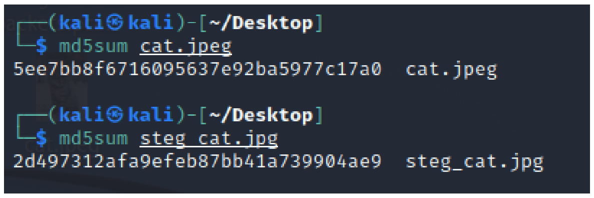


```
┌──(kali㉿kali)-[~/Desktop]
└─$ steghide extract -sf steg_cat.jpg
Enter passphrase:
the file "plaintext.txt" does already exist. overwrite ? (y/n) y
wrote extracted data to "plaintext.txt".
```



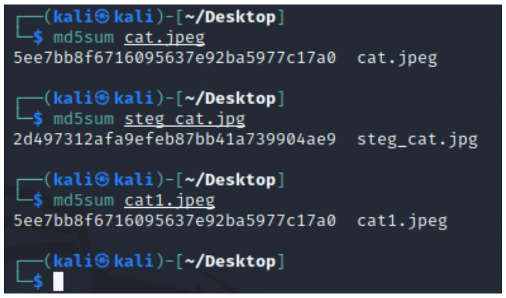
```
Issiah Deleon
YH2397
My password to all my devices is: password
```

Now we can check the integrity of our original image, and the image with the embedded data. To do so, enter: md5sum imagename.jpg and md5sum embeddedimagename.jpg. We will notice that



```
┌──(kali㉿kali)-[~/Desktop]
└─$ md5sum cat.jpeg
5ee7bb8f6716095637e92ba5977c17a0  cat.jpeg

┌──(kali㉿kali)-[~/Desktop]
└─$ md5sum steg_cat.jpg
2d497312afa9efeb87bb41a739904ae9  steg_cat.jpg
```

the checksums for each respective image are different, indicating that the image files are not *exactly* the same. There is something different about the files, and this can range from a single pixel that has been manipulated, to secret data having been encrypted within the file that now makes it a different size than the original image. Checksums do not indicate *what* has changed but rather that something *has* changed about the image. We can change the name of the original file to observe whether we get a different checksum or not. As shown below, we see that simply changing the name of the file does not generate a different checksum, so the *image* has not changed, just its name.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ md5sum cat.jpeg
5ee7bb8f6716095637e92ba5977c17a0  cat.jpeg

┌──(kali㉿kali)-[~/Desktop]
└─$ md5sum steg_cat.jpg
2d497312afa9efeb87bb41a739904ae9  steg_cat.jpg

┌──(kali㉿kali)-[~/Desktop]
└─$ md5sum cat1.jpeg
5ee7bb8f6716095637e92ba5977c17a0  cat1.jpeg

┌──(kali㉿kali)-[~/Desktop]
└─$ █
```

Conclusion

In conclusion, we have demonstrated the utilization of symmetric and asymmetric encryption tools. These tools, gpg, md5sum, and steghide are valuable additions to the retinue of encryption technology, allowing us the ability to encrypt our data symmetrically with one key, or asymmetrically with two or more keys. The use of netcat allowed us to visualize exactly how sending an encrypted file may differ from sending unencrypted data, as we demonstrated in the last assignment. This differed because although we were able to intercept and capture the “secret” message, it was fully encrypted in ascii text and was otherwise unreadable to the initial viewer.

GPG is a tool that allows the option of symmetrically or asymmetrically encrypting the data. GPG and steghide both provide authentication by requiring a password (key) for the encryption of the data, which adds some layer of protection against sending/receiving illegitimate data. GPG and steghide also both offer access control by encrypting the data so that only those who we want to see the data will see it, and in order to decrypt and view the data we require a key. Data confidentiality is tied in with the others; by encrypting the data and requiring a key to decrypt it, we are providing confidentiality to sensitive data. Data integrity is not as available in steghide or GPG, but that is where md5sum comes in. md5sum ensures the integrity of data by verifying whether or not the file has changed. We can use this tool to compare checksums to verify whether the file we are working with may have been manipulated since the last time we used it. md5sum provides non-repudiation because md5sum will allow us to verify that a file has changed, or has not changed in the event that someone tries to deny the changing of the file. GPG also provides key fingerprints, which can help prove the existence of an action or encryption operation.