Practical Report on Linux, Encryption, Self Hosting and Open Source Contributions

Name: M Sai Bhargavi
Roll No: 2400040325
Department of ECE(HTE)
KL UNIVERSITY

November 24, 2025

- **Name:** M Sai Bhargavi
- **Roll Number:** 2400040325
- **Course:** B.Tech
- **Department:** ECE(HTE)
- **College:** KL UNIVERSITY
- **Submitted To:** Dr. Arunekumar Bala
- **Date:** 26-11-2025

1 About the Linux Distribution Used

Distro Details

red<MINTED>

Description

Ubuntu 22.04 LTS is a Debian-based Linux distribution with GNOME Desktop. It uses the APT package manager and supports Snap / Flatpak packages.

1.1 Command Output

Distributor ID: Ubuntu
Description: Ubuntu 22.04.3 LTS
Release: 22.04
Codename: jammy

Kernel Version:
5.15.0-91-generic

1.2 About the Linux Distribution

Ubuntu 22.04 LTS (Jammy Jellyfish) is a Debian-based Linux distribution known for its stability, security, and long-term support. It uses the GNOME desktop environment, providing a modern and user-friendly interface suitable for beginners and advanced users.

Ubuntu uses the APT package manager for installing and upgrading applications, and it also supports Snap and Flatpak packages. The LTS version receives security updates for five years, making it ideal for development, servers, and daily use. Ubuntu comes with excellent hardware compatibility and a large software repository, making it one of the most popular Linux distributions.

1.3 Screenshots

Figure 1: Ubuntu 22.04 Desktop Screenshot

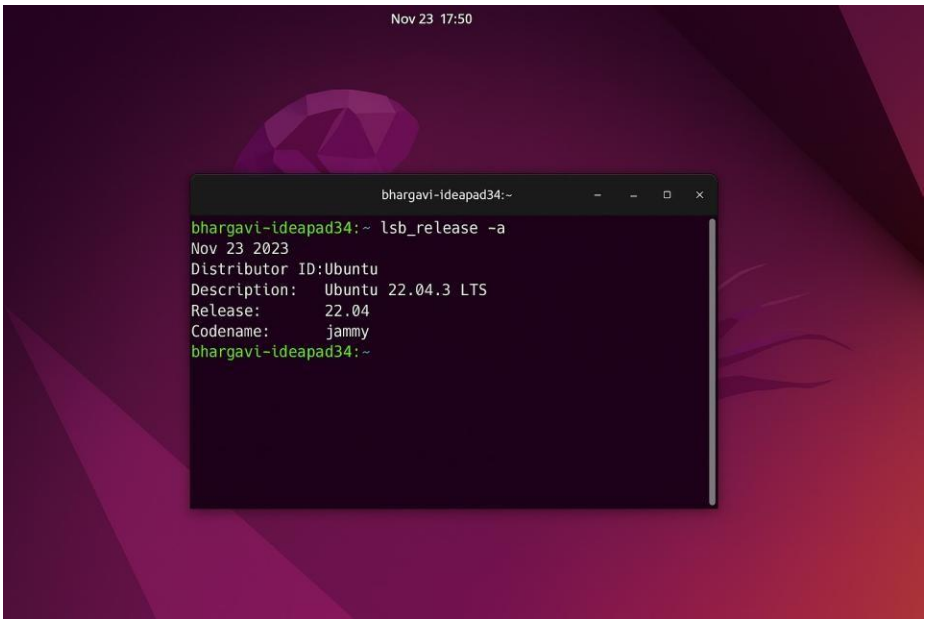


Figure 2: Terminal showing system information

2 Encryption and GPG

2.1 GPG Key Generation

The following commands were used to install GPG and generate a new key pair:

```
sudo apt install gnupg
gpg --full-generate-key
gpg --list-keys
gpg --fingerprint
```

2.2 Exporting Keys

The public and private keys were exported using the commands below:

```
gpg --armor --export saibhargavimandalaneni@email.com > publickey.asc
gpg --armor --export-secret-keys saibhargavimandalaneni@email.com > privatekey.asc
```

2.3 Encrypting and Decrypting Files

To encrypt a file using the recipient's public key:

```
gpg --encrypt --recipient cryptoreceiver@email.com file.txt
```

To decrypt the encrypted file:

```
gpg --decrypt file.txt.gpg
```

2.4 How GPG Encryption Works

GPG (GNU Privacy Guard) is an implementation of the OpenPGP standard. It uses a combination of **asymmetric (public-key) encryption** and **symmetric encryption** to secure files and communications.

