ROLL NO: 42

NAME: PIYUSH HINGORANI

DATE: 13/09/23

ASSIGNMENT NO – 12

AIM: Explore the GPG tool of linux to implement email security

LAB OUTCOME: LO6: Demonstrate the network security system using open source tools.

THEORY:

1. **What is private key ring and public key ring?**

⇨ A "private key ring" and a "public key ring" are terms often associated with the management of cryptographic keys, particularly in the context of encryption and secure communication. These terms are not commonly used in all cryptographic systems, but they can be related to the concept of key pairs, where you have a private key and a corresponding public key.

Private Key Ring:

A private key ring is a collection or set of private keys that are used for various cryptographic operations, such as decrypting data or signing messages.Each private key is typically associated with a specific user, device, or entity and should be kept secret and well-protected.

Private keys are used for decryption (in the case of asymmetric encryption), signing documents, and proving identity in secure communication.

Example: Imagine you have a private key ring for your email communication. It contains a private key for your email address (e.g., alice@example.com).

You use this private key to decrypt emails sent to you and to sign outgoing messages to prove they are indeed from you.

Public Key Ring:

A public key ring is a collection or set of public keys that are paired with the private keys in the private key ring.Public keys can be freely shared with anyone and are used for encryption (in the case of asymmetric encryption), verifying digital signatures, and ensuring the confidentiality and integrity of data.

Example: Continuing with the email scenario, your email contacts might have copies of your public key in their public key rings. They use your public key to encrypt emails they send to you, ensuring that only you can decrypt and read them with your corresponding private key.

Hence, private key rings contain the private keys used for decryption and signing, while public key rings contain the public keys used for encryption and verification. These key pairs play a crucial role in secure communication, data protection, and digital signatures in various cryptographic applications.

2. **Write the commands used for key generation, export and import of keys and signing and encrypting the message in gpg tool.**

⇨ GPGis the OpenPGP part of the GNU Privacy Guard (GnuPG). It is a tool to provide digital encryption and signing services using the OpenPGP standard. gpg features complete key management and all the bells and whistles you would expect from a full OpenPGP implementation.

To demonstrate key generation, exporting, importing keys, signing, and encrypting messages using the GPG (GNU Privacy Guard) tool. GPG is commonly used for secure email communication and file encryption.

Install GPG:

If GPG is not already installed on your Linux system, you can typically install it using your package manager. For example, on Debian/Ubuntu, you can run:

sudo apt-get install gnupg

Key Generation:

Generate private key and public key pairs for sender and receiverusing commandgpg:

gpg --gen-key or gpg –full-generate-key.

Exporting and Importing Keys:

Create a file containing sender's public key which then can be sent to other users.

gpg --export -a username>filename(creates file in ascii format) orgpg --output filename --armor --export user's_email(for sender)

Similarly create file containing sender's private key.

gpg --export-secret-key -a username>filename(for sender)

You can create a fingerprint of key using the command

gpg --fingerprint receiver's_email(for receiver)

Sender needs to add in his public key ring, the public key of receiver(for sender)

gpg --import filename_containing_public_key_of_receiver

Listing public keys in keyring:

gpg --list-keys(from public key rings of all users)

gpg --list-keys Piyush@yahoo.com

Signing a Message:

You can also digitally sign your emails to prove their authenticity. Most email clients with GPG support will have an option to sign messages. When you sign a message, your private key will be used to create a digital signature, which recipients can verify using your public key.

To sign a message with your private key:

gpg --sign your_message.txt

This will create a signed version of your message, typically with a .asc extension.

Encrypting a Message:

When composing an email, use your email client's GPG integration to encrypt the message. Typically, you'll find an option to encrypt the message for specific recipients. The email client will use the recipients' public keys to encrypt the message.

To encrypt a message for someone else using their public key:

gpg --encrypt -r receiver_email name_of_file

OR

gpg --encrypt --sign --armor -r receiver_email name_of_file

(encrypt and sign, ascii file created)

OR

gpg --encrypt --sign -r receiver_email name_of_file

This will create an encrypted version of your message, which only the recipient can decrypt using their private key.

