

Khushveen Kaur Umra (947005108)

CprE 308 Section 2

6th December 2022


Lab 8 – Report

For this lab, I learned that logistics of using SSH (Secure Shell) and gpg (GNU privacy guard). The following exercises were executed:

3.1.2: Secure File Transfer

-> For this exercise, I utilized PuTTY in order to copy my files from the remote machine (linux-1.ece.iastate.edu) to my local machine (linux-2.ece.iastate.edu) and vice versa. Initially, I found it extremely difficult to copy any of the files as it either kept printing the usage information, or it gave me errors stating that there is no such directory or file. After trying multiple different methods, and utilizing the help given by a TA on discord, I was able to figure out the error, and was able to transfer the files back and forth.

First, I transferred a .txt file from my local machine to my remote machine, using the scp command. The screenshot is as follows:



```
-bash-4.2$ hostname
linux-2.ece.iastate.edu
-bash-4.2$ pwd
/home/kumra
-bash-4.2$ ls
308 bash.rc Desktop Documents Downloads Music Pictures Public Templates tsclient Videos
-bash-4.2$ scp /home/kumra/Desktop/File.txt kumra@linux-1.ece.iastate.edu:/home/kumra/Documents/File.txt
File.txt 100% 56 40.8KB/s 00:00
-bash-4.2$ ls
308 bash.rc Desktop Documents Downloads Music Pictures Public Templates tsclient Videos
-bash-4.2$ cd Documents
-bash-4.2$ ls
File.txt
```

The transfer of the file was completed successfully, and in the screenshot, you will be able to see the successful transfer. Then, I transferred a .txt file from my remote machine to my local machine, utilizing the rsync -av command.

First, I got myself familiar with the difference between the two commands, to understand how they work independently. Based on my understanding, “scp” command is utilized to copy files and directories in a secure way, where it reads the source file and writes it to the destination, or in other words, a plain linear copy. On the other hand, “rsync -av” command is utilized for moving and synchronizing files. Here, we can manage files/directories effectively while backing up data on a regular basis. It also compares if there is any difference between the files that been transferred. The following screenshot shows the successful transfer of files.

```
-bash-4.2$ rsync -av /home/kumra/Documents/File2.txt kumra@linux-2.ece.iastate.edu:/home/kumra/Desktop/File2.txt
sending incremental file list
File2.txt

sent 213 bytes  received 35 bytes  496.00 bytes/sec
total size is 97  speedup is 0.39
```

```
-bash-4.2$ rsync -av /home/kumra/Documents/File2.txt kumra@linux-2.ece.iastate.edu:/home/kumra/Desktop/File2.txt
sending incremental file list

sent 69 bytes  received 12 bytes  162.00 bytes/sec
total size is 97  speedup is 1.20
-bash-4.2$ █
```

For the initial transfer, the following bytes were copied: sent 213 bytes received 35 bytes 496.00 bytes/sec

For the second transfer, the following bytes were copied: sent 69 bytes received 12 bytes 162.00 bytes/sec

Here, we can see that was a difference in the number of bytes that were copied, which occurred because we copied the same file again, without making any changes. Rsync uses a delta-transfer algorithm, which compares files from source and destination and sends only the differences between them. This is why, every time you copy the same file over and over again, utilizing the “rsync” command, lesser the number of bytes will be copied. This is what I learned from this exercise for this lab.

3.1.3 SSH escape sequences

This exercise was a little confusing for me, as I wasn't really sure what needed to be done. I followed the steps and logged into my remote machine, where I typed in the command to see all the supported escape sequences. And then I typed ~ CTRL+Z to suspend the connection. The escape sequences can allow one to assume the remote server has ssh active, and if one is able to ssh into the machine, then it will likely have scp active as well. I then used scp to fetch the File.txt on the Desktop of the remote machine and verified that I was in the remote machine. I learned that this allows you to copy and fetch files without have to log into the remote machine over and over again every time I was to copy a file from the remote machine. The following screenshot proves the above exercise.

```
-bash-4.2$ `?
> ~?
> ~?
Supported escape sequences:
~.    - terminate connection (and any multiplexed sessions)
~B    - send a BREAK to the remote system
~C    - open a command line
~R    - request rekey
~V/v  - decrease/increase verbosity (LogLevel)
~^Z   - suspend ssh
~#    - list forwarded connections
~&    - background ssh (when waiting for connections to terminate)
~?    - this message
~~    - send the escape character by typing it twice
(Note that escapes are only recognized immediately after newline.)
~^Z [suspend ssh]

[1]+  Stopped                  ssh kumra@linux-1.ece.iastate.edu
```