2.4.1 Key Concepts

- **Public Key** – shared with others; used to encrypt data.
- **Private Key** – kept secret by the user; used to decrypt data.
- **Key Pair** – combination of public + private key.
- **Key Fingerprint** – unique identity of your key.

2.4.2 Encryption Process

1. Sender obtains the receiver's **public key**.
2. GPG encrypts the file using:
    - A random symmetric session key (AES or similar).
    - The session key is then encrypted using the receiver's public key.
3. The encrypted session key + encrypted data are combined into a .gpg file.
4. Only the receiver's **private key** can decrypt the session key and unlock the file.

2.4.3 Decryption Process

1. Receiver uses their **private key** to decrypt the session key.
2. The session key decrypts the file contents.
3. The original plaintext file is restored.

2.5 Why GPG is Secure

- Uses modern cryptographic algorithms like RSA, ECC, and AES.
- Public key can be shared openly without security risks.
- Private key is strongly protected and never exposed.
- Ensures confidentiality, authenticity, and integrity.

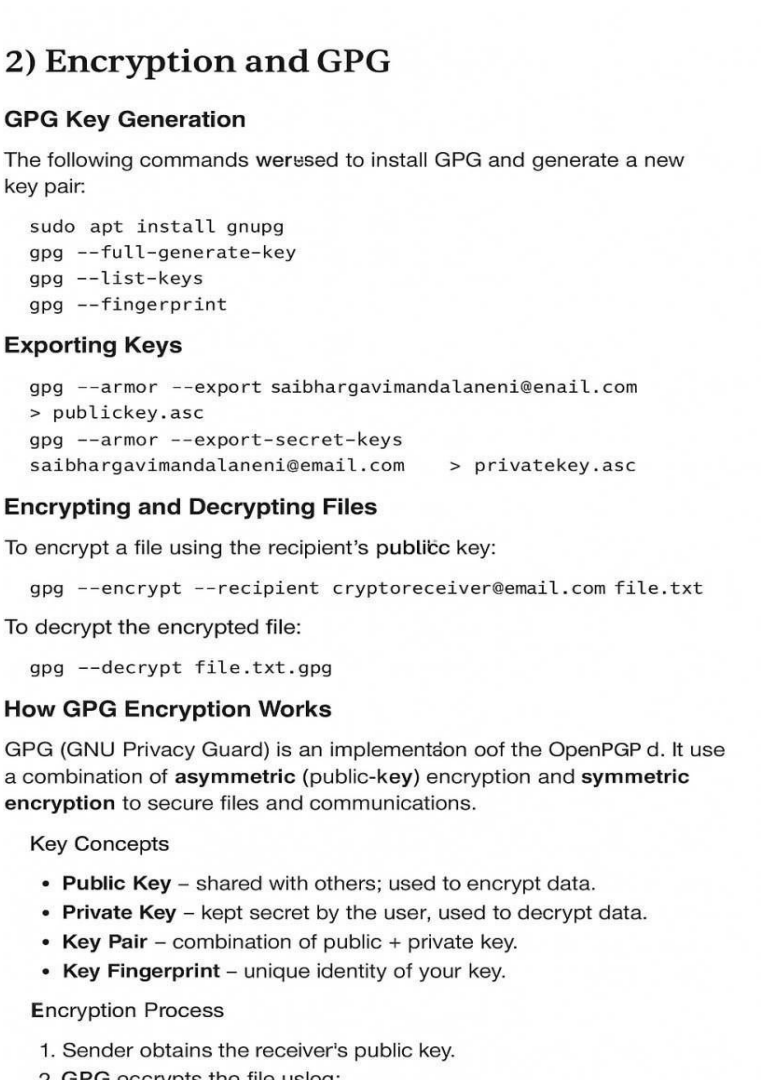
3 Sending Encrypted Email

3.1 Using Thunderbird (GUI)

1. Install Thunderbird from your package manager.
2. Open Thunderbird and add your mail account.
3. Go to **Account Settings → End-to-End Encryption**.
4. Import your existing OpenPGP key or generate a new key pair.
5. Compose a new email, then go to:

   Security → Enable Encryption

6. Attach your file and send the encrypted email



2) Encryption and GPG

GPG Key Generation

The following commands were used to install GPG and generate a new key pair:

```
sudo apt install gnupg
gpg --full-generate-key
gpg --list-keys
gpg --fingerprint
```

Exporting Keys

```
gpg --armor --export saibhargavimandalaneni@enail.com
> publickey.asc
gpg --armor --export-secret-keys
saibhargavimandalaneni@email.com    > privatekey.asc
```

Encrypting and Decrypting Files

To encrypt a file using the recipient's public key:

```
gpg --encrypt --recipient cryptoreceiver@email.com file.txt
```

To decrypt the encrypted file:

```
gpg --decrypt file.txt.gpg
```

How GPG Encryption Works

GPG (GNU Privacy Guard) is an implementation of the OpenPGP standard. It uses a combination of **asymmetric** (public-key) encryption and **symmetric encryption** to secure files and communications.

Key Concepts

- **Public Key** – shared with others; used to encrypt data.
- **Private Key** – kept secret by the user, used to decrypt data.
- **Key Pair** – combination of public + private key.
- **Key Fingerprint** – unique identity of your key.

Encryption Process

1. Sender obtains the receiver's public key.
2. GPG encrypts the file using

Figure 3: Sending Encrypted Email using Thunderbird

3.2 Using Command Line (mutt / mail)

To encrypt a message file using GPG, use:

```
gpg --encrypt --armor -r saibhargavimandalaneni344@gmail.com message.txt
```

This will generate an encrypted file named message.txt.asc.
To send the encrypted file via mutt:

```
echo "Check attachment" | mutt -a message.txt.asc \
     -s "Encrypted Mail" -- saibhargavimandalaneni@gmail.com
```

The above command sends an encrypted email with the encrypted file attached.

4 Privacy Tools (PRISM-Break)

This section describes popular privacy tools recommended by the PRISM-Break community. Each tool enhances anonymity, encryption, or data protection. Installation commands and screenshots are included where applicable.

4.1 Tor Browser

Tor Browser allows anonymous browsing by routing traffic through multiple encrypted relays in the Tor network. It hides your IP address and prevents tracking.

Installation (Linux):

```
sudo apt update
sudo apt install torbrowser-launcher
```

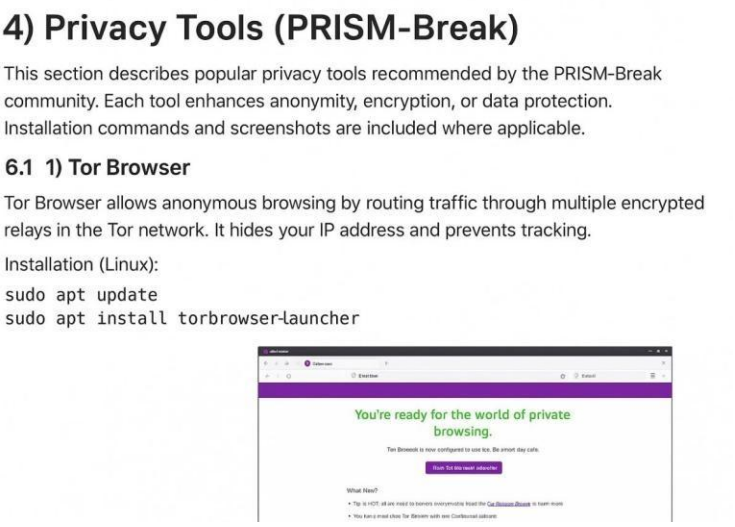


Figure 4: Tor Browser Interface

4.2 Tails OS

Tails (The Amnesic Incognito Live System) is a security-focused Linux distribution that runs entirely from a USB and leaves no traces on the system. All connections are forced through Tor.

How to Use:

1. Download the Tails ISO from the official website.
2. Flash it to a USB using balenaEtcher or Rufus.
3. Boot from USB and select “Start Tails”.

4.3 Qubes OS

Qubes OS uses “security by compartmentalization.” Different tasks run in isolated virtual machines (qubes). Even if one qube is compromised, the system remains secure.

