BIRMINGHAM CITY
University

Author: Sujal Manandhar

Course: Cyber Security

Code: CMP5329

Report: Coursework Logbook

Date: 2024/06/05

Student ID: 23189654

Page Count: 48

Word Count: 5369

TABLE OF CONTENTS

TABLE OF FIGURES

Introduction

Course Overview

The CMP5329 Cyber Security course is designed to provide students with a practical understanding of various cybersecurity principles and techniques. The labs associated with this course are structured to enhance theoretical knowledge with hands-on experience, focusing on the implementation and management of security protocols, cryptographic methods, and cyber defence strategies.

Lab Objectives

The primary objective of these labs is to enable students to:

1. Understand and Implement Security Protocols: Students will learn about and implement various security protocols that protect systems from unauthorised access and attacks.
2. Explore Cryptographic Techniques: Labs focus on the practical application of cryptography, including the use of tools like GPG for encryption and decryption, to ensure the confidentiality and integrity of information.
3. Develop Cyber Defense Skills: Students will engage in activities such as password cracking and the secure configuration of systems to recognize vulnerabilities and learn how to defend against them.
4. Enhance System Administration Capabilities: Through exercises involving user management, privilege control, and the secure handling of user data, students will gain skills in secure system administration.

Labs Overview

Each lab is designed to tackle different aspects of cybersecurity:

1. Lab 1: Introduction to Linux commands and basic security configurations.
2. Lab 2: Simple use of GPG for secure communication.
3. Lab 5: Use of Linux Discretionary Access Control commands.
4. Lab 6: Password security and cracking techniques.

Teaching Methodology

The labs are structured to provide a progressive learning curve, where each session builds upon the previous one. Each lab session includes a series of tasks that guide the students through the necessary steps to achieve the lab objectives. These tasks are complemented by reflective exercises that encourage students to think critically about the security measures they implement and their implications in real-world scenarios.

Openssl (Lab 1)

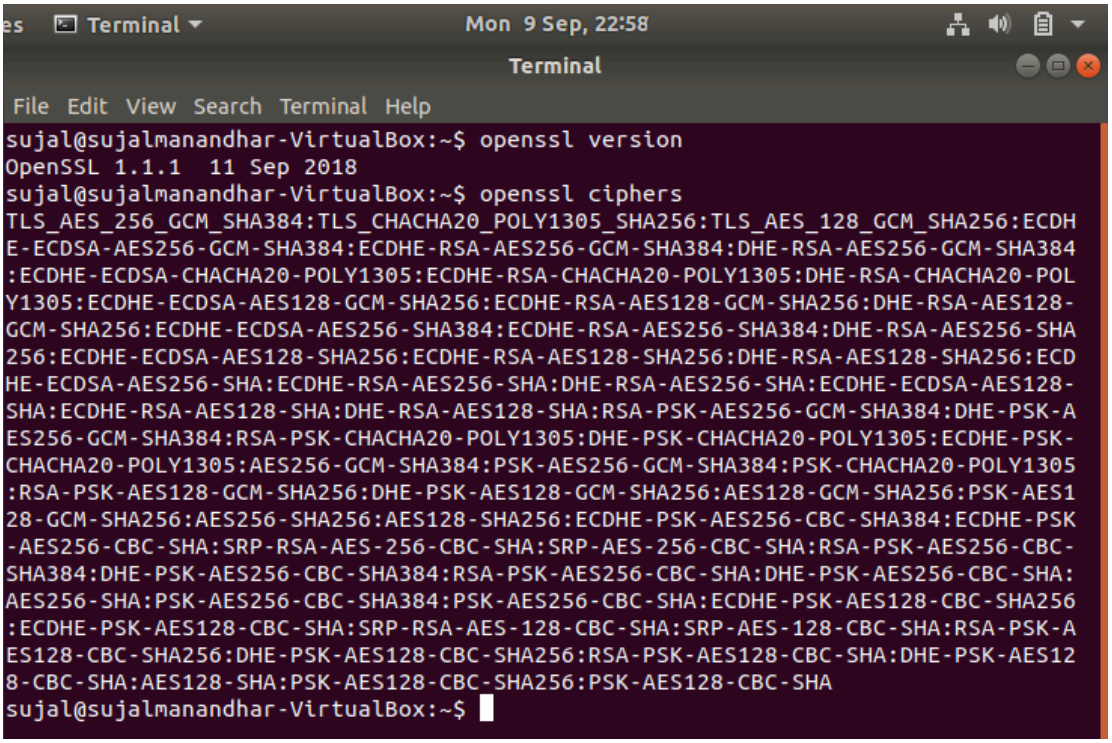
Introduction
In this lab, we explored the OpenSSL toolkit, which is widely used for securing communication over networks. We generated public and private key pairs, encrypted and decrypted files, and created digital signatures to ensure the confidentiality, integrity, and authenticity of data. This lab demonstrated how cryptographic functions in OpenSSL help secure sensitive information in transit.

OpenSSL Version and Supported Ciphers

Introduction: Understanding the OpenSSL version in use is critical, as newer versions support enhanced cryptographic protocols that safeguard against contemporary vulnerabilities.

Execution and Explanation:

- **Command**: “openssl version”
    1. **Purpose**: Verifies the installation and displays the version of OpenSSL, crucial for determining the availability of security patches and supported algorithms.
    2. **Output Explanation**: “OpenSSL 1.1.1 11 Sep 2018” confirms a modern version with support for updated cryptographic practices.
- **Command**: “openssl ciphers”
    1. **Purpose**: Lists all cryptographic ciphers supported by OpenSSL, allowing us to understand the breadth and security levels of encryption available.
    2. **Output Discussion**: This command outputs a list of ciphers, each suited for different security contexts. Mentioning specific ciphers here would help explain their relevance and applications in securing data.



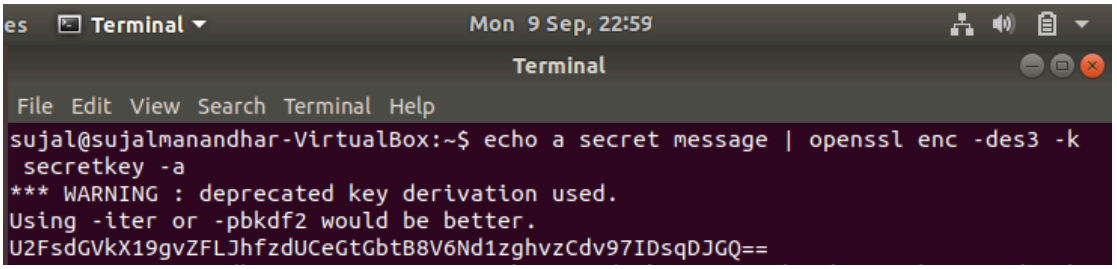
Fig(1): openssl version and openssl ciphers

DES Encryption and Decryption

Background on DES: Despite being historically significant, DES is now considered weak due to its short key length. Discussing its limitations provides insight into why modern cryptography has moved to more secure algorithms like AES.
Now, demonstrating the use of OpenSSL to encrypt and decrypt a message using the DES symmetric encryption algorithm.

Encryption Process:

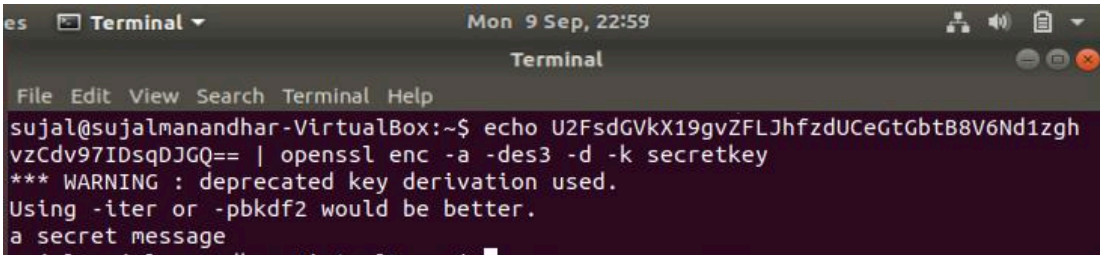
- **Command**: echo "a secret message" | openssl enc -des3 -k secretkey -a
    1. **Purpose**: Encrypts the message using Triple DES, an evolution of the original DES, providing layered security through multiple encryption cycles.
    2. **Key flags**:
        a. **-des3:** Specifies the use of Triple DES.
        b. **-k:** Sets the encryption key.
        c. **-a:** Ensures output in Base64, making the encrypted data ASCII-text friendly.



Fig(2): DES Encryption

Decryption Process:

- **Command**: Reverse the encryption to retrieve original text, highlighting the symmetrical nature of DES.
    1. **Key flags**:
        a. **-d:** Specifies decryption.
        b. **-k:** Uses the same secret key for decryption to demonstrate the reversibility of the process under correct key conditions.



Fig(3): DES Decryption

Security Best Practice Discussion: Post decryption, address the OpenSSL deprecation warning suggesting the use of **"-pbkdf2"** or **"iter"**, which are methods to strengthen key derivation against brute-force attacks.

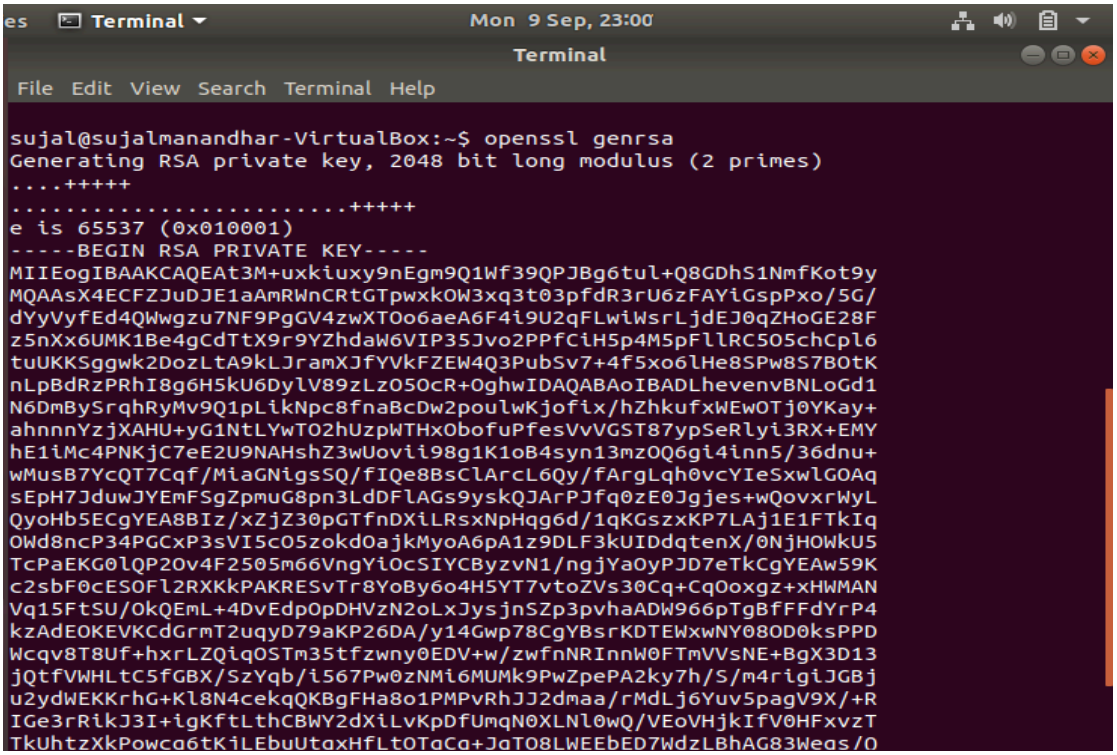
Generating an RSA Private Key

In this section, OpenSSL is used to generate a pair of keys, which is a fundamental concept in asymmetric cryptography. In asymmetric cryptography, there are two keys: a private key (which should be kept secret) and a public key (which can be shared publicly).

Key Generation:

- **Command**: "openssl genrsa -out mykey.pem 2048"
    1. **Purpose**: Generates a 2048-bit RSA private key. Discuss why choosing a key length of 2048 bits balances security and performance effectively.

2. **Output Discussion**: The command outputs the key details directly in the console. It's essential to understand the implications of key visibility and the importance of secure storage practices.



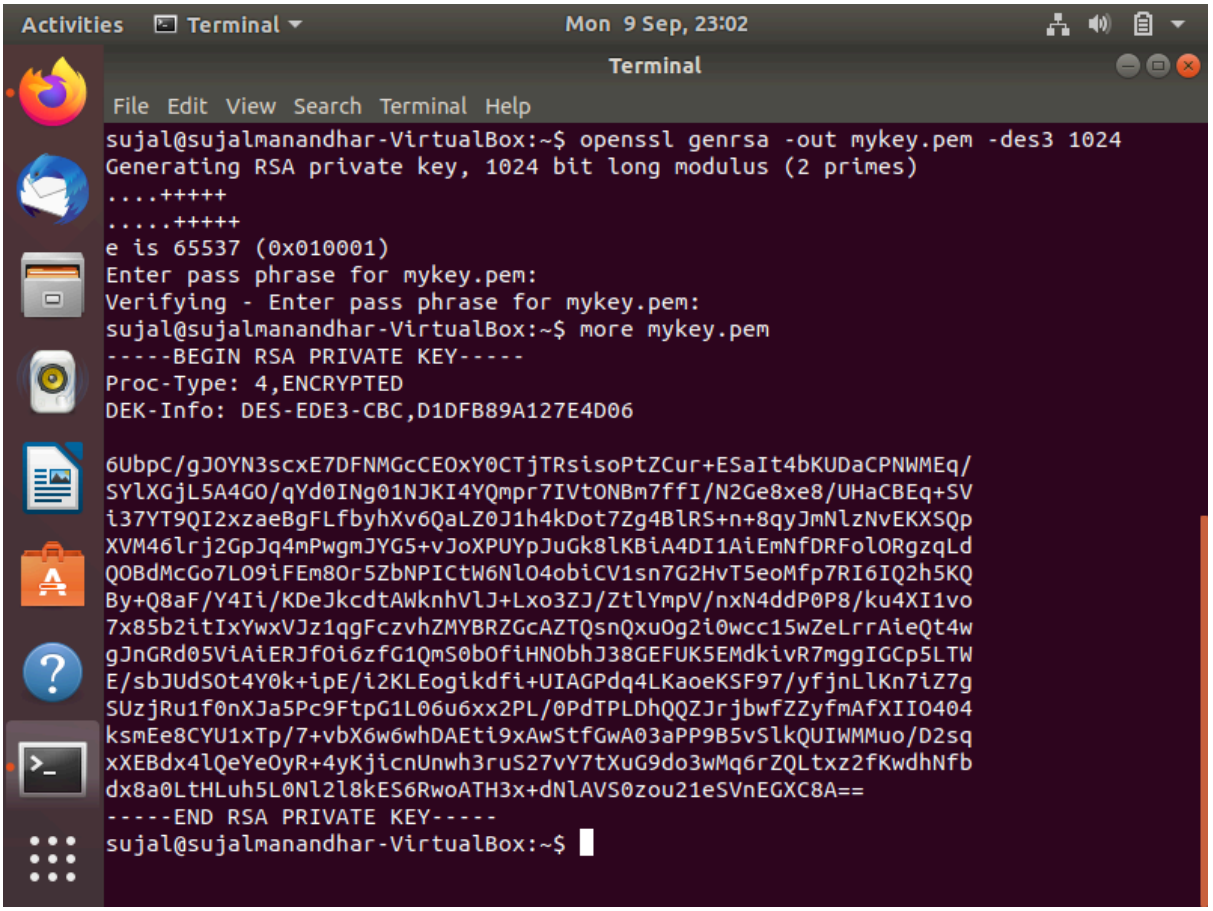
Fig(4): RSA Private Key

The command generates a private RSA key. By default, it creates a 2048-bit key, as illustrated above in the figure.

After generating a private RSA key using the **"openssl genrsa"** command, the next steps typically involve saving the key securely and potentially extracting the public key from the private key, which is crucial in cryptography for verifying signatures or encrypting data.

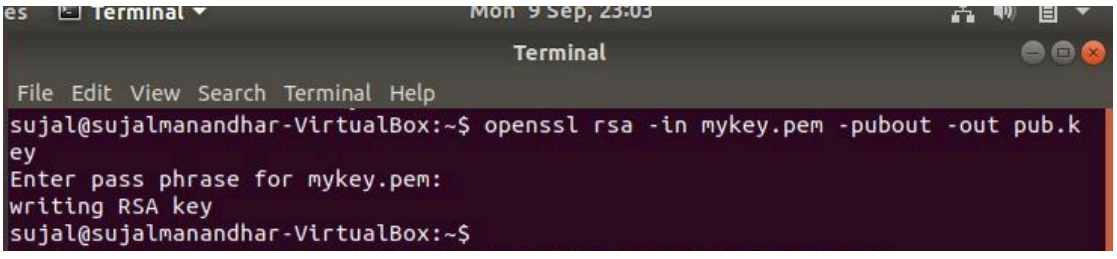
Command: "openssl genrsa -out mykey.pem -des3 1024"

- **Purpose**: Adds a layer of DES3 encryption to the private key for storage, requiring a passphrase to access, enhancing security against unauthorised access.



Fig(5): Generation and Display of Encrypted RSA Private Key

The command **"openssl rsa -in mykey.pem -pubout -out pub.key"** is then run, successfully generating the public key and saving it to **"pub.key"** as shown in the figure(6).



Fig(6): Extracting the Public Key from the Encrypted Private Key

Creation and Verification of a New "Buyorder" File

File Creation and Hashing: Create a file and calculate its **"SHA1" hash**, demonstrating both the creation and integrity verification steps in file management.