3.1.4 Known host

This was an interesting portion of the lab, as I learned how we can use the command “cat” to print and edit the contents of a file, which has the ability to warn the user about any changes that were made to their files. I learned how every SSH server has a public key that identifies to its client, and how changing the contents can further alert the user by warning them that it does not recognize the server's key.

3.1.5 Cryptographic Keys

This was really an interesting exercise, as it is also something you do in GitHub.

- 1) Under the .ssh directory, the private key is saved in “id_dsa”, which is always done by default. The public key is saved in “id_dsa.pub”, which is also always done by default.

```
// Your identification has been saved in /home/kumra/.ssh/id_dsa.
```

```
Your public key has been saved in /home/kumra/.ssh/id_dsa.pub.
```

- 2) When you create a key pair, you are asked to enter a passphrase. The difference between this passphrase and a password is that passphrase is generally referred to a secret used to protect an encryption key, where as a password is generally referred to something used to authenticate or log into a system. The passphrases protect your private key from being used by someone who doesn't know the passphrase. Hence, passphrases increase the security when using SSH keys. If you forget your passphrase, or you want to change the passphrase, you can use the command “ssh-keygen -p”, which will then ask you to confirm the location of the SSH key file, the old passphrase, and ask twice for the new passphrase. Using a password-encrypted public key is a little riskier, as there is no way to recover it if you forget it.
- 3) The private file/key is the one the passphrase is used to encrypt. The public file/keys do not have, or do not need passphrases. The public file is a file containing a single line, the protocol, the key, and an email used as an identifier, which is not something that can be used to gain access to any files. This is why only the private key is encrypted.

```
Your identification has been saved in /home/kumra/.ssh/id_dsa.
Your public key has been saved in /home/kumra/.ssh/id_dsa.pub.
The key fingerprint is:
SHA256:fL8/5mls+6j5W8sg/NIj1Cixm45SeLRUX8YZ8+V6sJI kumra@linux-2.ece.iastate.edu
The key's randomart image is:
+---[DSA 1024]-----+
|      .oo . |
|      =o o |
|      . . o o . |
|      o o . . + |
|      + . S oE.o . |
|      . + o o . + . |
|      o  + .oo. . |
|      . .o oo==+ |
|      .... *BBO= |
+---[SHA256]-----+
```

3.1.6 Key-based authentication

For this exercise, we followed the steps to allow to SSH into our account on the remote system from the local machine that has our private key. By doing so, we changed the authentication of from password protection to passphrase encrypted protection, which means that if your private key is password protected, the remote machine will prompt you for the passphrase, which is more secure than using passwords. This is the advantage of the method. From all these exercises, I learned about all the logistics of SSH keys, and how helpful they can be to protect your private files. Initially, I had no idea that you can change the authentication for your account on a remote machine to make it more secure. I thought having a password protected account was the only way to protect your account. Overall, all these exercises helped me to learn a lot about SSH keys, and how one can utilize it to protect their files from any cyber-attacks.

3.2 GnuPg

Executing the following exercises for this portion, I learned about GNU Privacy Guard, which is widely used to encrypt, decrypt, sign and check messages on the internet.