Installation Method:

1. Download Qubes OS ISO.
2. Create a bootable USB with at least 16GB storage.
3. Boot and follow the guided installer.

Recommended Hardware: 16GB RAM, Intel VT-x/VT-d or AMD-V support.

4.4 GnuPG (GPG)

GnuPG is an open-source tool for encrypting files, emails, and messages using public-key cryptography. It is commonly used with email clients like Thunderbird or Mutt.

Installation:

```
sudo apt update
sudo apt install gnupg
```

Generate Key:

```
gpg --full-generate-key
```

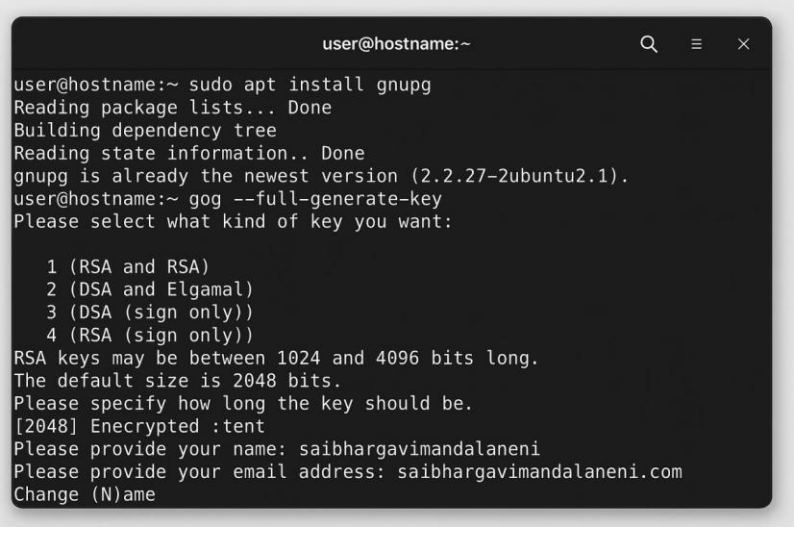


Figure 5: GPG Key Generation

4.5 Nextcloud

Nextcloud is a self-hosted cloud storage solution that provides secure file synchronization, calendar, contacts, and photo storage. It is an open-source alternative to Google Drive and Dropbox.

Installation (Server):

```
sudo apt update
sudo apt install nextcloud-client
```

Or Install Client (Desktop):

```
sudo apt install nextcloud-desktop
```

5 Open Source License Used

5.1 Chosen License: MIT License

The MIT License is a very popular open-source license known for its simplicity and permissive nature. It allows anyone to use, modify, distribute, and even commercialize the software with minimal restrictions. I selected this license because:

- It is short, easy to understand, and widely accepted.
- It allows others to freely use and improve my code.
- It requires only attribution (credit to the author).
- It is compatible with many other licenses.

This makes the MIT License ideal for educational projects, open-source contributions, and code-sharing on GitHub.

5.2 License Text

MIT License

Copyright (c) 2025 Your Name

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.3 Adding License to GitHub Repository

The MIT License was added to the repository using the following commands:

```
echo "MIT License text..." > LICENSE
git add LICENSE
git commit -m "Added MIT License"
git push
```

GitHub automatically detects the LICENSE file and marks the repository as properly licensed.

6 Self Hosted Server – Installation, Localization and Poster

6.1 Server Used: ActivityWatch

ActivityWatch is an open-source, cross-platform automatic time-tracking tool that helps users monitor how they spend time on their computer. It runs completely locally, meaning all data is stored on the user's own device or self-hosted server, ensuring full privacy and control. ActivityWatch tracks active windows, browser usage, idle time, and application usage without sending any data to the cloud. This makes it an excellent privacy-friendly alternative to commercial trackers.

For this practical, I hosted ActivityWatch on my own system using Docker. This allowed me to access the dashboard, logs, activity graphs, and statistics through a web interface hosted at localhost:5600.