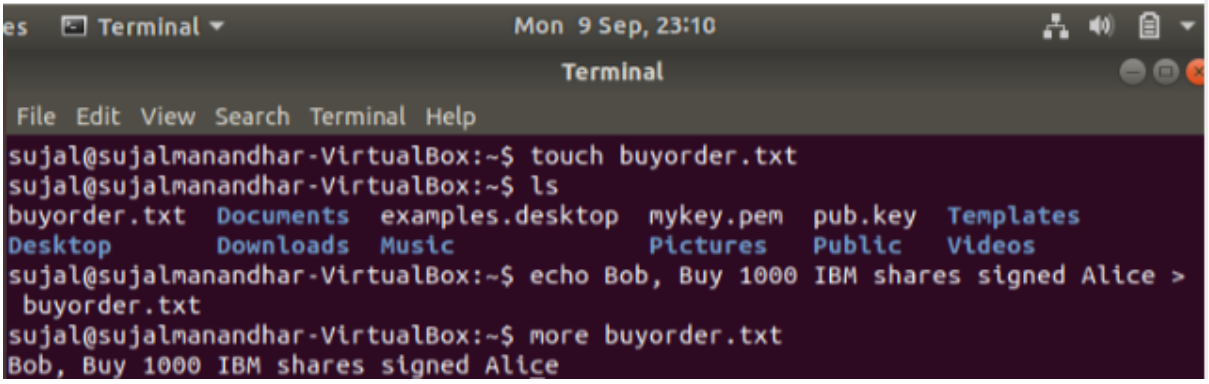
In this part, we need to create the file using **"touch buyorder.txt"**, an empty file creation command in Linux. Following this, the **"ls"** command is used to verify that the file has been

created, listing **"buyorder.txt"** among other directories. The next step involves adding content to the file using the **"echo"** command.

```
sujal@sujalmanandhar-VirtualBox:~$ echo Bob, Buy 1000 IBM shares signed Alice >
 buyorder.txt
```

This command appends the specified text to **"buyorder.txt"**. Then verifies the contents using **"more buyorder.txt"**, which successfully displays:

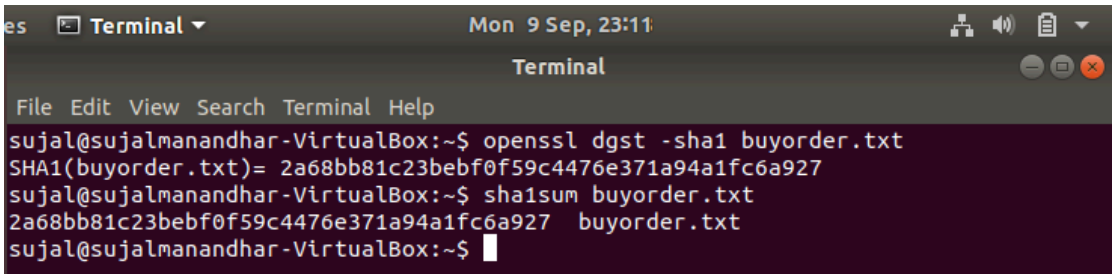
```
Bob, Buy 1000 IBM shares signed Alice
```



Fig(7): Creation and Verification of a New Buyorder File

Generating the "SHA1" Hash of the "Buyorder" File

In this section, we calculate the SHA1 hash of the newly created **"buyorder.txt"** file using two different commands. First, the OpenSSL **"dgst"** command generates a SHA1 digest (hash) of the file. Next, the same hash is verified using the **"sha1sum"** command, another way to compute the **"SHA1"** hash.



Fig(8): Generating the "SHA1" Hash of the "Buyorder" File

The output is consistent, confirming the same value for the file as shown in **figure(8)**.

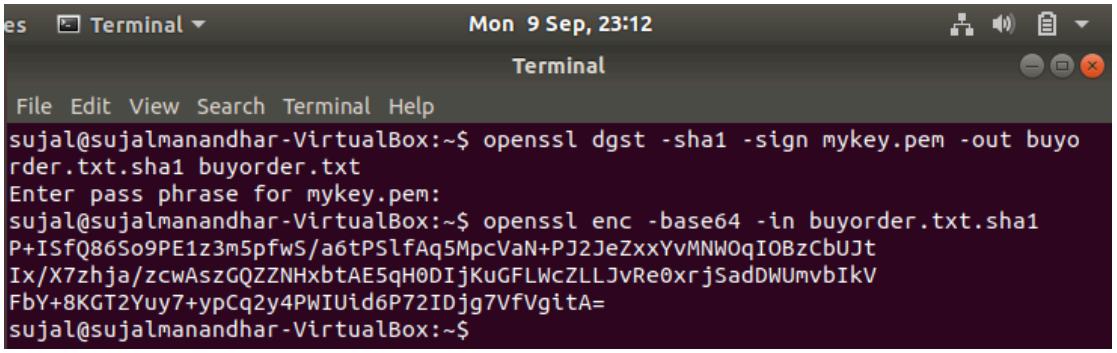
Signing the Hash and Encoding with Base64

The user first signs the previously computed **"SHA1"** hash using a private key stored in **"mykey.pem"**. After entering the passphrase for the private key, the **"SHA1"** hash is signed and stored in the file **"buyorder.txt.sha1"**. The signing process ensures that the hash was generated by the key holder and confirms the integrity of the **"buyorder.txt"** file.

Finally, the signature file is encoded in Base64 format using the command:

```
sujal@sujalmanandhar-VirtualBox:~$ openssl enc -base64 -in buyorder.txt.sha1
```

In **figure(9)**, the last output is the Base64-encoded signature of the **"buyorder.txt"** file.



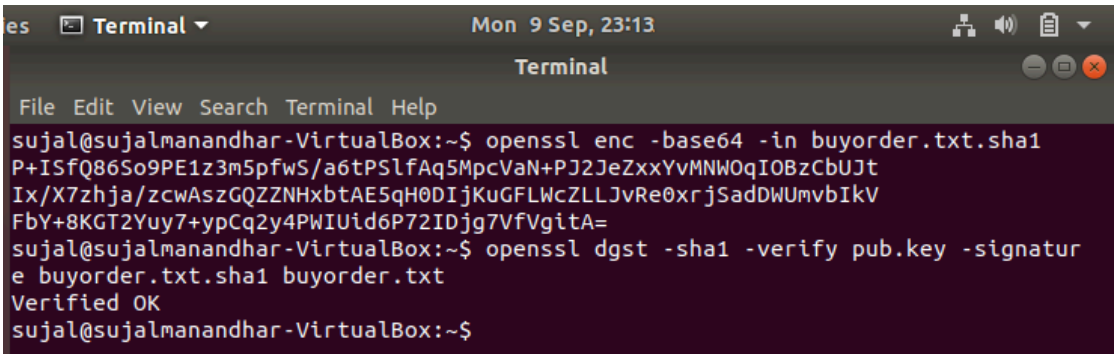
Fig(9): Signing the Hash and Encoding with Base64

Verifying the Signature with Public Key

In the final step, we continue working with the **"buyorder.txt"** file to verify the digital signature created in the previous steps. This process involves Base64 encoding and using the public key for verification.

First, we encode the signed **"buyorder.txt.sha1"** file in Base64 using the **"openssl enc -base64"** command. Then, verify the digital signature using the public key (**"pub.key"**) with the **"openssl dgst -sha1 -verify"** command. This command takes the public key, the signed hash file (**"buyorder.txt.sha1"**), and the original file (**"buyorder.txt"**). If the signature matches the original file and is verified using the public key.

"Verified OK" is displayed, which proves that the signature was made by the owner of the private key and the file has not been tampered with.



Fig(10): Verifying the Signature with Public Key

Conclusion

From this lab of practical experience with OpenSSL, helping us understand how it is used to keep communications secure. We learned how to create keys, make certificates, and use different encryption methods to protect data. The exercises showed how encryption and digital certificates help keep information safe and private. Overall, this lab helped us see how important these tools are in making sure our communications are secure and trustworthy.

Simple use of GPG (Lab 2)

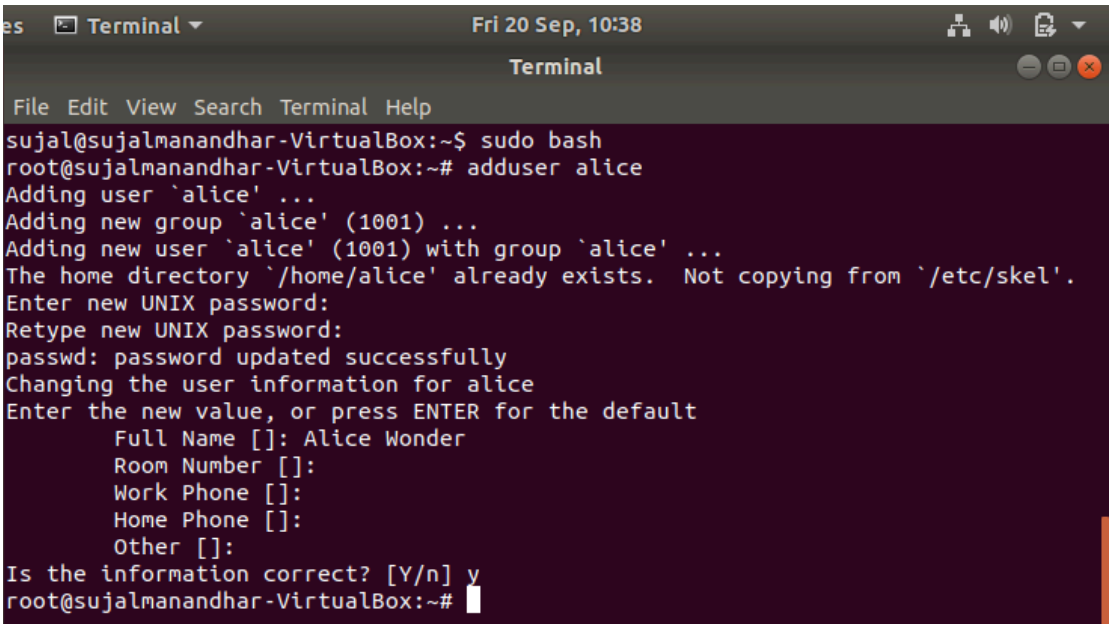
Introduction

GNU Privacy Guard (GPG), a tool for secure communication using asymmetric cryptography. We generated GPG key pairs, encrypted and decrypted messages, and signed files to ensure both security and authenticity. Through this lab, we learned how to securely share public keys and protect private information during exchanges.

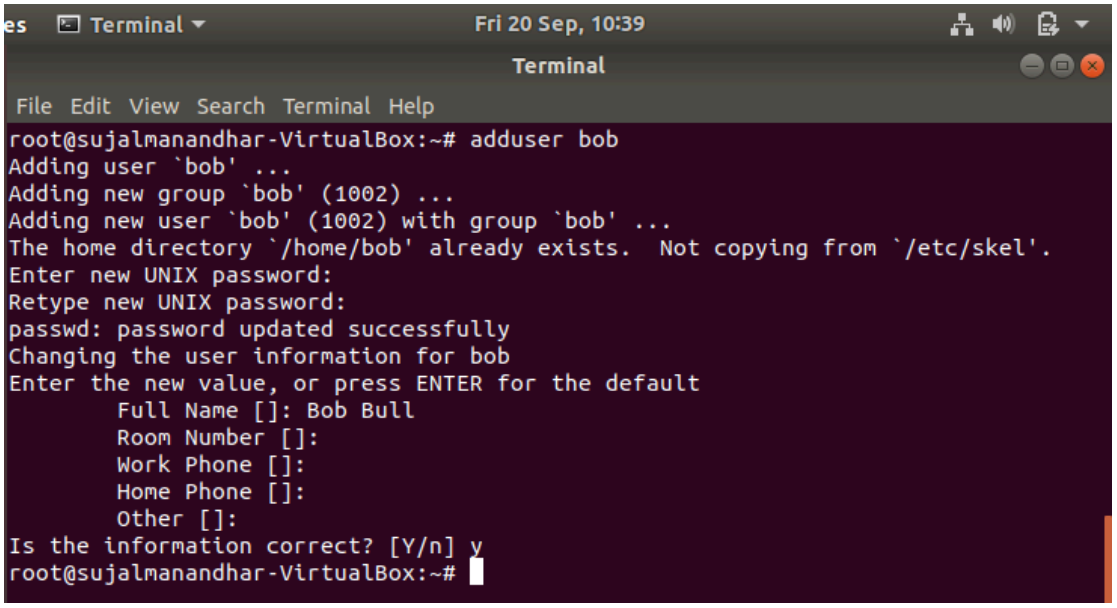
Adding new users

First, we created two new users: **"alice and bob".** This was done using the **"adduser"** command, and both users were assigned administrative privileges using the **"sudo"** group.

- **Command Used**:
    1. **"sudo adduser alice"**
    2. **"sudo adduser bob"**



Fig(11): Creating new user "Alice"



Fig(12): Creating new user "Bob"

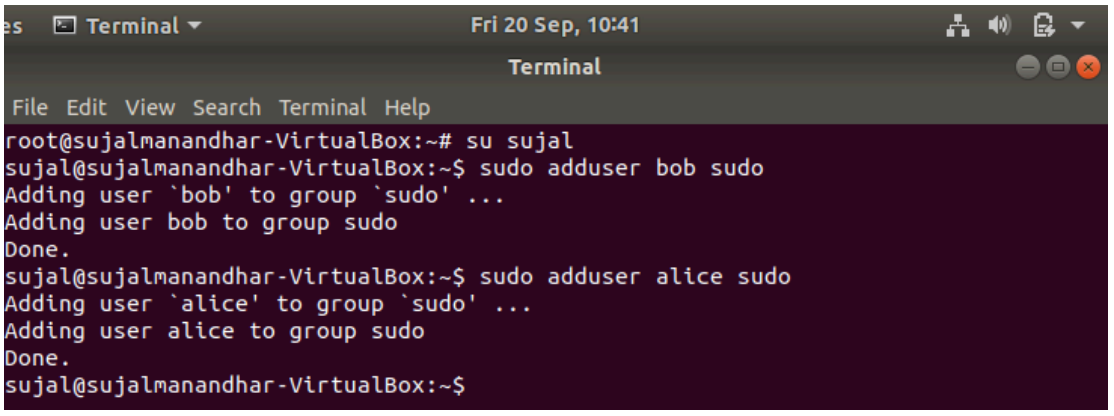
We confirmed the addition of users, and no errors were encountered.

Granting sudo Privileges

After creating the users, we added **"alice and bob"** to the sudo group to give them administrative rights. We used the **"adduser and usermod"** commands to achieve this:

- **Command Used:**
  1. **"sudo usermod -aG sudo alice"**
  2. **"sudo usermod -aG sudo bob"**

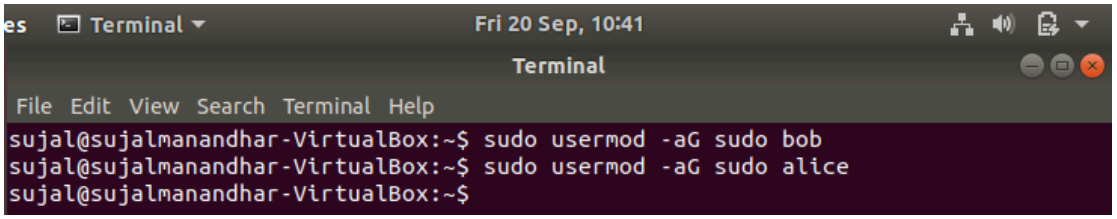
We then verified that both users had **"sudo"** privileges.



Fig(13): Granting sudo privileges to Alice and Bob

Modifying User Privileges

In this step, we used the usermod command to append **“bob** and **alice”** to the **“sudo”** group. The **“-aG”** flag ensures that the users are added to the **“sudo”** group without removing them from any other groups they belong to. This action grants both users administrative privileges, allowing them to run commands with elevated permissions.

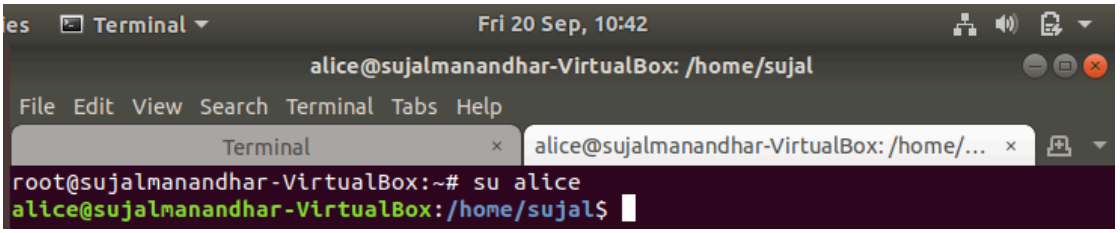


Fig(14): Modifying Bob and Alice's group membership using “usermod”

Switching Users and Verifying Directory

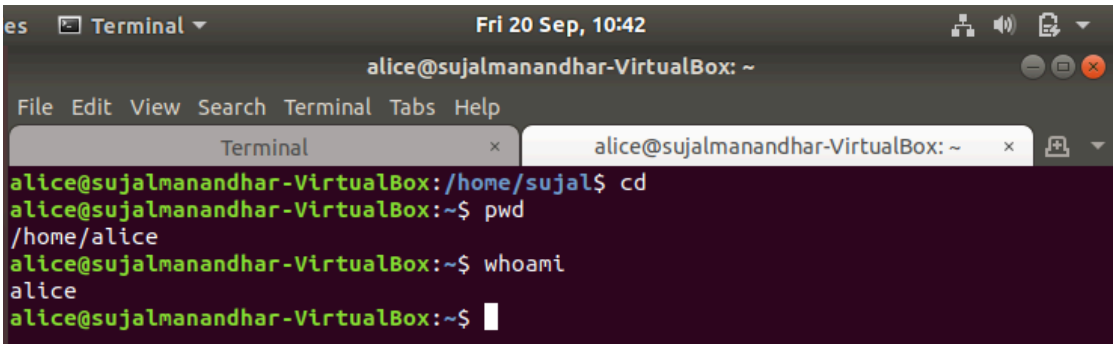
We switched to the **“alice and bob”** user accounts using the su command. After switching, we verified the current directory and user identity with the **“pwd and whoami”** commands.

- **Command Used:**
    1. **“su alice”**
    2. **“pwd”**
    3. **“whoami”**



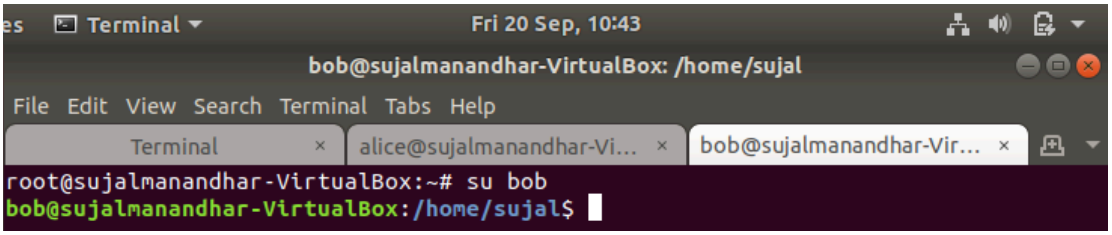
Fig(15):Switching to User Alice

After switching to the **“alice”** user, we ran a series of commands to verify the current working directory and active user. The “**pwd”** command confirmed that we are in **alice's** home directory (**/home/alice**), while the **“whoami”** command verified that we are indeed logged in as **alice**.