3.2.1 Generate a key pair

```
--
-bash-4.2$ gpg --gen-key
gpg (GnuPG) 2.0.22; Copyright (C) 2013 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.

gpg: directory `/home/kumra/.gnupg' created
gpg: new configuration file `/home/kumra/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/kumra/.gnupg/gpg.conf' are not yet active during
this run
gpg: keyring `/home/kumra/.gnupg/secring.gpg' created
gpg: keyring `/home/kumra/.gnupg/pubring.gpg' created
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)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 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) 10y
Key expires at Fri 03 Dec 2032 10:07:51 PM CST
Is this correct? (y/N) y

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

Real name: khushku695
Email address: kumra@iastate.edu
Comment: final lab
You selected this USER-ID:
    "khushku695 (final lab) <kumra@iastate.edu>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

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: /home/kumra/.gnupg/trustdb.gpg: trustdb created
gpg: key 8E2DB14B marked as ultimately trusted
public and secret key created and signed.

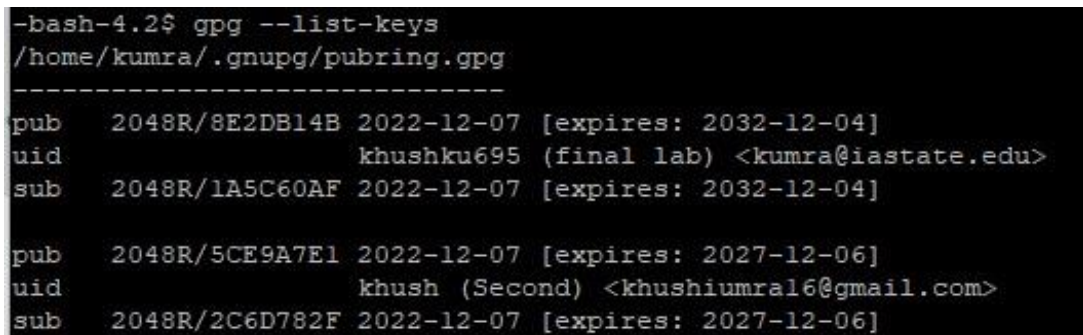
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2032-12-04
pub   2048R/8E2DB14B 2022-12-07 [expires: 2032-12-04]
       Key fingerprint = 2CF5 57FE F856 07AF D3F1 F217 02A2 0ADA 8E2D B14B
uid           khushku695 (final lab) <kumra@iastate.edu>
sub   2048R/1A5C60AF 2022-12-07 [expires: 2032-12-04]
```

This was such an interesting exercise to work on. The concept of GnuPG was completely new to me as I had never heard about it before or used it before. I had only worked with SSH keys and having the opportunity to learn about GnuPG was interesting. I learned that GnuPG is a tool for secure communication, as it used public-key cryptography, so that users may communicate securely. I learned that in a public-key system, each user has a pair of keys consisting of a private key and a public key. The difference is that GnuPG uses a more sophisticated scheme in which a user has a primary keypair and then zero or more additional subordinate keypairs.

3.2.2 Exchanging Keys

Listing keys:

Here I learned that by using the command “gpg --list-keys”, you can list all the keys on your public key ring, which will display all the gpg keys with its respective name, email address, comment, and the expiry date of the gpg key. The following screenshot showcases the above information:



```
-bash-4.2$ gpg --list-keys
/home/kumra/.gnupg/pubring.gpg
-----
pub   2048R/8E2DB14B 2022-12-07 [expires: 2032-12-04]
uid           khushku695 (final lab) <kumra@iastate.edu>
sub   2048R/1A5C60AF 2022-12-07 [expires: 2032-12-04]

pub   2048R/5CE9A7E1 2022-12-07 [expires: 2027-12-06]
uid           khush (Second) <khushiumral6@gmail.com>
sub   2048R/2C6D782F 2022-12-07 [expires: 2027-12-06]
```

I learned that the “2048R/8E2DB14B” portion is the format that is being used by old versions of GPG. It uses a single letter R for RSA, because in the older versions, there were only three types of keys to distinguish, and it uses the short key_id of 8 hex digits.