Decrypting a Message:

When you receive an encrypted email, your email client (configured with GPG) should automatically decrypt it using your private key if it's addressed to you.

To verify the authenticity of a signed email, your email client will use the sender's public key to verify the digital signature. If the signature is valid, it means the email was not tampered with during transit.

To decrypt a message you received:

gpg -o myfiledecrypted -d myfile.txt.gpg

You'll be prompted to enter your passphrase to unlock your private key for decryption.

These are the basic commands for key management, signing, and encrypting/decrypting messages using GPG. Remember to replace "Your Name" and "Recipient Name" with the actual names or email addresses associated with the keys you're working with. Additionally, always exercise caution with private keys and keep them secure.

3. **Paste appropriate screenshots showing output of all commands**

⇨ Step 1: Generate private key and public key pairs for sender and receive rusing command *gpg --gen-key or gpg –full-generate-key (repeat for sender and receiver)* **Similarly create for receiver.**

Step 2: Create a file containing sender's public key which then can be sent to other users.

gpg --export -a username>filename (creates file in ascii format) or
gpg --output filename --armor --export user's_email (for sender)

```
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.

gpg: directory '/root/.gnupg' created
gpg: keybox '/root/.gnupg/pubring.kbx' created
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: sender
Email address: sender@gmail.com
You selected this USER-ID:
    "sender <sender@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: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key B5024F6948FF1057 marked as ultimately trusted
gpg: directory '/root/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/root/.gnupg/openpgp-revocs.d/63C1EAB750B87BE2B37C7FDAB5024F6948FF1057.rev'
public and secret key created and signed.

pub   rsa3072 2023-10-17 [SC] [expires: 2025-10-16]
      63C1EAB750B87BE2B37C7FDAB5024F6948FF1057
uid                      sender <sender@gmail.com>
sub   rsa3072 2023-10-17 [E] [expires: 2025-10-16]
```

```
      065243D15C41BD7002C6937348E2B446CE691BAF
uid                      receiver <receiver@gmail.com>
sub   rsa3072 2023-10-17 [E] [expires: 2025-10-16]

root@ Arnav :/home/gpg# gpg --export -a sender>s_public
root@ Arnav :/home/gpg# gpg --export -a receiver>r_public
root@ Arnav :/home/gpg#
```

Step 3: Similarly create file containing sender's private key.

gpg --export-secret-key -a username>filename (for sender)

```
uid                      receiver <receiver@gmail.com>
sub   rsa3072 2023-10-17 [E] [expires: 2025-10-16]

root@ Arnav :/home/gpg# gpg --export -a sender>s_public
root@ Arnav :/home/gpg# gpg --export -a receiver>r_public
root@ Arnav :/home/gpg# gpg --export-secret-key -a sender>s_private
root@ Arnav :/home/gpg#
```

Step 4: You can create a fingerprint of key using the command *gpg --fingerprint receiver's_email (for receiver)*

```
root@ Arnav :/home/gpg# gpg --fingerprint receiver@gmail.com
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: next trustdb check due at 2025-10-16
pub   rsa3072 2023-10-17 [SC] [expires: 2025-10-16]
      0652 43D1 5C41 BD70 02C6  9373 48E2 B446 CE69 1BAF
uid           [ultimate] receiver <receiver@gmail.com>
sub   rsa3072 2023-10-17 [E] [expires: 2025-10-16]
```

Step 5: Sender needs to add in his public key ring, the public key of receiver (for sender)