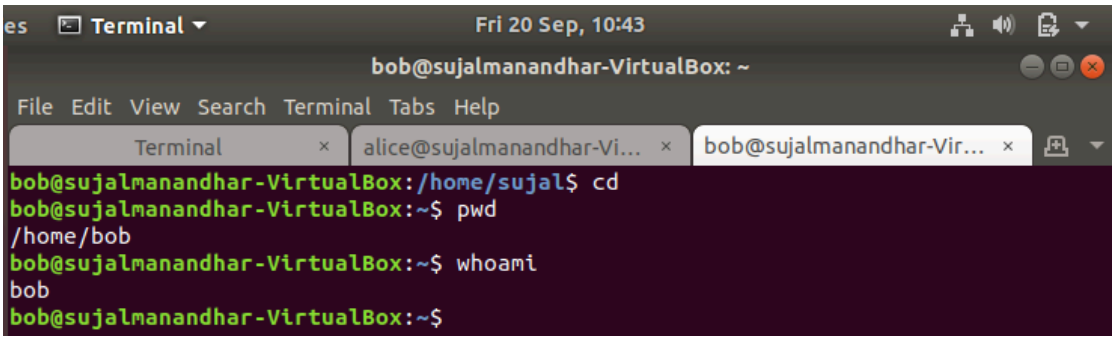


Fig(16): Verifying User and Directory for Alice

Next, we switched to the **"bob"** user account and performed similar verification checks. We used the **"pwd"** command to confirm that we are in **bob's** home directory, and the **"whoami"** command to ensure that we are logged in as **bob**.



Fig(17): Switching to User "Bob"

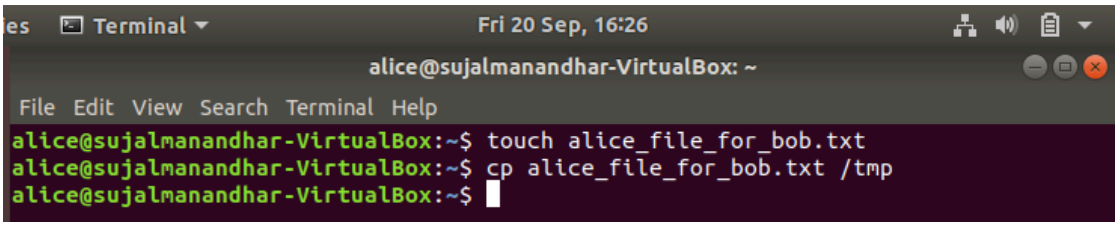


Fig(18): Verifying User and Directory for "Bob"

Creating and Copying Files

Next, while logged in as **alice**, we created a file named **"alice_file_for_bob.txt"** and attempted to copy it to the **"/tmp"** directory. And, then copy the file successfully.

- **Command Used**:
    1. **"touch alice_file_for_bob.txt"**
    2. **"cp alice_file_for_bob.txt /tmp"**

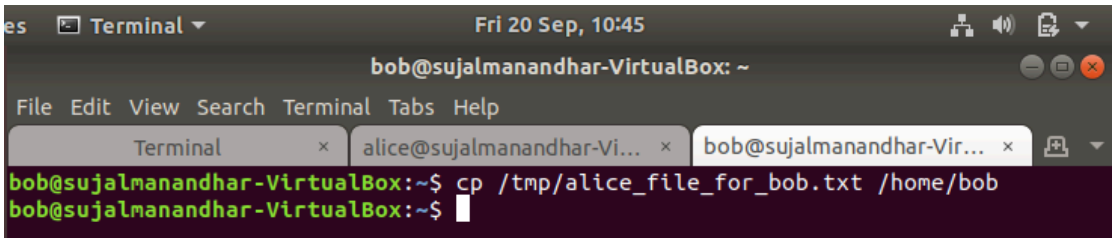


Fig(19): Creating and copying files with Alice

Copying Files in Linux with Bob's Account

In this step, we switched to the **bob** user account and copied the file **"alice_file_for_bob.txt"**

from the **"/tmp"** directory to **bob's** home directory.

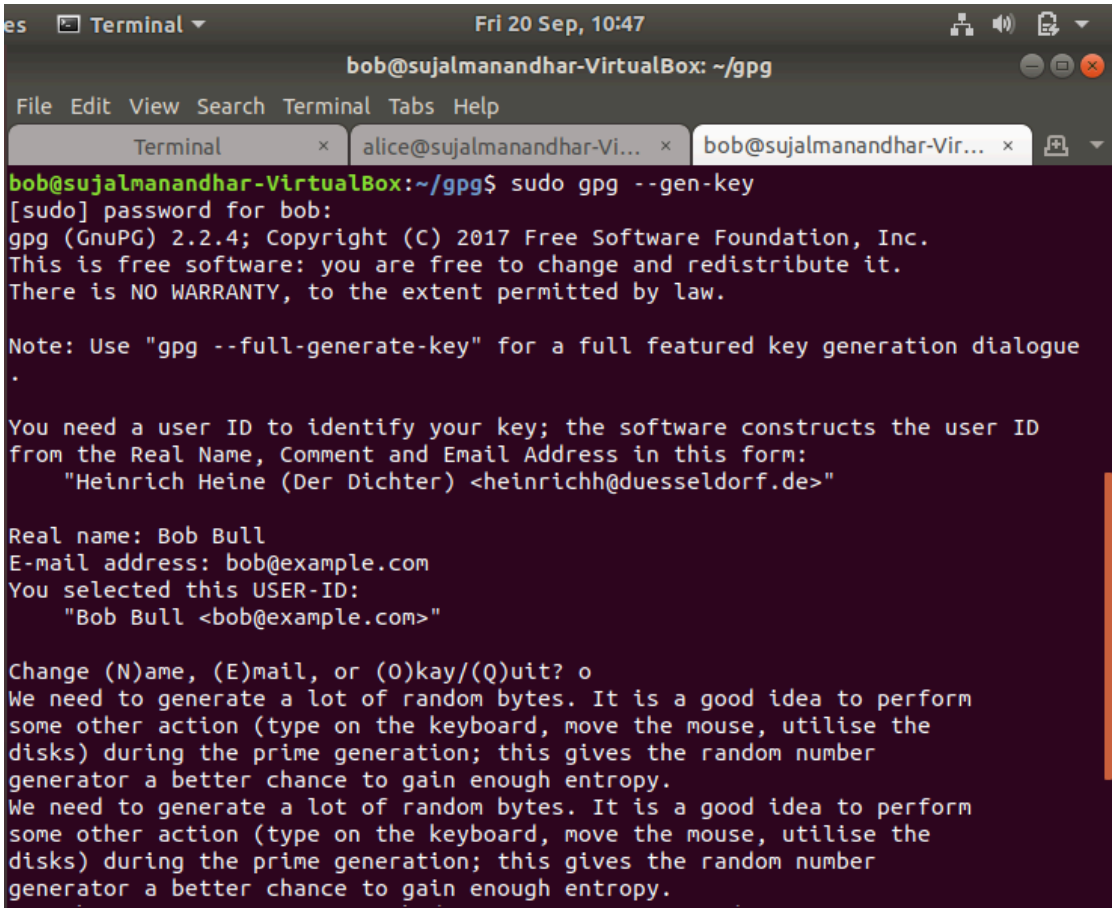


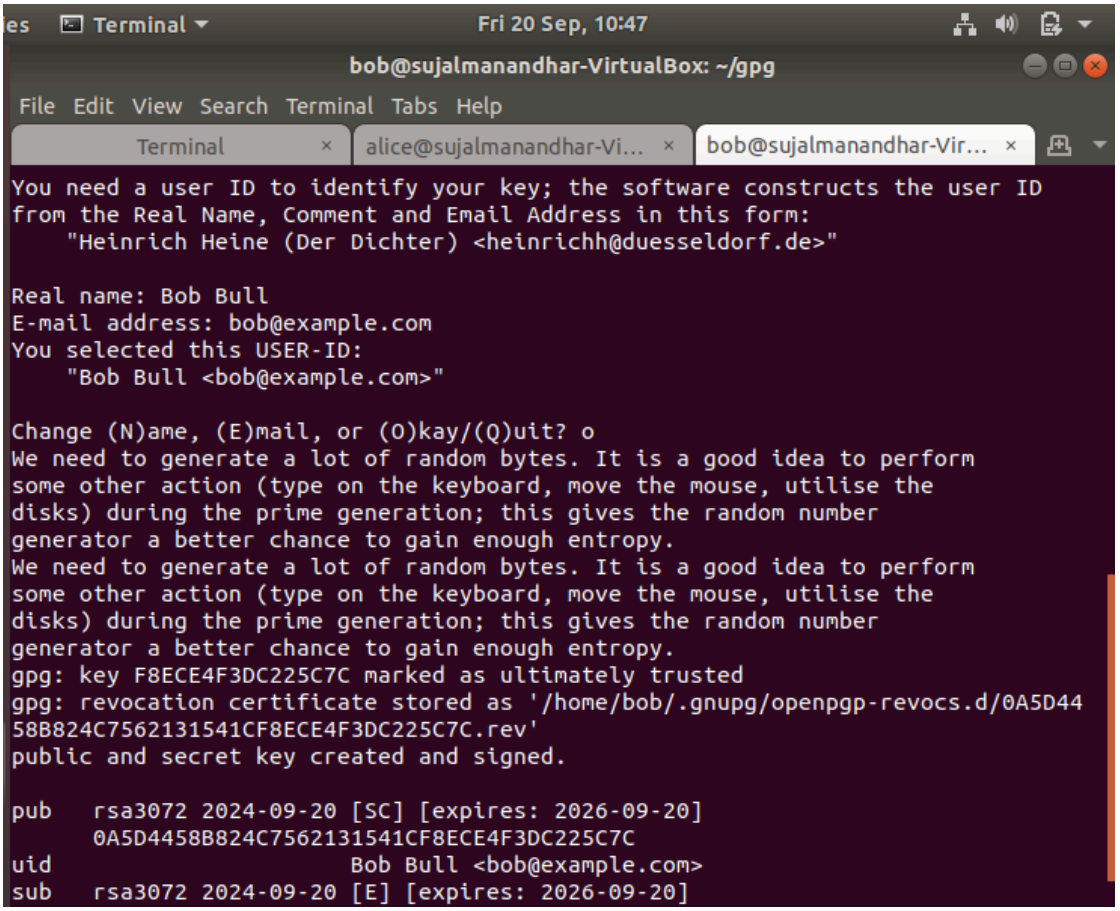
Fig(20): Copying files with Bob

Generating GPG Keys as Bob and Alice

In this step, we used the **"gpg --gen-key"** command to generate a new GPG key pair while logged in as **bob**. After entering the required password, we were prompted to provide a user ID for the key, which includes a real name and email address. We chose **"Bob Bull"** as the real name and **"bob@example.com"** as the email. The system constructed the user ID in the format **"Bob Bull bob@example.com"**.

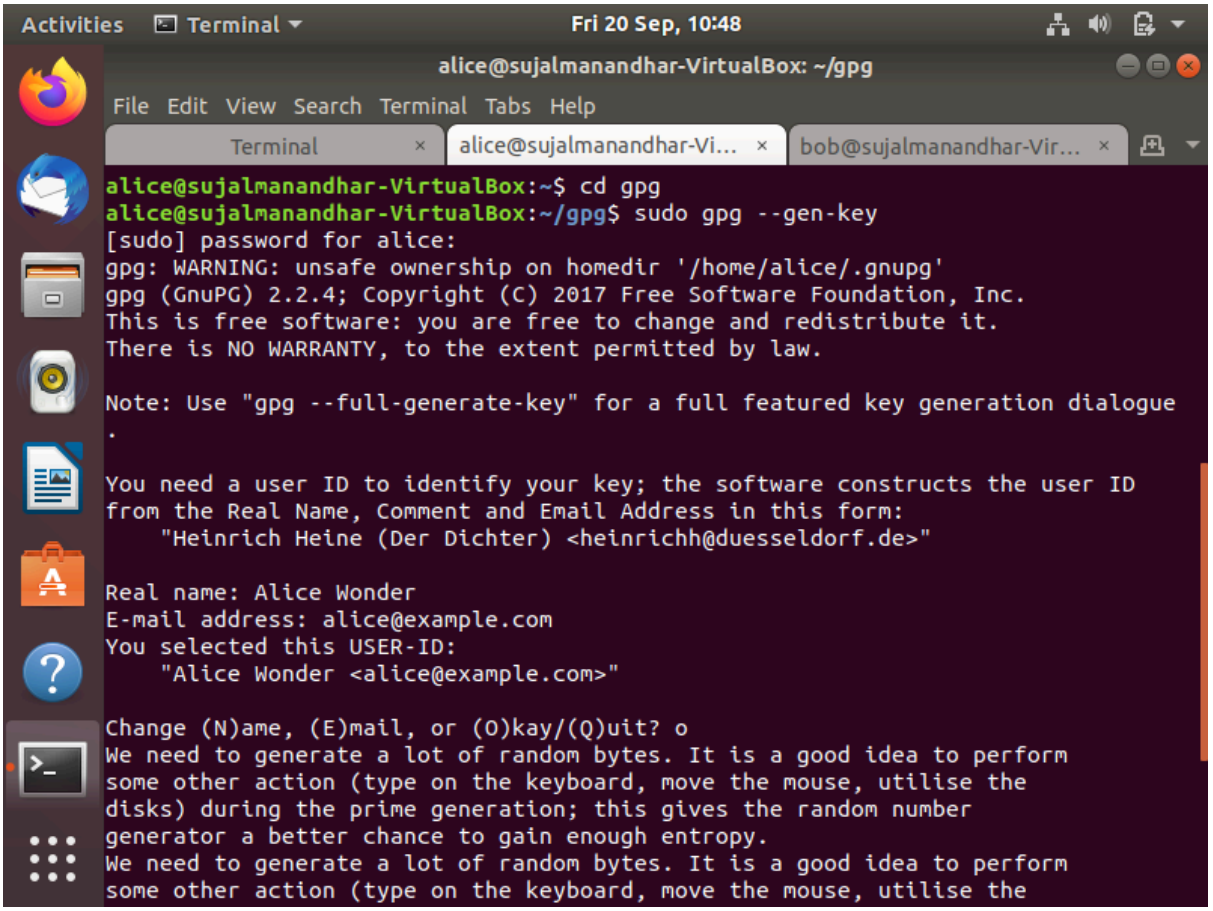
During the key generation process, we were advised to perform additional actions (like typing or moving the mouse) to help the random number generator accumulate enough entropy, which is essential for secure key generation.

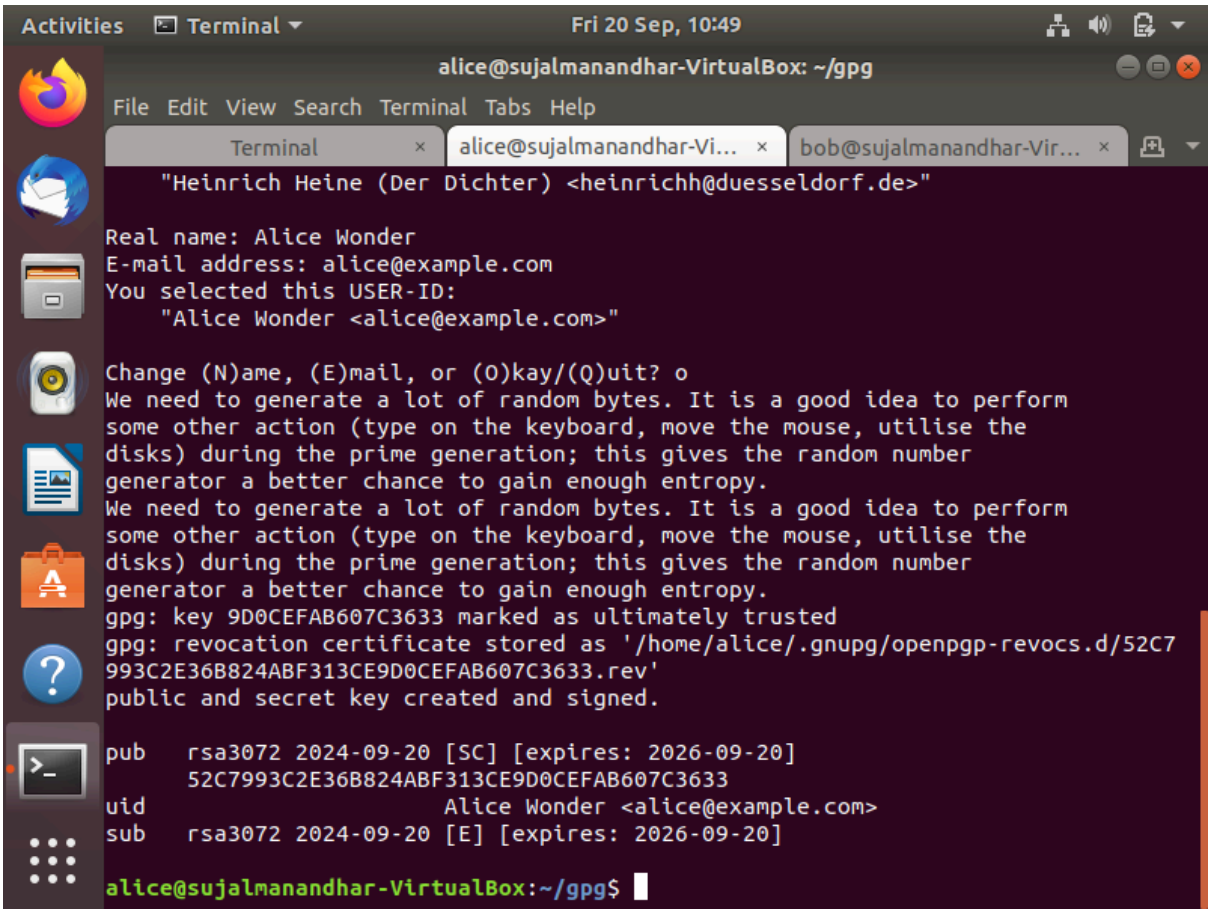




Fig(21): Generating GPG key pair for Bob

Next, we switched to **alice's** terminal and generated a new GPG key pair using the same process as we did for **bob**.