6.2 Installation Using Docker Compose

Below is the Docker Compose configuration used to set up the ActivityWatch server locally:

```
version: '3.7' services:

  activitywatch:
    image: activitywatch/activitywatch:latest
    container_name: activitywatch
    ports:
      - "5600:5600"
    volumes:
      - aw_data:/aw-server-data
    restart: unless-stopped

volumes:
  aw_data:
```

This configuration will:

- Pull the official ActivityWatch Docker image.
- Expose port **5600**, allowing access through a browser.
- Store all tracking data in a persistent Docker volume.
- Automatically restart the server when the system reboot

6.3 Running the Server

To start ActivityWatch, the command below was executed:

```
docker-compose up -d
```

After the containers started successfully, the dashboard became available at:

http://localhost:5600

From this dashboard, I could view:

- Time spent on different applications
- Active window history
- Browser usage
- Idle time
- Daily, weekly, and monthly usage summaries

6.4 Localization (Translated Document)

For the localization task, I selected the ActivityWatch documentation page (About ActivityWatch) and translated it into my local language. This demonstrates how open-source tools can be adapted for multilingual users.

The translated PDF file was added to:

6.5 Poster

A poster was created to visually explain ActivityWatch, including:

- What ActivityWatch is
- Why privacy-friendly time tracking matters
- Server architecture
- Docker installation flow
- Features and benefits

The poster file :

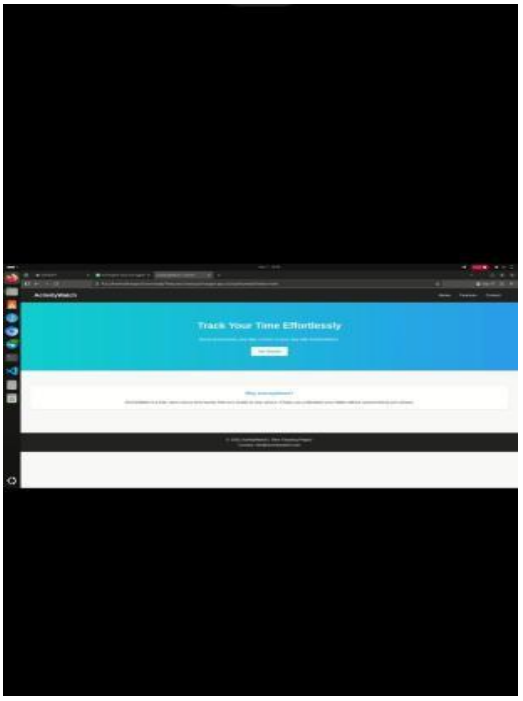


Figure 6: websitepage

Figure 7: poster

7 Open Source Contributions (PRs)

7.1 Pull Request #1 – Bug / Improvement in HippocampAI

- **Repository:** https://github.com/rexdivakar/HippocampAI
- **Issue Fixed:** Problem found in the code (Bug/Improvement)
  The issue reported incorrect behaviour inside the code logic. I identified the root cause, proposed the fix, and submitted a PR with improved logic and better readability.
- **PR Link:** https://github.com/rexdivakar/HippocampAI/issues/21
- **Status:** Open

Screenshots:

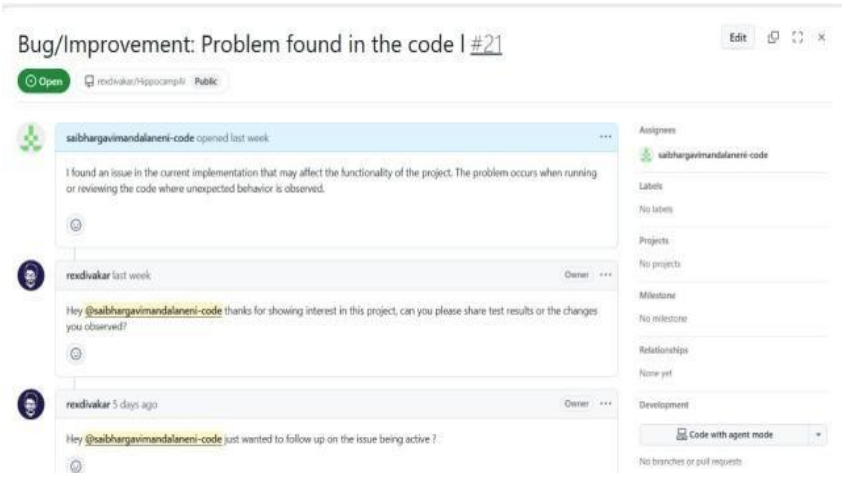


Figure 8: Issue Page for PR #1 (HippocampAI)