Exporting a public key:

This is something that was confusing at first, as I really wasn’t sure what this did. But after doing some research, and executing the command, “gpg --output <file> --export <identity>”. I learned that this allows you to send your key to correspondents or to a keyserver by exporting the key. Here, you cannot see any output, because not only did I export my public key, I redirected the output to another user, in a binary format. I also learned that adding --armor can cause the output

to be generated in an ASCII-armored format similar to unencoded documents. Both the above information is seen in the screenshot below.

```
-bash-4.2$ gpg --output kumra.gpg --export khushiumral6@gmail.com
File 'kumra.gpg' exists. Overwrite? (y/N) y
-bash-4.2$ gpg --armor --export khushiumral6@gmail.com
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQENBGOQG6gBCACyMw0WuFqzdcsLPE72CyEUsr36TbHBzt28laY6KPLMKUpi5RgH
RemwyH75u8vrzyGcMe5hl7AdvkIMB0d/u9bFbFNmTGe0yEFdH9/EmslxwlFi0OeH
cht7pE6gsSPgwk2ZCwLPcoEbDzlawffF4vJ8+IhjcP+O58O9oExfVIvZD8dpHW3A
zul7WDBFlAI8ET5Am7lzZBw697BuE3dlXqUjI/gNU0r3cHrFf8yuJLvY+WuP/NS9
9EFkqEqtnLEpWVZiHCWqGTNKcvlZs//HKXMASXXooYOkYUjSIOJnQL7xli4FACaX
P+RwCO5rALw46ye3KcusunL7QET+wW4j4smPABEBAAG0J2todXNoIChTZWNvbWQp
IDxraHVzaGllbXJhMTZA2Z2lhaWwuyY29tPokBPwQTAQIAKQUCY5AbqAIbAwUJCWYB
gAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEPQde/lc6afh+2sH/RLCjAfc
zU+fI82AB0aavULlAwHiuh48hE5yrSwzhKlz+KOi6+YneI3SCigwu4dpygflogN5
l/T9ON6SlUkFyTuWRN7WkIgiw/647BSDQ7wflkgCMgaFlzsGKoe0ybiC5c94vunY
eGoOFIAamoXDZXVZK+eXwzJKmYObUSGbATVz5bMNIqg0HuvrK0g2UxTWWMB+BHGJ
E/B/+jK60hhRRBM0lriuUdTW69uPG6lV4kOHcu8D3aNBwfcTD/MJUMzbDZcoY/g4
k/D9NRvQG7LQAUvd2UOF438bvO4Au34Vqh65dFW/FI0nxYAX8CR1lhPg7uisjH14
Q7g3yHd6WGi0ZIC5AQ0EY5AbqAEIALWvMq2xjth+fOWRE5J46GP4nSuRlc07WRP4
194sHNhlJcdlQE0lsUumhGaPnnI0lJ3b9Dwrql7C/K/J34i5zSrKvRoQz1kdjb8B
OFJmrGbSlBC9+KTHCSxjlFoqltEmR6onSlh97/Wc2j8Qzw9mmXFZlW7gJrc+JzD2
hJxiRcf6bMYvP6vOqKsbQc+XxJKZpO5JJ4YwnmGK7ChBy9/DJhqvvv+dDclTv2DK
7l7T8tyF0cqRyfd+V7qQQ28Ob9V6m/IM9M8xBBP2NMBW+LGjAsgXyLJ+K4rmXxJ4
0eATG/kRDjaxItccC77t9t4zv4XUdTbjqhsDjegcmdy2IVluz/sAEQEAAAYkBJQQY
AQIADwUCY5AbqAIbDAUJCWYBgAAKCRD0HXv5XOmn4YTtB/wKBiObLQEff4TagIzl
IMIOL6fdkBU0rAN8aAwJVJVk5WQ7Se+FCQaxjaBONLHkDIFqArKI4LSKcNdclMXo
Zya74lxdZ7J9kOdU6i7hs4qvKAsxz9Pcb08wrlY6wv10blaXQ9+TlOJwdquqPWfQ
oEfLWlTYSfAMXyM9RHDlm9L+vCtIFGZ2w/HXRRW5FqOfvEnz5HDsaCjg2ks/YldU
Qr42X8eYiVXJDFcad45NMJ4pEkVfKzCyv7Or9EH7Q/kZnAMF75tcW25zmqAPqN3W
tki4QU/mbSAU7VljJ4w+gw7oz90G18oW0ai0wphPmbkdwiTR360LSXuUYalmBm02
mAEA
=QD3I
-----END PGP PUBLIC KEY BLOCK-----
-bash-4.2$ █
```