Activities
Terminal
Fri 20 Sep, 10:48
alice@sujalmanandhar-VirtualBox: ~/gpg
File Edit View Search Terminal Tabs Help
Terminal
alice@sujalmanandhar-Vi...
bob@sujalmanandhar-Vir...
alice@sujalmanandhar-VirtualBox:~$ cd gpg
alice@sujalmanandhar-VirtualBox:~/gpg$ sudo gpg --gen-key
[sudo] password for alice:
gpg: WARNING: unsafe ownership on homedir '/home/alice/.gnupg'
gpg (GnuPG) 2.2.4; Copyright (C) 2017 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 dialogue
.
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
Real name: Alice Wonder
E-mail address: alice@example.com
You selected this USER-ID:
"Alice Wonder <alice@example.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, utilise 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, utilise the

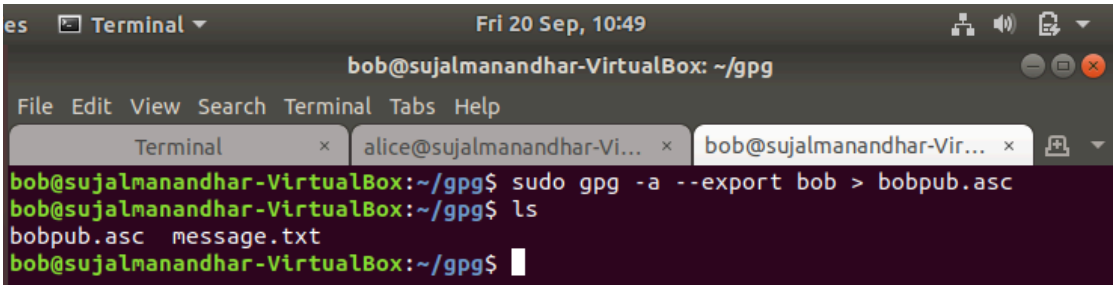

Activities
Terminal
Fri 20 Sep, 10:49
alice@sujalmanandhar-VirtualBox: ~/gpg
File Edit View Search Terminal Tabs Help
Terminal
alice@sujalmanandhar-Vi...
bob@sujalmanandhar-Vir...
"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
Real name: Alice Wonder
E-mail address: alice@example.com
You selected this USER-ID:
"Alice Wonder <alice@example.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, utilise 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, utilise the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key 9D0CEFAB607C3633 marked as ultimately trusted
gpg: revocation certificate stored as '/home/alice/.gnupg/openpgp-revocs.d/52C7
993C2E36B824ABF313CE9D0CEFAB607C3633.rev'
public and secret key created and signed.
pub rsa3072 2024-09-20 [SC] [expires: 2026-09-20]
52C7993C2E36B824ABF313CE9D0CEFAB607C3633
uid Alice Wonder <alice@example.com>
sub rsa3072 2024-09-20 [E] [expires: 2026-09-20]
alice@sujalmanandhar-VirtualBox:~/gpg$

Fig(22): Generating GPG key pair for Alice

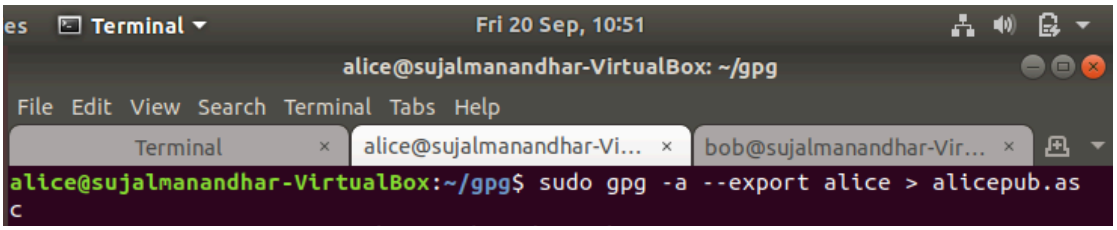
Exporting GPG Public Keys

In this step, we exported **bob's** public key to a file named **"bobpub.asc"** using the command **"gpg -a --export bob"**. After executing the command, we confirmed the successful creation of the file by listing the contents of the directory, which now included **"bobpub.asc"** and **"message.txt"**.

Similarly, we switched to alice's terminal and executed the command **"gpg -a --export alice > alicepub.asc"** to export **alice's** public key into a file named **"alicepub.asc"**.



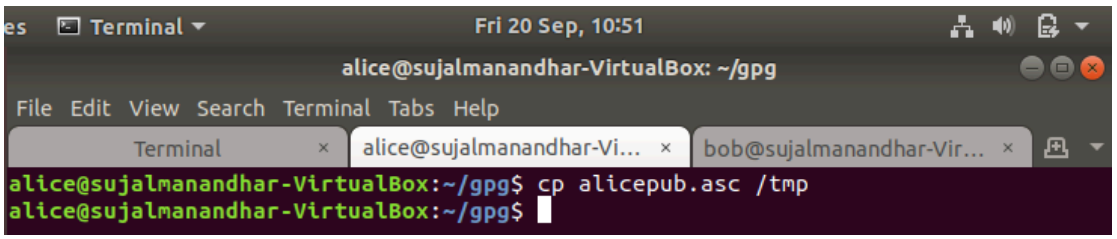
Fig(23): Exporting Public Key for Bob



Fig(24): Exporting Public Key for Alice

Moving Alice's Public Key to "/tmp"

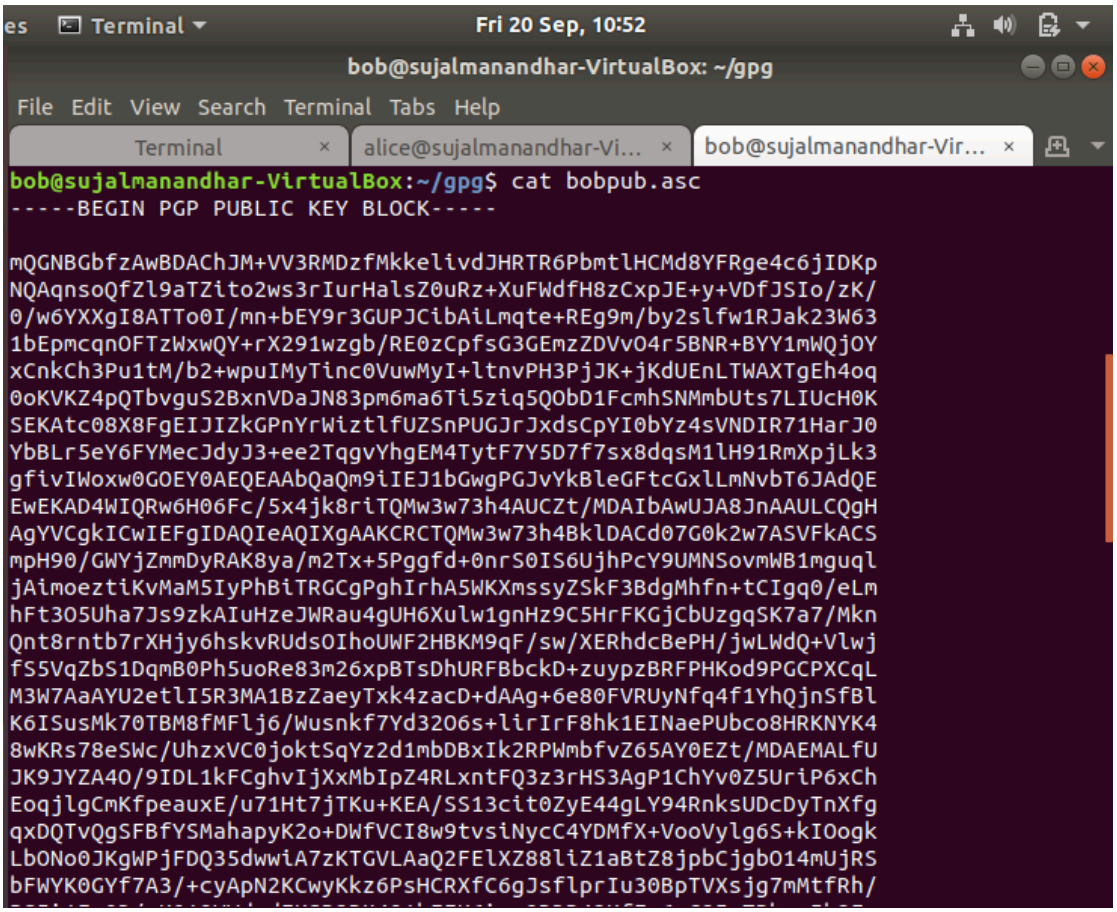
In this step, we are preparing to move **alice's** public key file, **"alicepub.asc"**, to the /tmp directory. This action is often performed to temporarily store files for easy access or sharing. However, it seems the command to execute this action is missing.

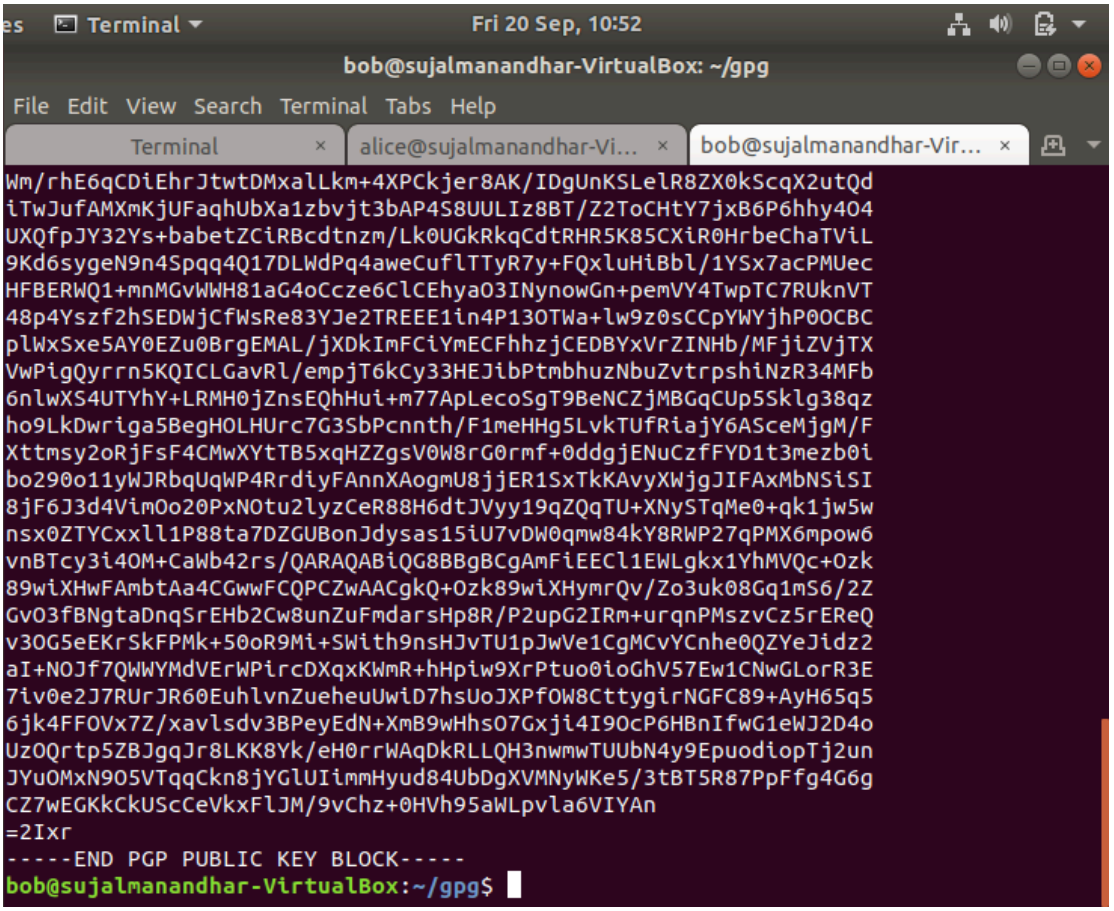


Fig(25): Moving Alice's Public Key to /tmp

Viewing Bob's Public Key

In this step, we switched to user Bob and used the cat command to display the contents of **“bobpub.asc”**, which contains **bob's** public key. The output begins with the header **“-----BEGIN PGP PUBLIC KEY BLOCK-----”**, indicating that this file is formatted as a PGP public key block. This format is standard for sharing public keys in a secure manner.

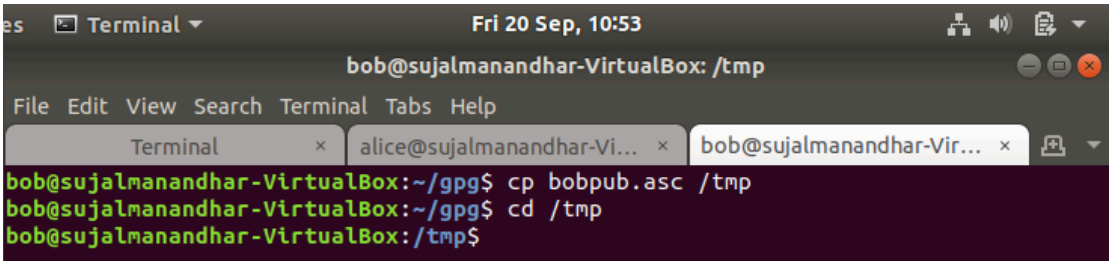




Fig(26): Displaying Bob's Public Key

Copying Bob's Public Key to "/tmp"

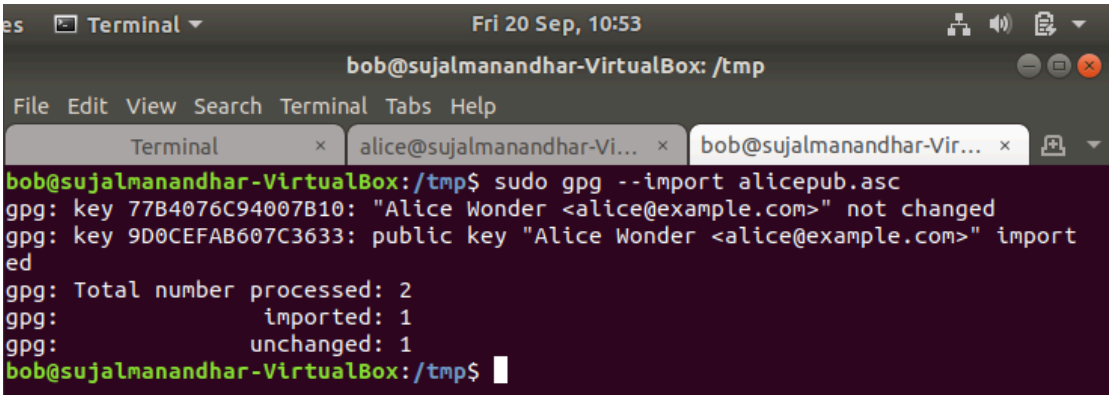
In this step, we copied **bob's** public key file, **bobpub.asc**, to the **"/tmp"** directory using the cp command. This action allows for easy access and sharing of the public key file. After copying, we navigated to the **"/tmp"** directory, indicated by the cd command.



Fig(27): Copying Bob's Public Key to /tmp Directory

Importing Alice's Public Key into Bob's GPG Keyring

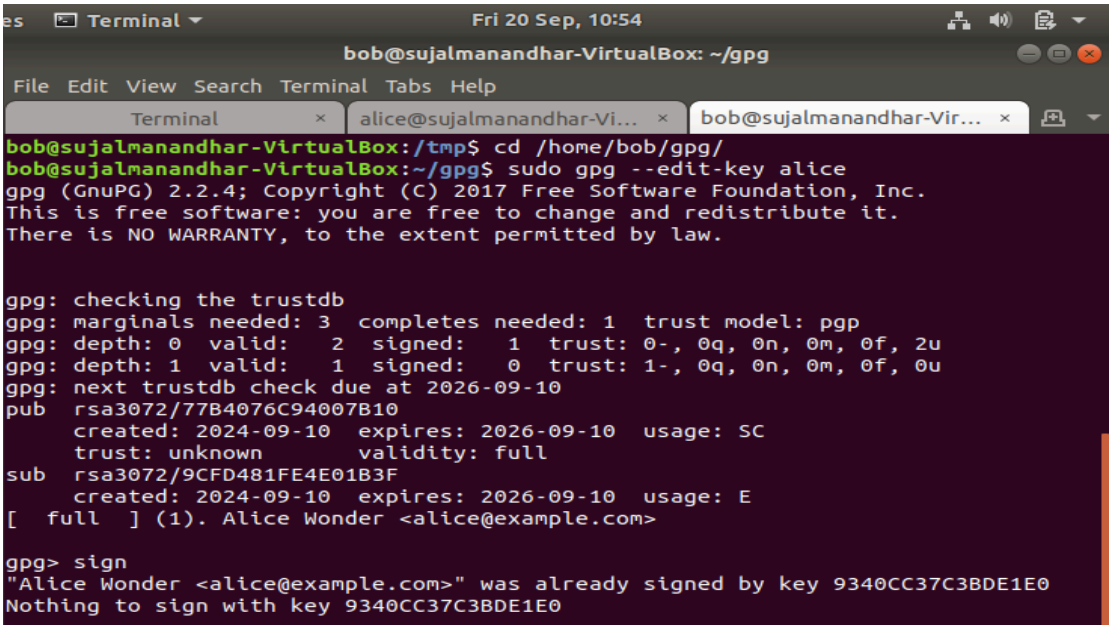
In this step, we imported **alice's** public key from the "**alicepub.asc"** file into **bob's** GPG keyring using the command **"gpg --import alicepub.asc"**. The output indicates that one key was successfully imported while another key was unchanged. Specifically, **bob's** keyring now includes Alice's public key, allowing **bob** to encrypt messages for **alice** or verify messages signed by her.



Fig(28): Importing Alice's Public Key into Bob's GPG Keyring

Editing Alice's GPG Key and Signing

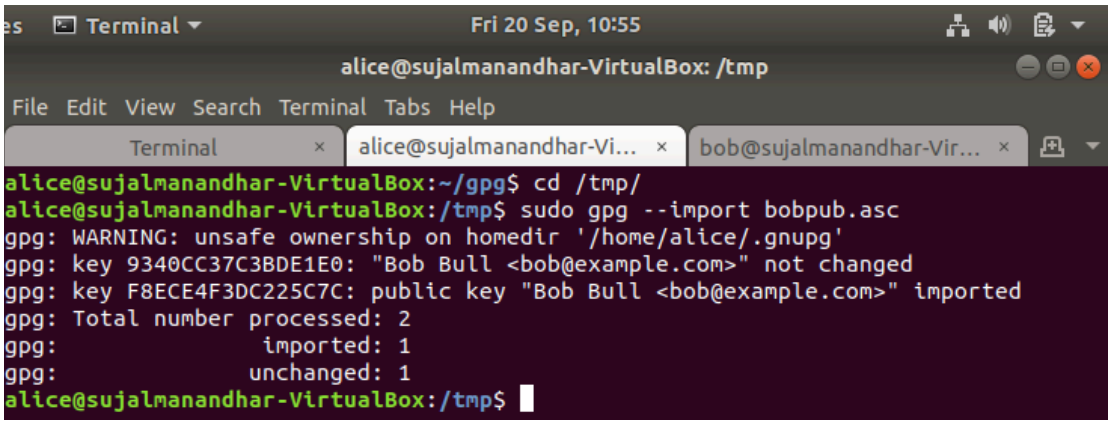
In this step, we navigated to **bob's** GPG directory and initiated the **"gpg --edit-key alice"** command to modify **alice's** public key settings. The output shows the current status of the key, including its validity, trust levels, and usage information.
While attempting to sign **alice's** key, we received a message indicating that the key had already been signed by another key (**key ID: 9340CC37C3BDE10**), suggesting that no further action was needed for signing.