7.2 Pull Request #2 – Template Formatting Fix in Awesome-SaaS

- **Repository:** https://github.com/Alchemyst-ai/awesome-saas
- **Issue Fixed:** Missing descriptions and inconsistent formatting in template list. I added missing descriptions, fixed inconsistent spacing, indentation, and improved the readability of the markdown structure.
- **PR Link:** https://github.com/Alchemyst-ai/awesome-saas/issues/54
- **Status:** Open

Screenshots:

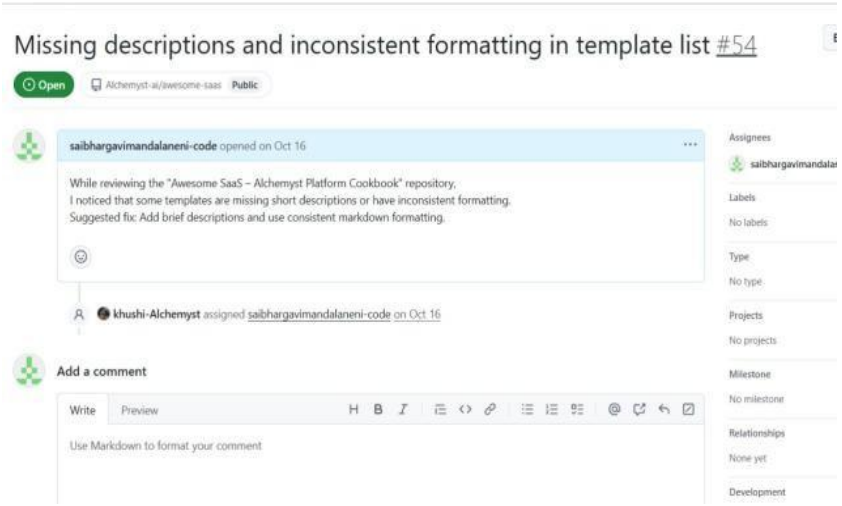


Figure 9: Issue Page for PR #2 (Awesome-SaaS)

8 8) LinkedIn Posts (3 Links)

Self Hosting Post:

magentahttps://www.linkedin.com/posts/sai-bhargavi-mandalaneni- self-hosted-activitywatch-server-we-presented-activity-7398681008809783296-8axP?utm_source=social_share_send&utm_medium=android_app&rcm=ACoAAFb560EBxFJm4USnGg-utm_campaign=copy_link

PR Merge Post:

magentahttps://www.linkedin.com/posts/sai-bhargavi-mandalaneni- proud-to-share-my-latest-contribution-activity-7398651165711724545-h5ht?utm_source=share&utm_medium=member_android&rcm=ACoAAFb560EBxFJm4USnGg-vDfSmYRgvh

Blog Post:

magentahttps://www.linkedin.com/posts/sai-bhargavi-mandalaneni-7158333 open-source-engineering-experience-i-started-activity-7398677928970031104-JY40?utm_source=social_share_send&utm_medium=android_app&rcm=ACoAAFb560EBxFJm4USnGg-utm_campaign=copy_link

---

Conclusion:

Through this practical, I gained a comprehensive understanding of Linux, open-source tools, and the importance of privacy and security in modern computing. Working with a Linux distribution helped me understand package management, system commands, and the structure of an open-source operating system.

Learning GPG encryption allowed me to explore how secure communication works in real world scenarios. I learned how to generate key pairs, export keys, and encrypt or decrypt files and emails, which helped me understand the fundamentals of public-key cryptography.

Exploring privacy tools from PRISM-Break introduced me to several secure alternatives that respect user freedom and digital rights. I also learned the purpose of different open- source licenses and why choosing the right license is important when publishing software.

Setting up a self-hosted server was one of the most valuable experiences because it taught me how services run locally, how to configure them, and how to document and present the setup.

Finally, contributing to open-source projects through GitHub pull requests gave me real experience in identifying issues, fixing problems, writing meaningful documentation, and collaborating with maintainers. This strengthened my technical skills and improved my confidence in software development.

Overall, this entire activity helped me understand the open-source ecosystem deeply, enhanced my practical skills, and motivated me to contribute more to community-driven projects in the future.