Importing a public key:

Here, I learned how I can import someone else's public key to my public keyring with the `--import` command. I also learned that once a key is imported, it should be validated. GnuPG uses a powerful and flexible trust model that does not require us to personally validate each key that we import, whereas, some key may need to be personally validated.

```
-bash-4.2$ gpg --import kumra.gpg
gpg: key 5CE9A7E1: "khush (Second) <khushiumral6@gmail.com>" not changed
gpg: Total number processed: 1
gpg:             unchanged: 1
```

3.2.3 Encrypting and decrypting documents

I loved this part of the lab, as it was something I had never done before. All the way from creating gpg keys, to importing keys, and encrypting/decrypting files was extremely interesting to me. I learned how you can select to encrypt a file and send it to a correspondent, and how they would need to decrypt the file, in order to read the message.

```
-bash-4.2$ gpg --local-user khushku695 --recipient khush --output output.txt --e
ncrypt output.txt
File 'output.txt' exists. Overwrite? (y/N) y
-bash-4.2$ gpg --decrypt output.txt

You need a passphrase to unlock the secret key for
user: "khushku695 (final lab) <kumra@iastate.edu>"
2048-bit RSA key, ID 1A5C60AF, created 2022-12-07 (main key ID 8E2DB14B)

gpg: Invalid passphrase; please try again ...

You need a passphrase to unlock the secret key for
user: "khushku695 (final lab) <kumra@iastate.edu>"
2048-bit RSA key, ID 1A5C60AF, created 2022-12-07 (main key ID 8E2DB14B)

gpg: Invalid passphrase; please try again ...

You need a passphrase to unlock the secret key for
user: "khushku695 (final lab) <kumra@iastate.edu>"
2048-bit RSA key, ID 1A5C60AF, created 2022-12-07 (main key ID 8E2DB14B)

gpg: encrypted with 2048-bit RSA key, ID 1A5C60AF, created 2022-12-07
      "khushku695 (final lab) <kumra@iastate.edu>"
```

3.2.4 Making and verifying signatures

This was such an interesting and old fashioned way of authenticating documents, and verifying the signature on the document. I learned that the digital signature certifies and timestamps a document. If the document is subsequently modified in any way possible, a verification of the signature will fail eventually. I learned that this digital signature could serve the same purpose as a hand-written signature with the additional benefit of being tamper-resistant.

I learned that creating and verifying signatures uses the public/private keypair in an operation different from encryption and decryption. A signature is created by using the private key of the signer, whereas the signature is verified using the corresponding public key.

```
-bash-4.2$ gpg --output output.txt --sign output.txt

You need a passphrase to unlock the secret key for
user: "khushku695 (final lab) <kumra@iastate.edu>"
2048-bit RSA key, ID 8E2DB14B, created 2022-12-07

gpg: Invalid passphrase; please try again ...

You need a passphrase to unlock the secret key for
user: "khushku695 (final lab) <kumra@iastate.edu>"
2048-bit RSA key, ID 8E2DB14B, created 2022-12-07

gpg: Invalid passphrase; please try again ...

You need a passphrase to unlock the secret key for
user: "khushku695 (final lab) <kumra@iastate.edu>"
2048-bit RSA key, ID 8E2DB14B, created 2022-12-07

File 'output.txt' exists. Overwrite? (y/N) y
-bash-4.2$ gpg--verify output.txt
bash: gpg--verify: command not found...
-bash-4.2$ gpg --verify output.txt
gpg: Signature made Tue 06 Dec 2022 11:46:01 PM CST using RSA key ID 8E2DB14B
gpg: Good signature from "khushku695 (final lab) <kumra@iastate.edu>"
-bash-4.2$ █
```

I learned that when you digitally sign a document, it compresses the document, and output is in binary format. For a signed document, you can either check the signature, or check the signature and recover the original document. This was such an interesting thing to execute, as this is not something that I had ever done before, and I am grateful that I got the opportunity to do something extremely new like this.