Fig(29): Editing and Attempting to Sign Alice's GPG key

Importing Bob's Public Key into Alice's GPG Keyring

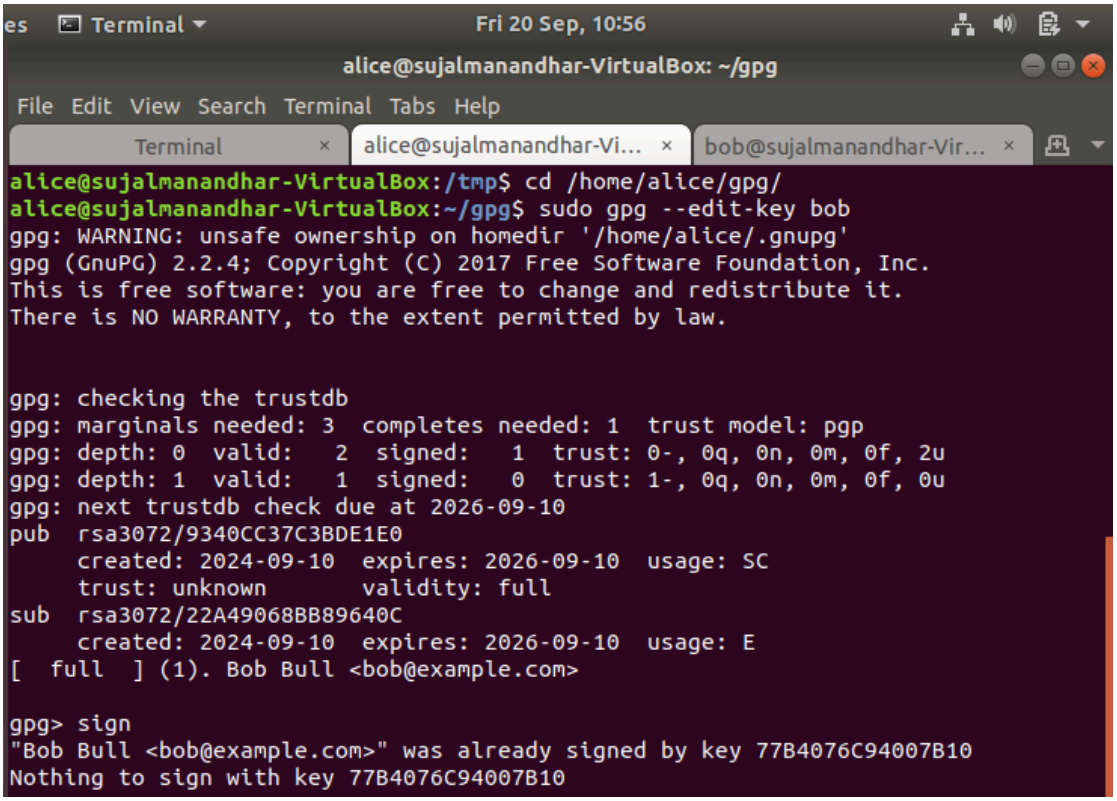
In this step, **alice** imported **bob's** public key by using the command **"gpg --import bobpub.asc"**. A warning about unsafe ownership on the GPG home directory was issued, which may indicate permission or ownership issues. Despite this, the import was successful, with one key being imported and another remaining unchanged, allowing **alice** to now send encrypted messages to **bob** or verify messages signed by him.



Fig(30): Importing Bob's Public Key into Alice's GPG Keyring

Verifying Bob's Public Key in Alice's GPG Keyring

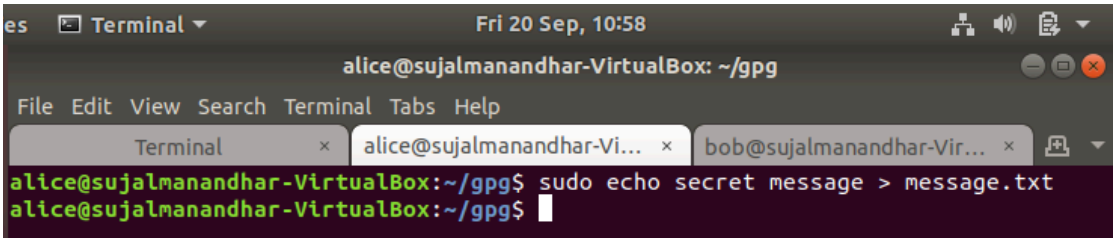
In this step, **alice** used the command **"gpg --edit-key bob"** to review and sign **bob's** public key. Despite a warning regarding unsafe ownership of the GPG home directory, the process was successful. The output indicates that **bob's** key was already signed by **alice's** key, confirming that **alice** trusts **bob's** public key for secure communication.



Fig(31): Verifying and Signing Bob's Public Key

Creating a Secret Message File as Alice

In this step, we created a file named **"message.txt"** containing the text **"secret message"** while logged in as **alice**. The command **"sudo echo secret message > message.txt"** was intended to write the message to the file, but using **"sudo"** with **"echo"** can sometimes lead to permission issues due to how redirection works. It's typically better to create the file without sudo or use a different method to ensure the message is saved correctly.

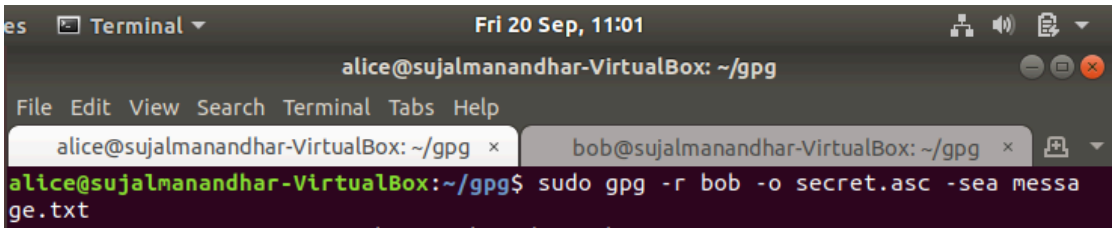


Fig(32): Creating a Secret Message

Encrypting a Message for Bob

In this step, we encrypted the message.txt file for bob using GPG. The command **"gpg -r bob -o secret.asc -sea message.txt"** specifies that we want to encrypt the file as a secret message,

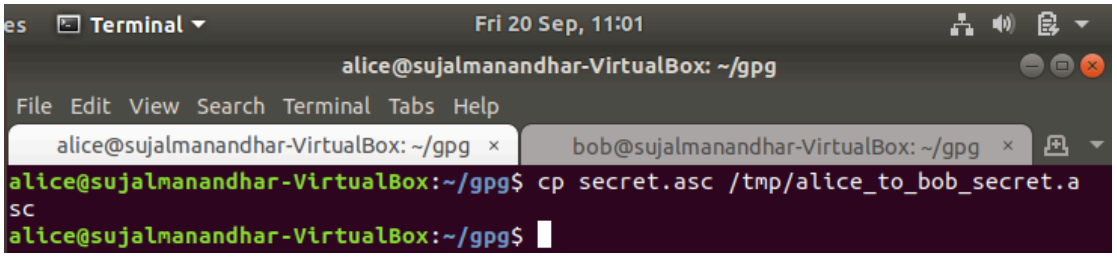
designating bob as the recipient. The output file, secret.asc, will contain the encrypted message, ensuring that only **bob** can decrypt and read its contents using his private key.



Fig(33): Encrypting a Message for Bob using GPG

Copying the Encrypted Message to "/tmp"

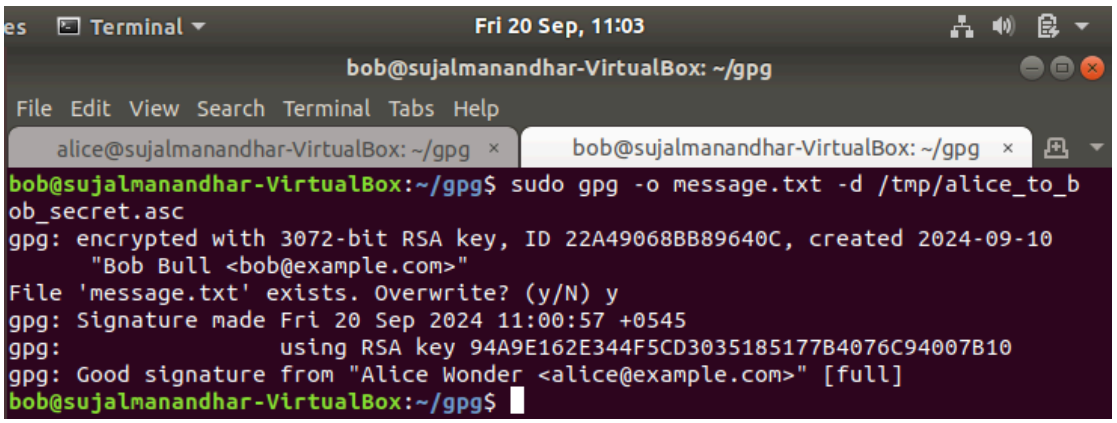
In this step, we copied the encrypted file "**secret.asc"** to the **"/tmp"** directory and renamed it to "**alice_to_bob_secret.asc"**. This action helps facilitate the sharing of the encrypted message with **bob** while ensuring it is stored in a temporary location for easy access.



Fig(34): Copying the Encrypted Message to /tmp

Decrypting Alice's Message as Bob

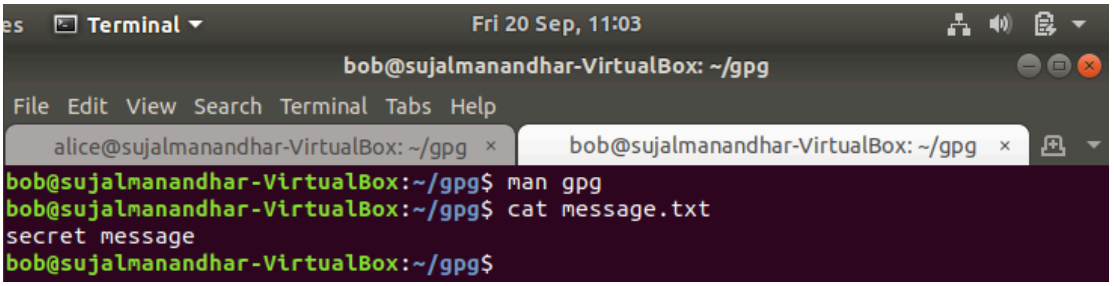
In this step, **bob** used the command **"gpg -o message.txt -d /tmp/alice_to_bob_secret.asc"** to decrypt the encrypted message file sent by **alice**. After confirming the overwrite of the existing **"message.txt"** file, the decryption process was completed successfully. The output indicates that the message was encrypted with **alice's** RSA key and verifies the signature, confirming that it was signed by **"Alice Wonder"**.



Fig(35): Decrypting Alice's Encrypted Message as Bob

Viewing the Decrypted Message

In this step, **"bob"** accessed the manual for the **"gpg"** command using **"man gpg"** for more information on its usage. Afterward, **"bob"** opened the decrypted **"message.txt"** file with the cat command, which displayed the content: **"secret message"**. This confirms that the decryption was successful, and **"bob"** can now read the original message sent by "**alice"**.



Fig(36): Viewing the Decrypted Message

Conclusion

In this lab, we explored the use of GPG for secure communication between users by generating key pairs, exporting/importing public keys, and encrypting/decrypting messages. This exercise demonstrated the practical implementation of cryptography in securing message exchanges, ensuring that sensitive data remains confidential between the communicating parties. By following these steps, we learned how to manage public and private keys effectively and how encryption can be used in real-world scenarios.

Discretionary Access Control (Lab 5)

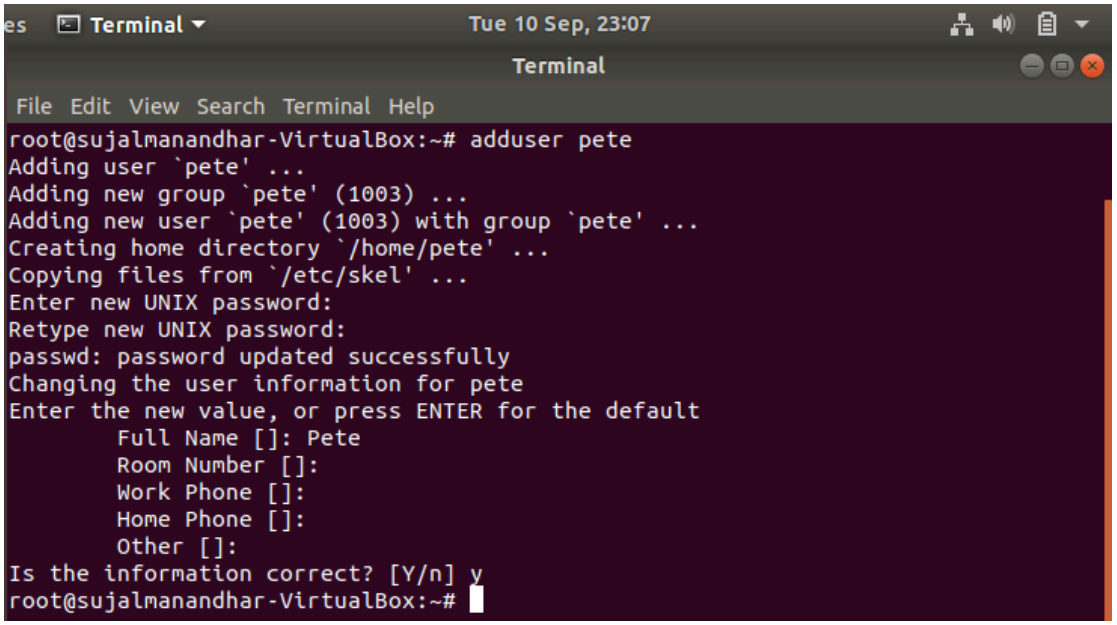
Introduction: Discretionary Access Control (DAC) is a type of access control model where the resource owner determines the access rights for other users. In DAC, permissions are typically managed via file ownerships, user groups, and permission settings. In this lab, we demonstrate how to create users, groups, and control access through various permission settings.

Adding New Users

To begin, new users are added to the system with dedicated home directories. The following command sequence creates a new user, sets a password, and confirms the user details:

Command: "sudo adduser pete"

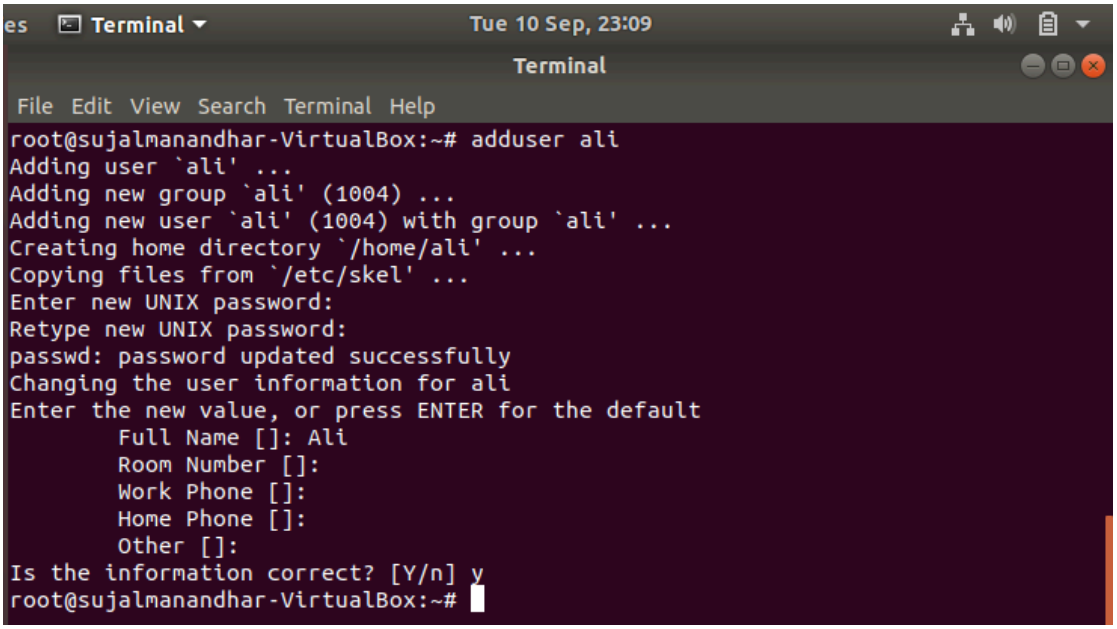
1. **Explanation**: This command creates a new user **"pete"** with a home directory. The user's account details are also set up, including the password.



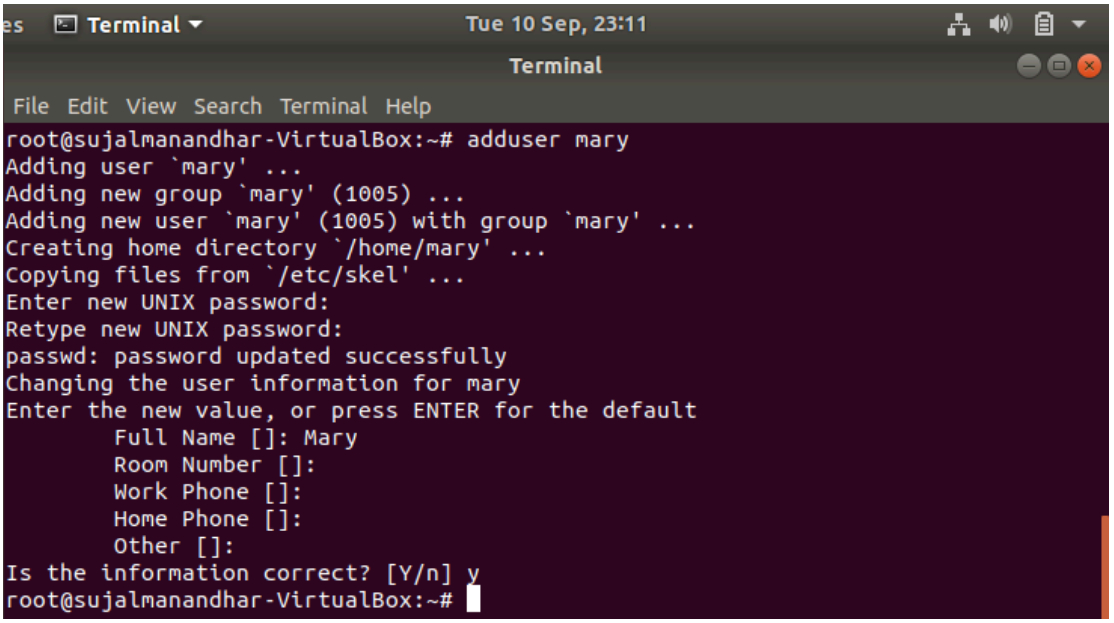
Fig(37): New User "Pete"

Repeat the same process to create users **ali** and **mary**:

- **Command:**
    1. sudo adduser ali
    2. sudo adduser mary



Fig(38): New User "Ali"

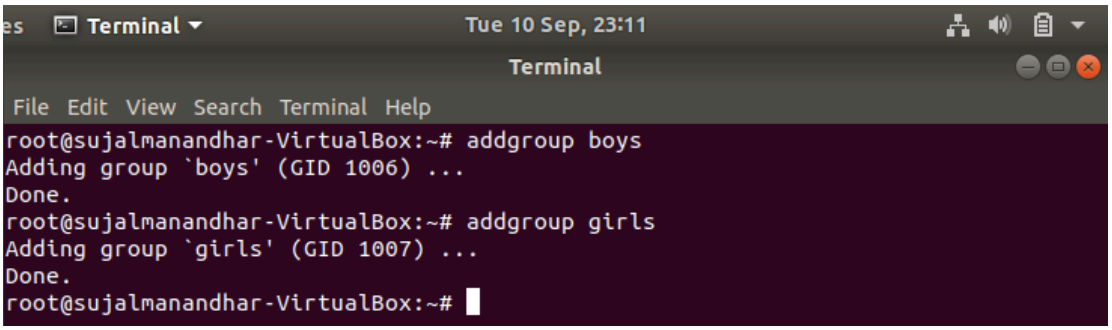


Fig(39): New User "Mary"

Creating Groups

Next, we define two new groups: "boys" and "girls", which are assigned specific Group IDs (**GID**).