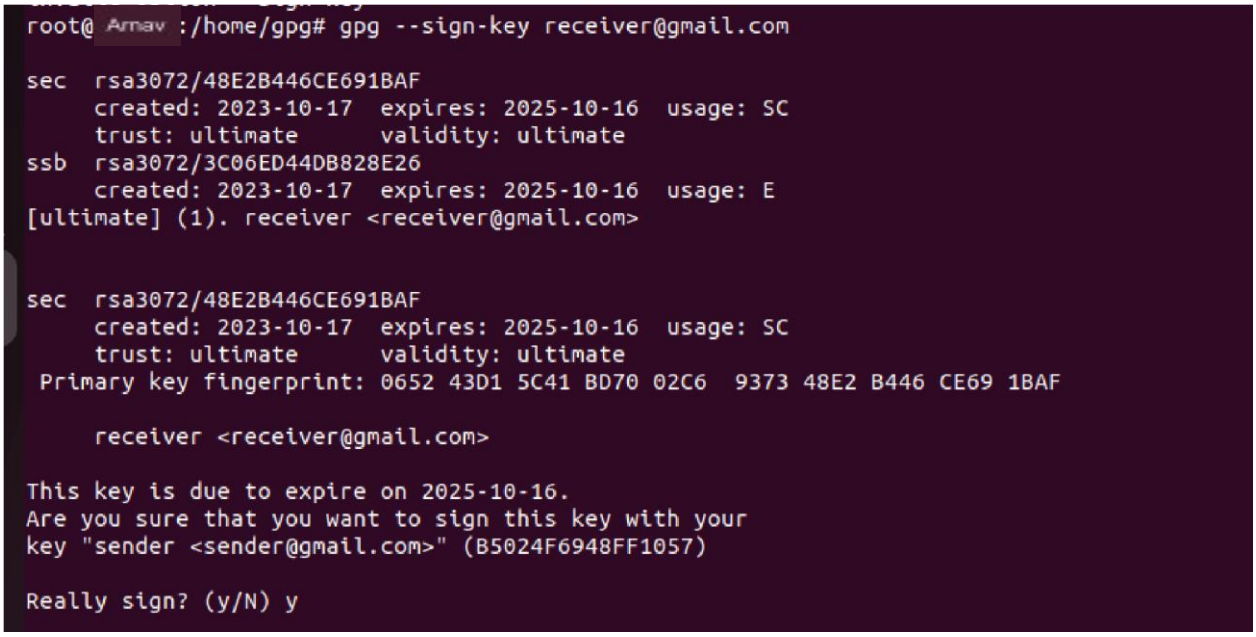
gpg --import filename_containing_public_key_of_receiver

```
root@ Arnav :/home/gpg# gpg --import r_public
gpg: key 48E2B446CE691BAF: "receiver <receiver@gmail.com>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
```

Step 6: Listing public keys in keyring *gpg --list-keys (from public key rings of all users) gpg --list-keys shachi_natu@yahoo.com (from public key rings of specific users)*

How do you know that the person giving you the public key is who they say they are?

Step 7: Sender can sign the public key of receiver using command *gpg --sign-key receiver_email*



When you sign the key, it means you verify that you trust the person is who they claim to be. This can help other people decide whether to trust that person too. If

someone trusts you, and they see that you've signed this person's key, they may be more likely to trust their identity too.

Step 8: Encrypt the data to send. (create a file beforehand to be encrypted) *gpg --encrypt -r receiver_email name_of_file (only encrypt, .gpg file created)*

OR

gpg --encrypt --sign --armor -r receiver_email name_of_file (encrypt and sign, ascii file created)

OR

gpg --encrypt --sign -r receiver_email name_of_file (encrypt and sign, .gpg file created)

```
root@ Arnav :/home/gpg# cat > test_file.txt
hi this is a test file.
root@ Arnav :/home/gpg#
```

A file is created named (test_file.txt)

```
root@ Arnav :/home/gpg# gpg --encrypt -r receiver@gmail.com test_file.txt
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: next trustdb check due at 2025-10-16
root@ Arnav :/home/gpg# ls
r_public  s_private  s_public  test_file.txt  test_file.txt.gpg
root@ Arnav :/home/gpg#
```

Step 9: Decrypt the file

gpg -o myfiledecrypted -d myfile.txt.gpg

CONCLUSION: Successfully, we learnt use of GPG on Linux to enhance the security of your email communication, ensuring that your messages are encrypted and digitally signed for privacy and authenticity.