1. **Command:** "sudo groupadd boys"
- **Explanation**: Creates the "boys" group with GID 1006.
2. **Command**: sudo groupadd girls
- **Explanation**: Creates the "girls" group with GID 1007.



Fig(40): Creating User Groups

Successfully created groups “boys” and “girls” with GIDs 1006 and 1007 respectively.

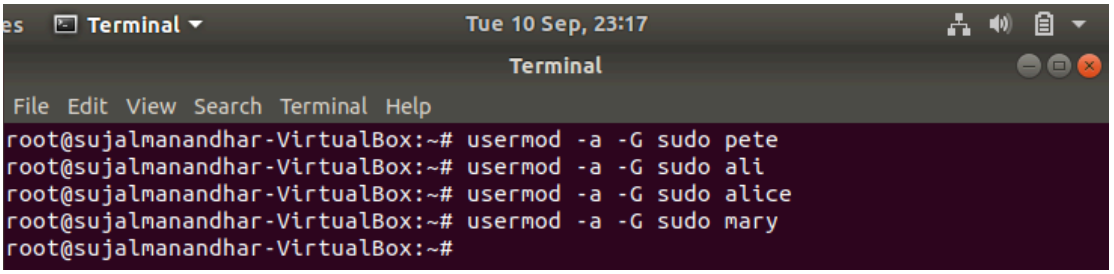
Adding Users to the Sudo Group

For administrative privileges, we add the new users to the sudo group, allowing them to execute privileged commands.

- **Command: “**sudo usermod -aG sudo pete”
    1. **Explanation**: This adds **“pete”** to the sudo group, granting administrative privileges.

The same steps are repeated for users **ali, alice** and **mary**.

- **Command:**
    1. “sudo usermod -aG sudo ali”
    2. “sudo usermod -aG sudo alice”
    3. “sudo usermod -aG sudo mary”



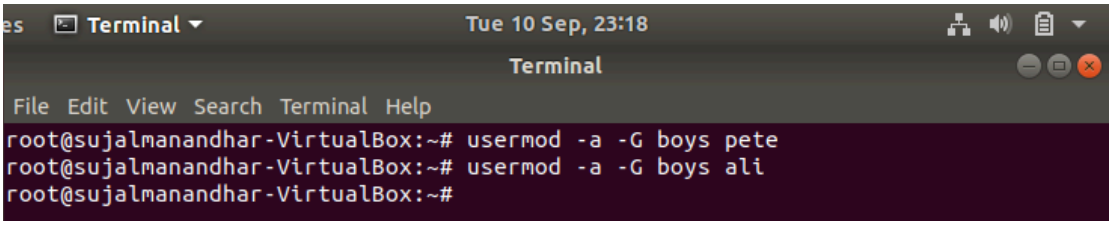
Fig(41): Users Added to Sudo Group

All specified users are successfully added to the **“sudo”** group.

Adding Users to Specific Groups

This section demonstrates adding users to specific groups, facilitating shared access control for members of a group. The “boys” and “girls” groups are used here.

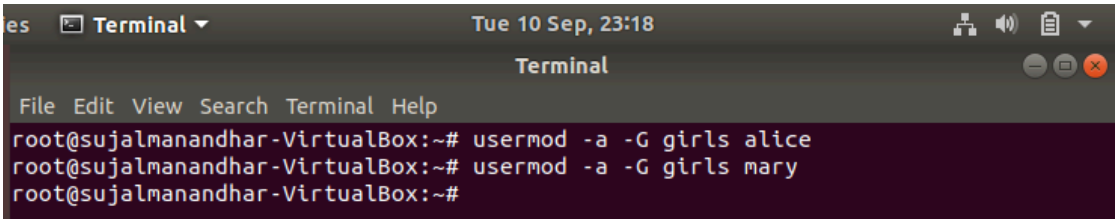
- **Commands:**
    1. “sudo usermod -aG boys pete”
    2. “sudo usermod -aG boys ali”
- **Purpose**: Adds Pete and Ali to the “boys” group.



Fig(42): Users Added to Boys Group

Similarly, add Alice and Mary to the “girls” group:

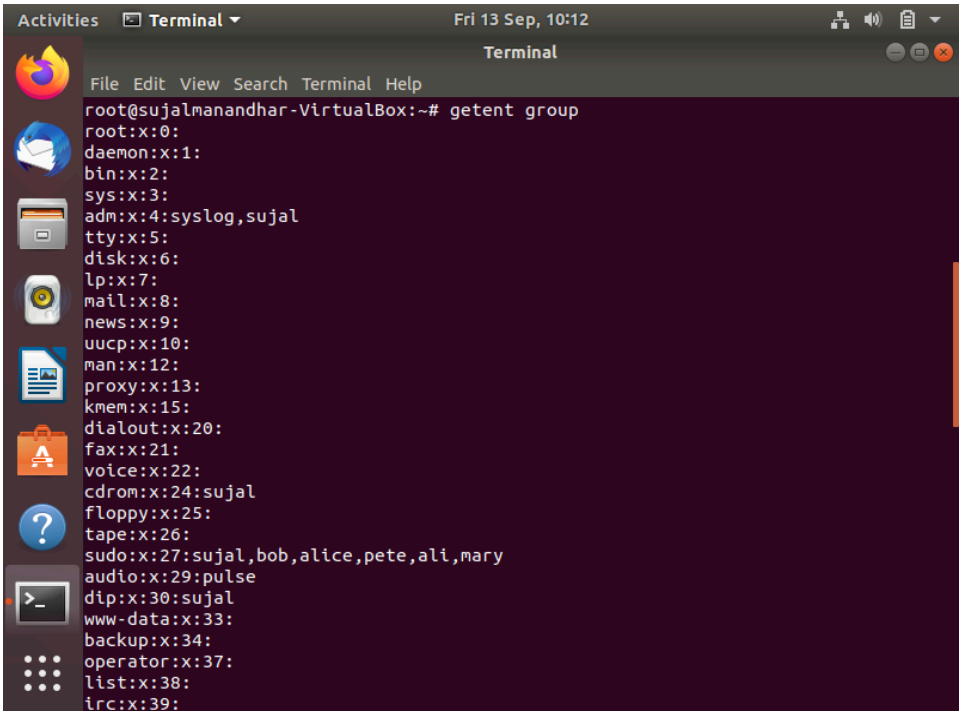
- **Commands:**
    1. “sudo usermod -aG girls alice”
    2. “sudo usermod -aG girls mary”

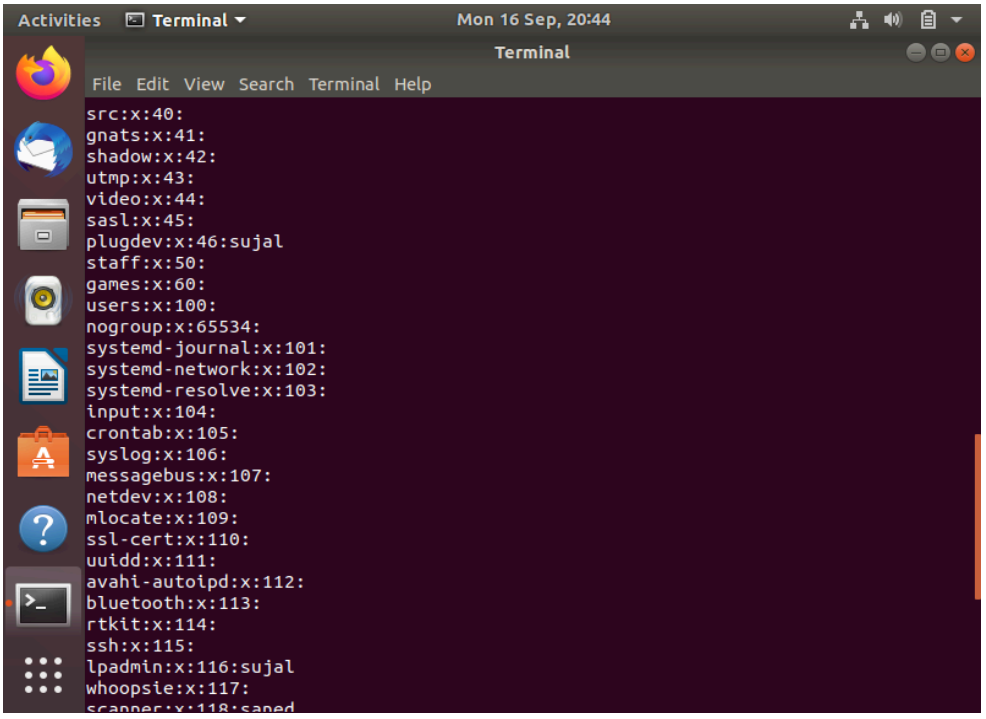


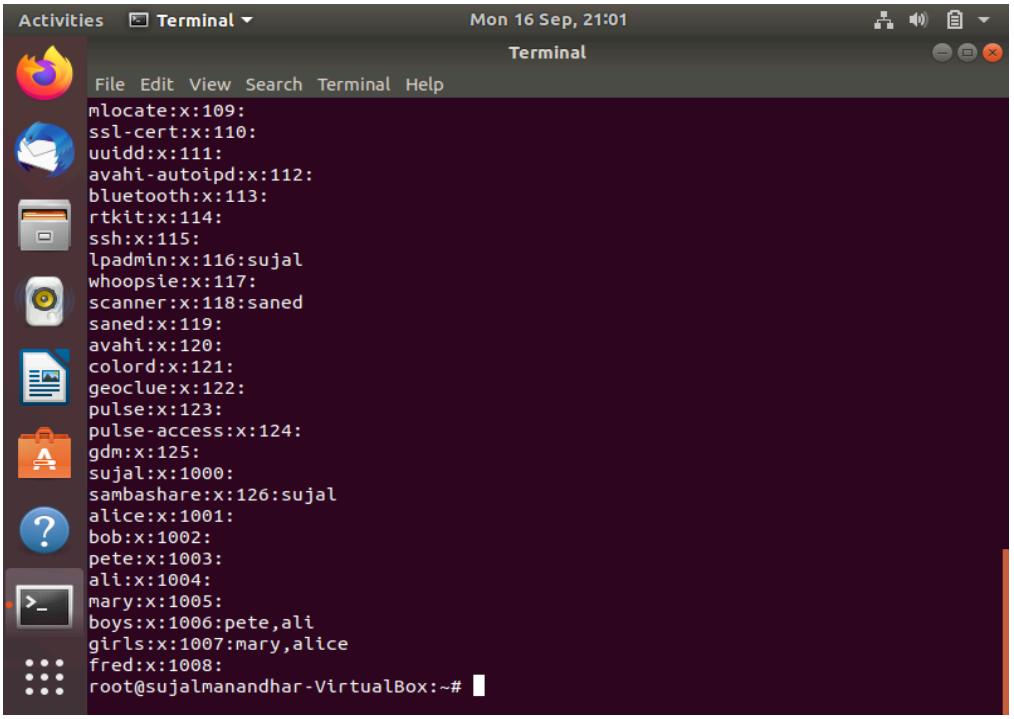
Fig(43): Users Added to Girls Group

Viewing Group Information

- **Command: “**getent group”
- **Purpose:** Displays a list of all groups and their associated members, providing a quick way to verify group memberships and GIDs.


Activities
Terminal
Fri 13 Sep, 10:12
Terminal
File Edit View Search Terminal Help
root@sujalmanandhar-VirtualBox:~# getent group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,sujal
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
cdrom:x:24:sujal
floppy:x:25:
tape:x:26:
sudo:x:27:sujal,bob,alice,pete,ali,mary
audio:x:29:pulse
dip:x:30:sujal
www-data:x:33:
backup:x:34:
operator:x:37:
list:x:38:
irc:x:39:


Activities
Terminal
Mon 16 Sep, 20:44
Terminal
File Edit View Search Terminal Help
src:x:40:
gnats:x:41:
shadow:x:42:
utmp:x:43:
video:x:44:
sasl:x:45:
plugdev:x:46:sujal
staff:x:50:
games:x:60:
users:x:100:
nogroup:x:65534:
systemd-journal:x:101:
systemd-network:x:102:
systemd-resolve:x:103:
input:x:104:
crontab:x:105:
syslog:x:106:
messagebus:x:107:
netdev:x:108:
mlocate:x:109:
ssl-cert:x:110:
uuidd:x:111:
avahi-autoipd:x:112:
bluetooth:x:113:
rtkit:x:114:
ssh:x:115:
lpadmin:x:116:sujal
whoopsie:x:117:



Fig(44): Listing User and Group IDs

Creating and Setting Permissions for a Directory

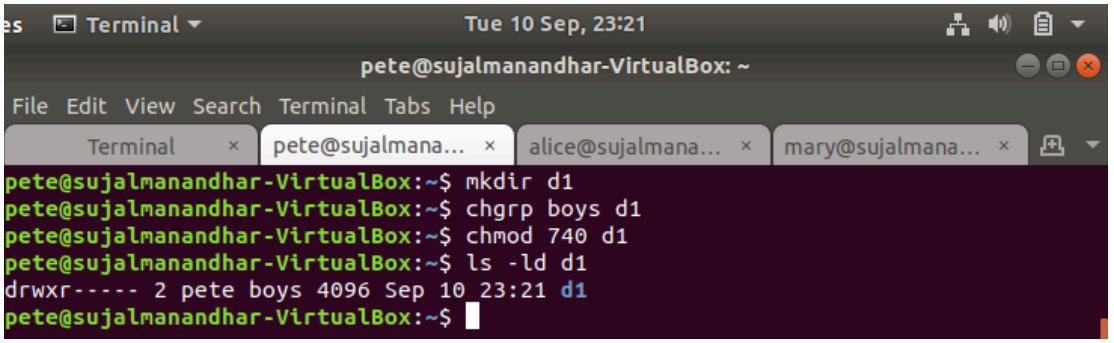
First, we need to switch to **"pete"** user. Then, create a new directory named **"d1"**.
And, change the ownership of the directory **"d1"** to the **"boys"** group.
Again, set the permissions for the directory **"d1"** to **"740"**, meaning the owner has full permissions, the group has read-only permissions, and others have no permissions.
Finally, list the details of **"d1"**, showing its permissions, ownership and group members.

Steps Involved:

1. **Switch to Pete's user**: "su - pete"
2. **Create a new directory**: "mkdir d1"
3. **Change ownership of the directory to the "boys" group**: sudo chown :boys d1
4. **Set permissions to "740":** "chmod 740 d1"
    - **Explanation**: The owner has full permissions, the group has read-only permissions, and others have no permissions.

Outcome:

- Pete successfully owns the directory **"d1"**, and it is shared with the **"boys"** group.



Fig(45): Pete's Home Directory and d1

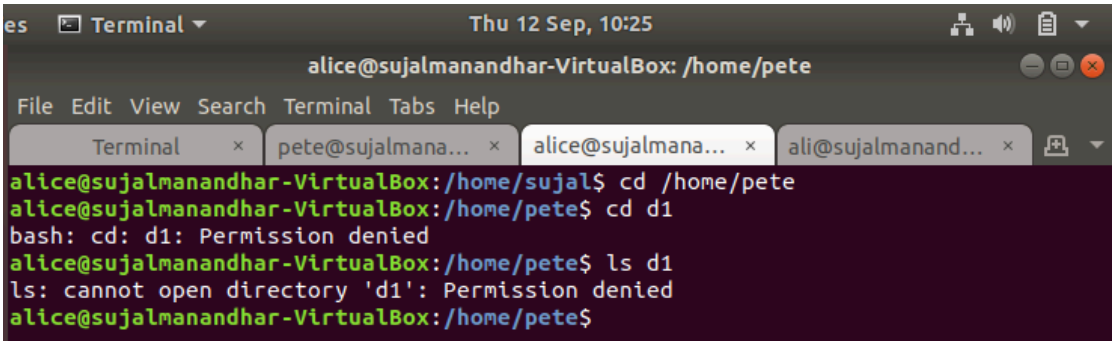
The above figure shows that **"d1"** is owned by **"pete"**, and belongs to the **"boys"** group.

Alice's Access Denied to "d1" Directory

Explanation: Since Alice is part of the "girls" group and not "boys", she is denied access to **Pete's "d1"** directory, which is owned by the "boys" group and restricted by the 740 permission setting.

Command Output:

- Alice attempts to access **"d1"**, but due to insufficient permissions, she is denied entry.



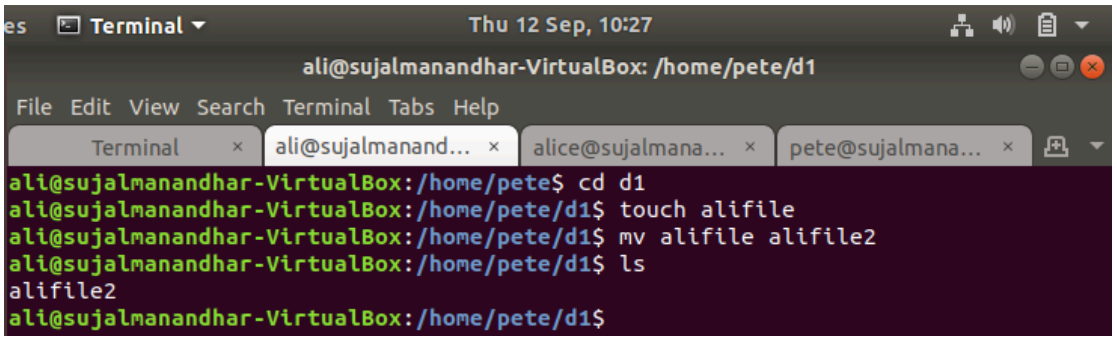
Fig(46): Alice's Permission Denied for "d1" Directory

File Management in Directory "d1" by Ali

Ali accessed the **"d1"** directory, created **"alifile"**, and renamed it to **"alifile2"**, showing proper permissions for file management.

Command:

- **Ali**, as a member of the **"boys"** group, successfully creates a file in **"d1"** and renames it using the following commands:
    1. "touch alifile"
    2. "mv alifile alifile2"



Fig(47): Ali Creating and Renaming a File in d1 Directory

Permission Management for Directories

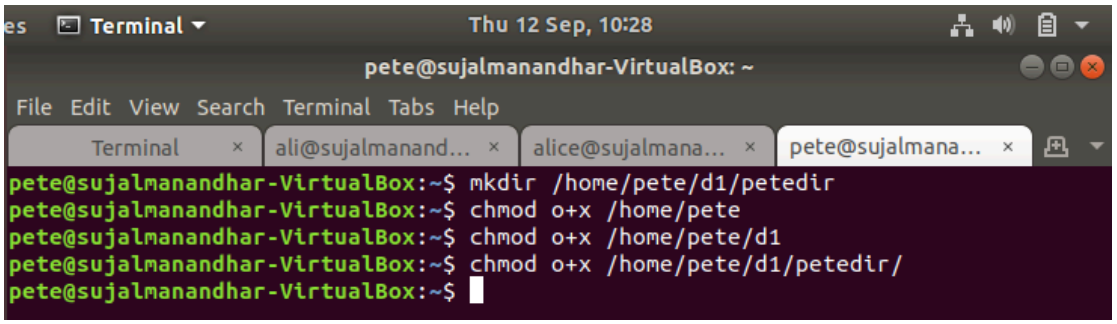
Here, **Pete** created the **"petedir"** directory and updated permissions for the **"/home/pete"**, **"/home/pete/d1"**, and **"/home/pete/d1/petedir"** directories, allowing others to access them by adding execute permissions.

1. **Purpose: Pete** updates permissions for directories to allow other users access while maintaining secure control over his files.
2. **Commands:**

- **Create a directory: "**mkdir petedir"
- **Update permissions**:
  - "chmod 755 /home/pete"
  - "chmod 750 /home/pete/d1"
  - "chmod 700 /home/pete/d1/petedir"

3. **Outcome:**

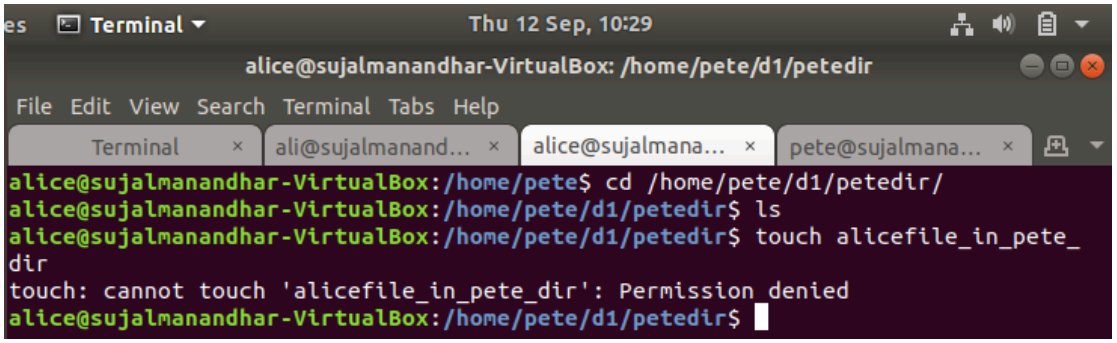
- The permissions allow read-only access to the home directory and controlled access to **"d1"** and **"petedir"**.



Fig(48): Directory "petedir" Permissions Setup

Permission Denied for File Creation

Alice attempted to create a file in **Pete's "petedir"**, but the action was denied due to insufficient permissions for writing in that directory.



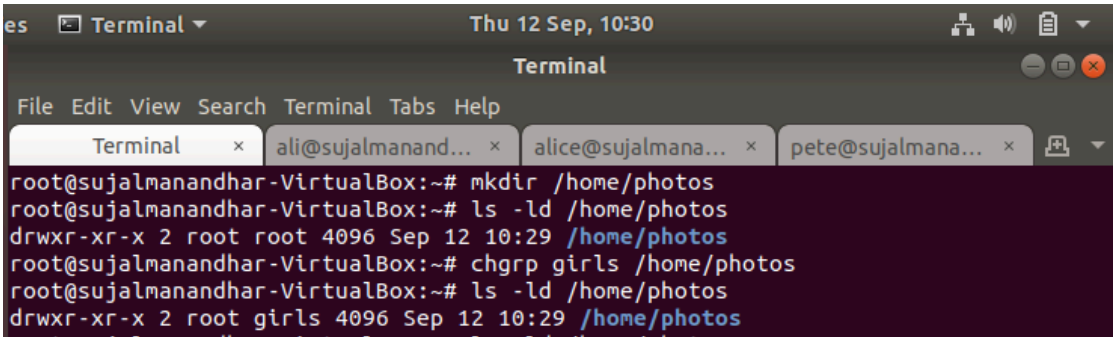
Fig(49): Attempt to Create a File in "petedir" by Alice

Changing Group Ownership of a Directory

The **"/home/photos"** directory was initially owned by the **"root"** group. After changing its group ownership to **"girls"** using **"chgrp"**, the directory's group now reflects the new group assignment.

Command: "sudo chgrp girls /home/photos"

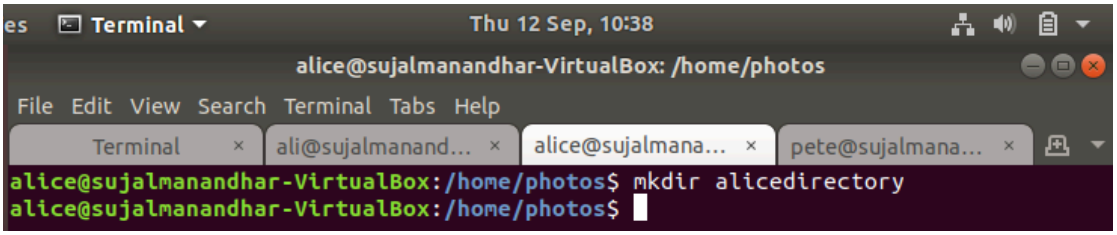
- **Purpose:** Changes the group ownership of the directory **"/home/photos"** from **"root"** to the **"girls"** group, allowing **Alice** and **Mary** to access the directory.



Fig(50): Directory Ownership Update "/home/photos"

Creating a Directory in a Group-Owned Directory

Alice successfully created a directory named **"alicedirectory"** within the **"/home/photos"** directory, which is owned by the **"girls"** group.
Command: "mkdir alicedirectory"



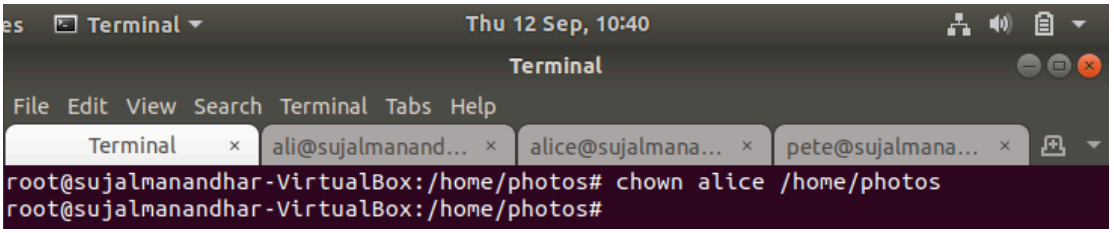
Fig(51): Alice created "alicedirectory" in "/home/photos"

Changing Ownership of a Directory

The ownership of the **"/home/photos"** directory changed to **"Alice"**.

Command: "sudo chown alice /home/photos"

- **Purpose:** Transfers the ownership of the **"/home/photos"** directory to **Alice**, making her the sole owner.



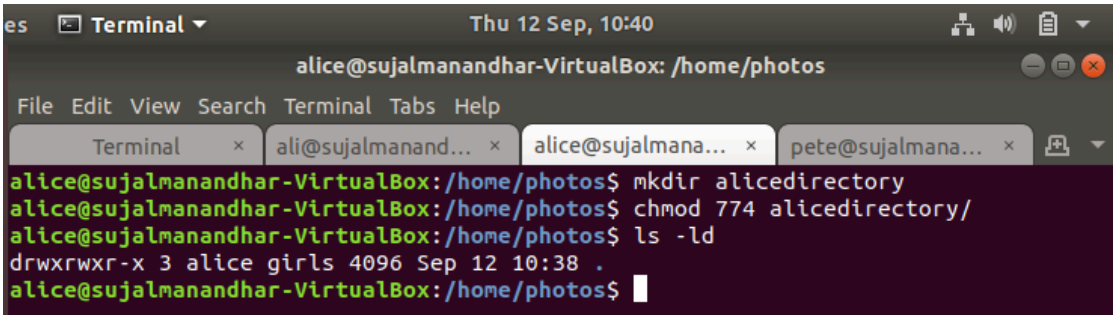
Fig(52): Changing Ownership of "/home/photos" to "Alice"

Setting Permissions of a New Directory

The directory **"alicedirectory"** was created by **Alice**, and permissions were set as follows:

Command: "chmod 744 alicedirectory"

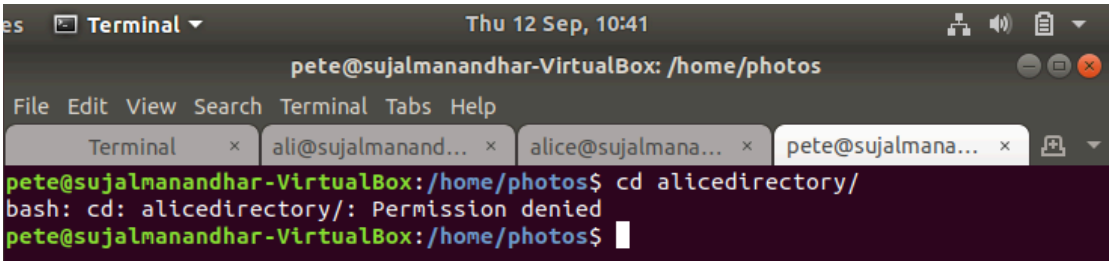
- **Explanation:** Grants full access (read, write, execute) to the owner, read-only access to the group, and read-only access to others.



Fig(53): Directory Permissions for "alicedirectory"

Access Permissions Denied for Directory

Pete is denied access to **"alicedirectory"** because the permissions (744) restrict access to the owner and group members, excluding others.



Fig(54): Permission Denied for Pete to access "alicedirectory"

Conclusion

This lab provided hands-on experience with Linux Discretionary Access Control (DAC), demonstrating how to manage users, groups, and permissions for directories and files. The exercise illustrated the importance of correct permission settings for securing resources while allowing appropriate access levels for specific user groups.

Password Cracking (Lab 6)

Introduction

Lab 6 focused on the process of password cracking, where we compiled and ran a C program aimed at breaking password hashes. Through the execution of this program and the analysis of the resulting hashes, we gained an understanding of how attackers leverage weak passwords. This exercise highlighted the significance of implementing robust password policies, encryption techniques, and secure hash functions to prevent unauthorised access**.**

Creating and Compiling a Password Cracking Program

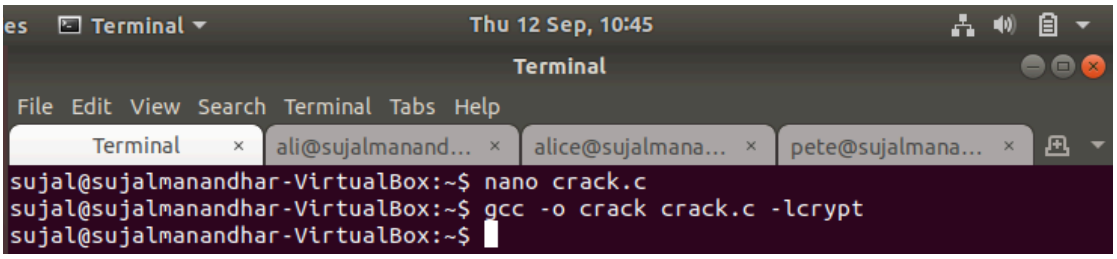
First, we open the **"crack.c"** file in the Nano editor where we can write the C code for cracking password hashes.
Next, we compile the **"crack.c"** file, linking the cryptographic library with **"-lcrypt"**, and create an executable file named crack.

Steps:

1. **Open the "crack.c" file**:
    - **Command:** "nano crack.c"

- **Purpose**: Write the C code for cracking password hashes in the Nano text editor.

2. **Compile the program**:
   - **Command: "**gcc -o crack crack.c -lcrypt"
   - **Explanation:** This command compiles the **"crack.c"** file and links it with the cryptographic library (**"-lcrypt"**), creating an executable file named **"crack'**.



Fig(55): Password Cracker Compilation

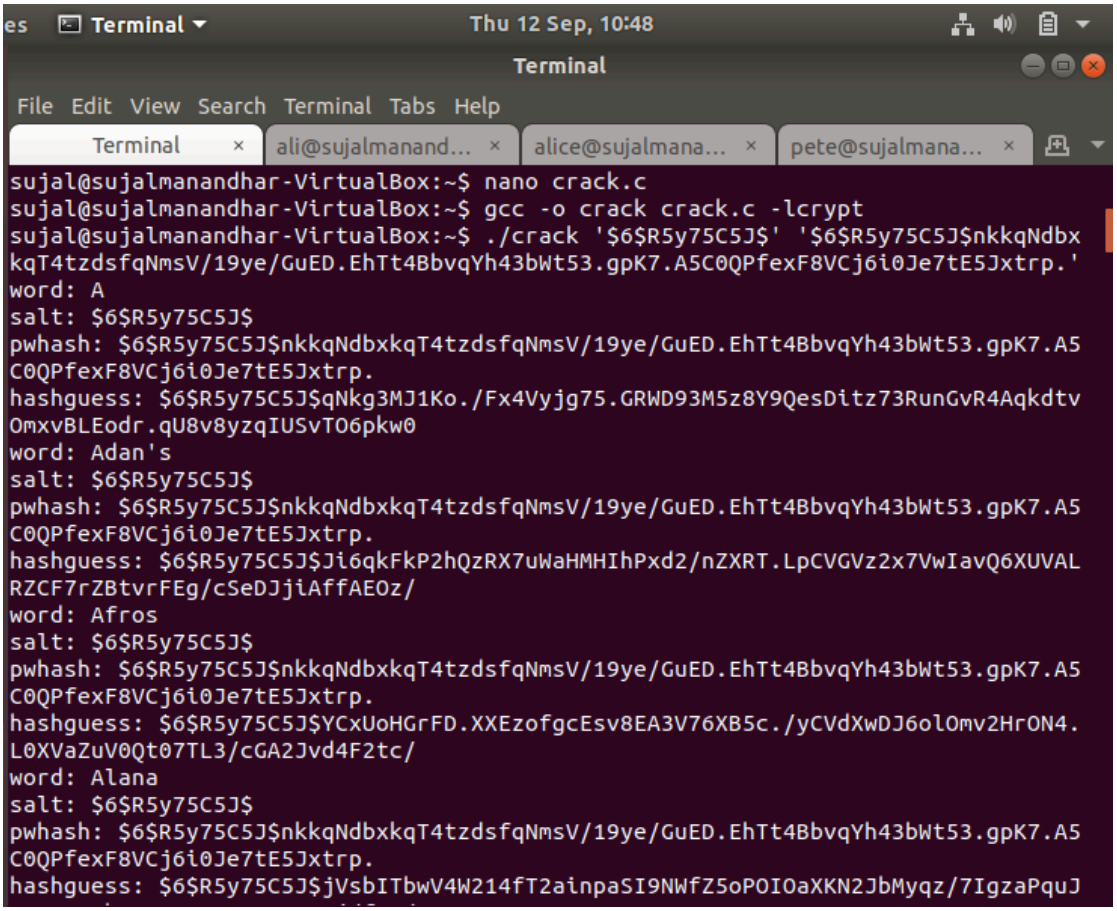
Running the Password Cracking Program

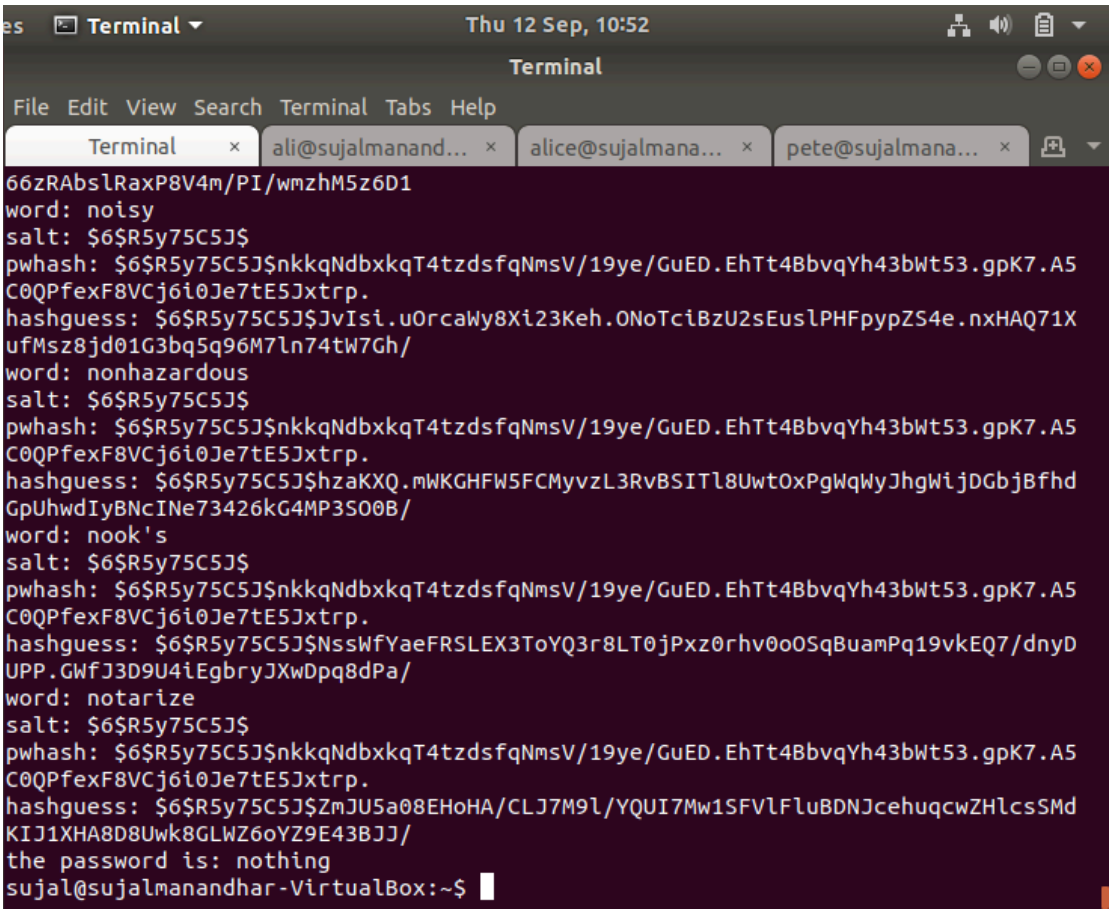
After compiling the program, we execute the cracking process with the following command:

```
sujal@sujalmanandhar-VirtualBox:~$ ./crack '$6$R5y75C5J$' '$6$R5y75C5J$nkkqNdbx
kqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5C0QPfexF8VCj6i0Je7tE5Jxtrp.'
```

This command runs the executable crack, passing two arguments: a salt ("6R5y75C5J$") and a hashed password ("6R5y75C5J$nkkqNdbx…").

The program then tries various word combinations (such as "A", "Adan's", "Afros", etc.), generating corresponding hash guesses. It compares each hash guess with the target password hash to find the matching word.


es Terminal Thu 12 Sep, 10:48
Terminal
File Edit View Search Terminal Tabs Help
Terminal
ali@sujalmanand...
alice@sujalmana...
pete@sujalmana...
sujal@sujalmanandhar-VirtualBox:~$ nano crack.c
sujal@sujalmanandhar-VirtualBox:~$ gcc -o crack crack.c -lcrypt
sujal@sujalmanandhar-VirtualBox:~$ ./crack '6R5y75C5J$' '$6$R5y75C5J$nkkqNdbx
kqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5C0QPfexF8VCj6i0Je7tE5Jxtrp.'
word: A
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$qNkg3MJ1Ko./Fx4Vyjg75.GRWD93M5z8Y9QesDitz73RunGvR4Aqkdtv
OmxvBLEodr.qU8v8yzqIUSvTO6pkw0
word: Adan's
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$Ji6qkFkP2hQzRX7uWaHMHIhPxd2/nZXRT.LpCVGVz2x7VwIavQ6XUVAL
RZCF7rZBtvrFEg/cSeDJjiAffAEOz/
word: Afros
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$YCxUoHGrFD.XXEzofgcEsv8EA3V76XB5c./yCVdXwDJ6olOmv2HrON4.
L0XVaZuV0Qt07TL3/cGA2Jvd4F2tc/
word: Alana
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$jVsbITbwV4W214fT2ainpaSI9NWfZ5oPOIOaXKN2JbMyqz/7IgzaPquJ


es Terminal Thu 12 Sep, 10:52
Terminal
File Edit View Search Terminal Tabs Help
Terminal
ali@sujalmanand...
alice@sujalmana...
pete@sujalmana...
66zRAbslRaxP8V4m/PI/wmzhM5z6D1
word: noisy
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$JvIsi.uOrcaWy8Xi23Keh.ONoTciBzU2sEuslPHFpypZS4e.nxHAQ71X
ufMsz8jd01G3bq5q96M7ln74tW7Gh/
word: nonhazardous
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$hzaKXQ.mWKGHFW5FCMyvzL3RvBSITl8UwtOxPgWqWyJhgWijDGbjBfhd
GpUhwdIyBNcINe73426kG4MP3SO0B/
word: nook's
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$NssWfYaeFRSLEX3ToYQ3r8LT0jPxz0rhv0oOSqBuamPq19vkEQ7/dnyD
UPP.GWfJ3D9U4iEgbryJXwDpq8dPa/
word: notarize
salt: 6R5y75C5J$
pwhash: 6R5y75C5J$nkkqNdbxkqT4tzdsfqNmsV/19ye/GuED.EhTt4BbvqYh43bWt53.gpK7.A5
C0QPfexF8VCj6i0Je7tE5Jxtrp.
hashguess: 6R5y75C5J$ZmJU5a08EHoHA/CLJ7M9l/YQUI7Mw1SFVlFluBDNJcehuqcwZHlcsSMd
KIJ1XHA8D8Uwk8GLWZ6oYZ9E43BJJ/
the password is: nothing
sujal@sujalmanandhar-VirtualBox:~$

Fig(56): Password Cracking Execution and Output

The output shows various guesses and their hashes. The password **"nothing"** was identified as correct because its hash matched the target hash provided to the cracking program. The program found this match by testing multiple guesses until it produced a matching hash.

Checking for the Password in Dictionary

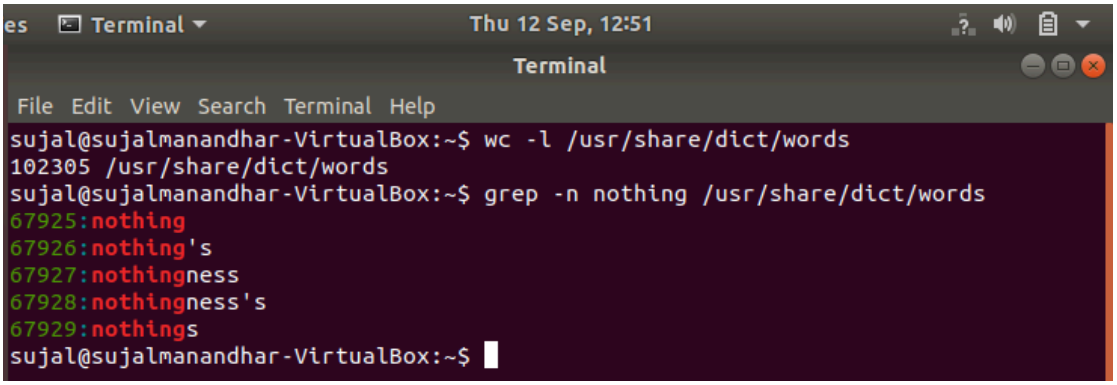
Purpose: We verify whether the cracked password (**"nothing"**) exists in the dictionary file to confirm it as a valid guess.

Commands:

1. **Count the number of words in the dictionary**:
   - **Command:** "wc -l /usr/share/dict/words"
   - **Explanation**: This command counts the number of lines in the dictionary file, which contains 102,305 entries.
2. **Search for the word "nothing"**:
   - **Command:** grep -n "nothing" /usr/share/dict/words
   - **Explanation**: This command searches for the word **"nothing"** in the dictionary file and displays its line number and variations (e.g., "nothing's", "nothingness").

Outcome:

- The output shows the variations of the word like "nothing", "nothing's", "nothingness", and others. The presence of **"nothing"** in the dictionary confirms that it was a valid guess among the tested passwords.



Fig(57): Dictionary Search for "nothing"

Adding a New User "Fred"

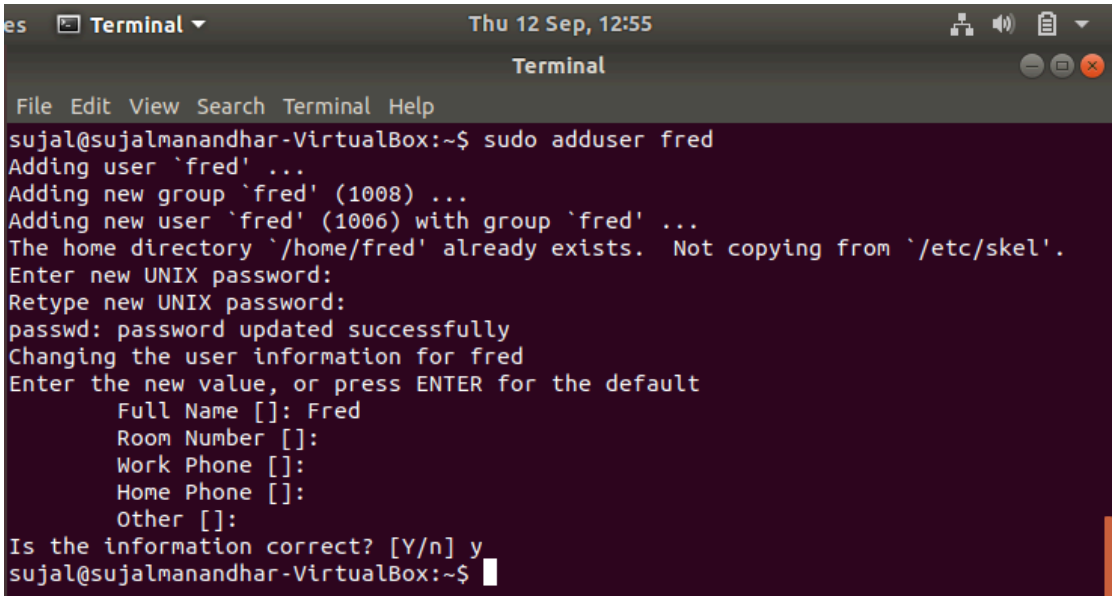
Then, we add a new user named **"fred"**, set up a home directory, update the password, and provide additional user information, completing the process successfully.

Steps:

1. **Add a new user named "fred"**:
    - **Command:** "sudo adduser fred"
    - **Explanation**: This command creates a new user **"fred"** with a home directory and prompts for a password.
2. **Set up additional user information**:
    - The system asks for details such as full name, phone number, etc., which are filled out or skipped as necessary.

Outcome:

- A new user **"fred"** is successfully created, and the system updates the user database with the provided information.



Fig(58): New User "Fred"

Editing the Dictionary File

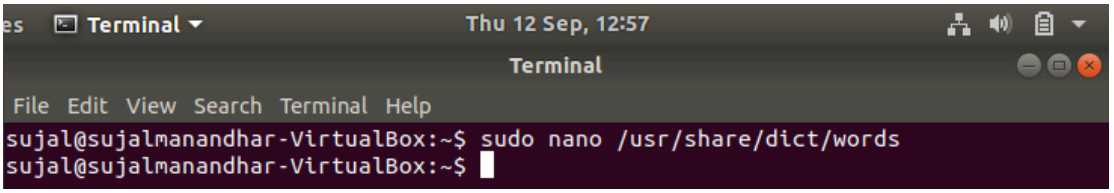
Here, we open the dictionary file in Nano with superuser privileges to edit or review the contents, which can be useful for adding or modifying password guesses.

Command:

- "sudo nano /usr/share/dict/words"
- **Explanation**: Opens the dictionary file in Nano with root privileges, allowing modifications if needed.

Outcome:

- The dictionary file is accessed for viewing and editing, showing all word entries.



Fig(59): Editing Dictionary File

Running the Cracking Program with Root Privileges

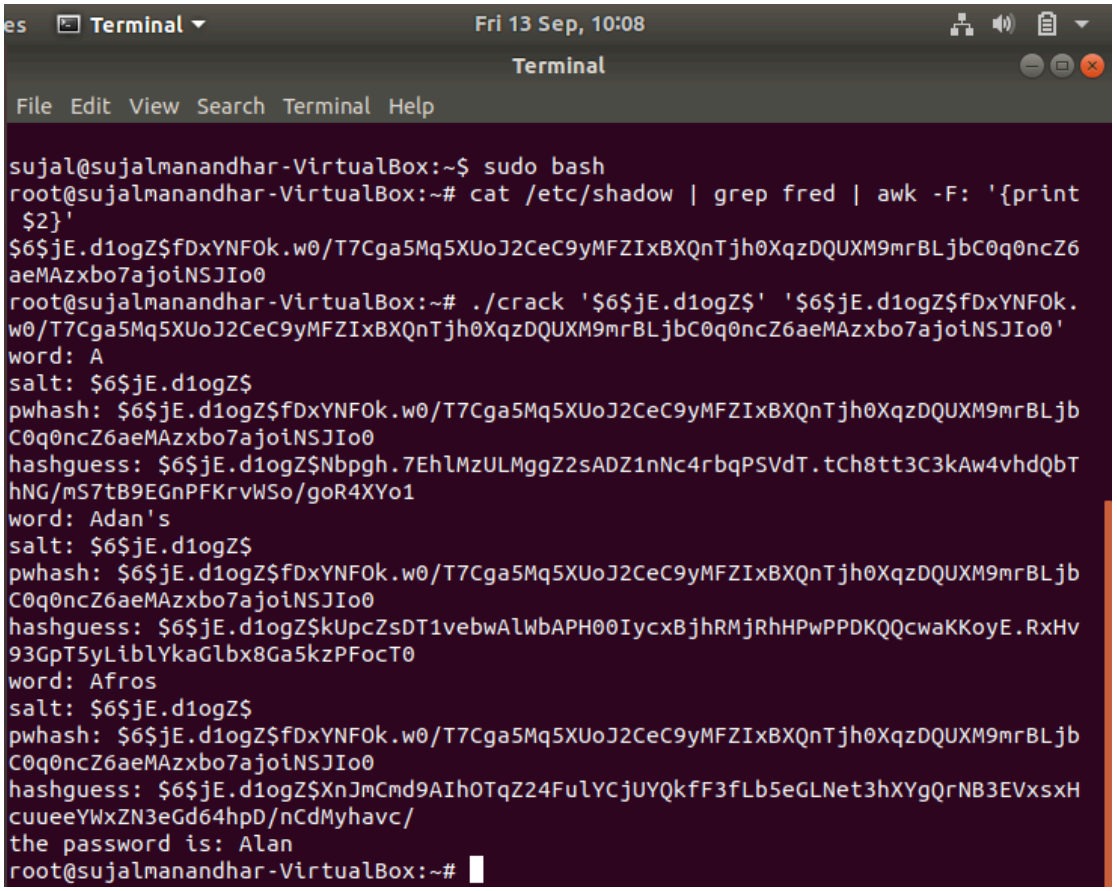
In this section, we perform password cracking as a superuser, which allows us to extract hashed passwords from critical system files.

Steps:

1. **Gain root access:**
   - **Command:** "sudo su"
   - **Explanation**: Switch to the root user to gain administrative privileges for executing system-critical commands.
2. **Extract the hashed password:**
   - **Command:** "sudo cat /etc/shadow | grep fred"
   - **Explanation**: Extracts the hashed password of the user **"fred"** from the **"/etc/shadow"** file, where password hashes are stored for security.
3. **Run the cracking program:**
   - **Command:** "./crack 6salt$hashed_password"
   - **Explanation**: Runs the password cracking program with the extracted salt and hash from the **"/etc/shadow"** file.

Outcome:

- The cracking program tests various password guesses, and the correct password **"Alan"** is found after comparing generated hashes with the target hash.



Fig(60): Password Cracking with Root Access

Conclusion

In this lab, we explored the process of cracking password hashes by writing and compiling a C program that tests various word combinations. We used both a dictionary-based approach and root access to extract hashed passwords from the system's shadow file. The experiment highlighted the vulnerability of weak passwords and the importance of using strong password policies to prevent unauthorised access.

Reflective Report

Cybersecurity and cryptography are captivating subjects, featuring a deep historical background and significant influence on real-world applications, while also presenting intriguing challenges in engineering, theory, and mathematics(Herzberg, 2022). Cryptography is the field focused on studying and implementing secure communication techniques, even when faced with adversaries or malicious actors(Banoth & Regar, 2023). Access control safeguards data by preventing unauthorised subjects from accessing assets and confidential information(Mohamed et al., 2022). Cryptography is fundamental to many of these technologies, enabling impressive functionalities while providing strong assurances of data confidentiality(Balsa et al., 2022). Access control helps to prevent data leaks or unauthorised modifications by potentially harmful entities(Mohamed et al., 2022).

For instance, cryptography helps encrypt data, making it unreadable to unauthorised users, while access control ensures that only specific individuals can access certain data or systems. Securing data effectively requires the use of LWC encryption, especially for sensitive text on low-resource devices(Alkhudaydi & Gutub, 2021). In the labs, we highlighted how cryptography secures data through various methods. For example, OpenSSL was used to establish secure connections, protecting data in transit from eavesdropping. GPG encryption secured files at rest, ensuring only authorised users could access them. Additionally, Linux Discretionary Access Control implemented file permissions to restrict access, further safeguarding sensitive information. Lastly, the password cracking exercise underscored the importance of strong passwords and secure hashing to prevent unauthorised access. Together, these practices demonstrate the vital role of cryptography in protecting data confidentiality and integrity.

Numerous access control models are available, each differing in design, components, policies, and application domains(Penelova, 2021). From the lab exercises, different levels of access were implemented to prevent unauthorised users from compromising data. For instance, in the Linux Discretionary Access Control (DAC) lab, we set specific permissions for files and directories. By configuring a directory owned by the "boys" group to have 740 permissions, we ensured that only the owner had full access, while the group had read-only permissions, and others were completely restricted. This setup prevented users like Alice from accessing sensitive files created by Pete. Similarly, in the GPG lab, we generated keys that allowed only designated users to decrypt messages, ensuring that even if data was intercepted, only authorised users could read it. These examples illustrate how carefully managed access controls can effectively safeguard data from unauthorised access. An access control mechanism can efficiently oversee resource access activities and guarantee that authorised users access information resources under proper circumstances(Qiu et al., 2020).

Methods used to limit access to a resource system or physical site and access control is used to prevent unauthorised access to systems that encrypt(Hamouda, 2020).Cryptography and access control each have their strengths and weaknesses. Cryptography offers strong data protection through encryption, ensuring confidentiality and integrity, but can be complex to

implement and may introduce performance overhead. In contrast, access control is simpler to manage and provides immediate enforcement of user permissions, but it relies on users adhering to security protocols, which can leave data vulnerable if not monitored effectively. A balanced approach that combines both techniques can enhance overall security.

In conclusion, the interplay between cryptography and access control is crucial for creating a robust security framework in organisations. Cryptography ensures data confidentiality and integrity through encryption, while access control restricts unauthorised access to sensitive information. Together, they form a comprehensive defence against potential threats, addressing both the secure transmission and proper management of data. By leveraging the strengths of both techniques, organisations can effectively safeguard their assets, mitigate risks, and enhance overall data protection, ensuring that only authorised users can access and manage critical information. This balanced approach is essential for maintaining security in today's digital landscape.

References

Alkhudaydi, M., & Gutub, A. (2021). Securing data via cryptography and arabic text steganography. *SN Computer Science*, *2*(1), 46.

Balsa, E., Nissenbaum, H., & Park, S. (2022, November). Cryptography, Trust and Privacy: It's Complicated. In *Proceedings of the 2022 Symposium on Computer Science and Law* (pp. 167-179).

Banoth, R., & Regar, R. (2023). An introduction to classical and modern cryptography. In *Classical and modern cryptography for beginners* (pp. 1-21).

Hamouda, B. E. H. H. (2020). Comparative study of different cryptographic algorithms. *Journal of Information Security*, *11*(3), 138-148.

Herzberg, A. (2022). *Applied introduction to cryptography*.

Mohamed, A. K. Y. S., Auer, D., Hofer, D., & Küng, J. (2022). A systematic literature review for authorization and access control: definitions, strategies and models. *International journal of web information systems*, *18*(2/3), 156-180.

Penelova, M. (2021). Access control models. *Cybernetics and Information Technologies*, *21*(4), 77-104.

Qiu, J., Tian, Z., Du, C., Zuo, Q., Su, S., & Fang, B. (2020). A survey on access control in the age of internet of things. *IEEE Internet of Things Journal*, *7*(6), 4682-4696.