Project Evaluation

Project Report
Semester-IV (Batch-2023)

SSH Access Setup Tool

Supervised By:

Dr. Mankirat Kaur

Submitted By:

Aditya Singh Aswal,2310990098(G18)

Anshika Kumari, 2310991612(G18)

Anuj Maurya, 2310991615(G18)

Dhanur Relhan, 2310991647(G18)

Department of Computer Science and Engineering
Chitkara University Institute of Engineering & Technology,
Chitkara University, Punjab

Table of Contents

1. Introduction

1.1 Background and Motivation

In the age of interconnected systems and cloud computing, securing Linux environments has become more essential than ever. Among various entry points into a system, SSH (Secure Shell) stands out as the most widely used protocol for secure remote access and administrative control. However, with great power comes great responsibility—default SSH settings can expose a system to brute-force attacks, unauthorized access, or resource misuse.

This project is driven by the need to establish a secure and efficient remote management setup for a Linux server using OpenSSH. By hardening SSH configurations, we aim to reduce vulnerabilities, enhance administrative control, and promote best practices for Linux system management. This initiative also provides a hands-on learning experience in real-world Linux administration and network security.

1.2 Objectives of the Project

The key objectives of this project are:

- To install, configure, and secure the OpenSSH server on a Linux system.
- To implement **key-based authentication**, replacing traditional password-based logins.
- To **disable root login and password authentication** to minimize unauthorized access risks.
- To apply **idle timeout policies**, session restrictions, and detailed logging for accountability.
- To test all configurations under practical scenarios, ensuring robust and consistent behavior.
- To enable secure LAN access to the system using SSH.

1.3 Scope of the Work

This project focuses specifically on SSH server setup and hardening on a single Linux system within a local network. It includes:

- SSH server installation and service configuration
- Secure key generation and user-specific access management
- Session hardening techniques (e.g., timeout intervals, authentication attempts)
- Basic monitoring and validation of access
- Configuration verification using LAN access and test users

The project does not include broader Linux security measures such as SELinux configuration, advanced firewall setups, or intrusion detection tools, but lays a solid foundation for extending into those areas.

1.4 Report Structure

This report is organized into well-defined sections to provide a clear understanding of the project's development:

- **System Environment** describes the hardware, OS, and tools used.
- **Conceptual Overview** explains the technologies and commands involved.
- **UML Diagrams** illustrate the flow of interactions and activities.
- **Implementation Details** document the configuration steps and scripts.
- **Security and Optimization** highlights the measures taken to improve performance and protection.
- **Testing and Validation** covers test scenarios and troubleshooting methods.
- **Challenges and Limitations** reflect on any obstacles and how they were overcome.
- **Conclusion and Future Work** summarizes achievements and outlines possible enhancements.

The appendices include configuration files, scripts, and other supporting materials.

2. System Environment

2.1 Hardware and Software Requirements

The project was implemented on a standard Linux system suitable for educational and light production use. The following hardware and software configuration was used:

Hardware:

- CPU: Intel Core i5 (or equivalent)
- RAM: 4 GB minimum
- Storage: 20 GB available space
- Network: LAN access enabled (Wi-Fi or Ethernet)

Software:

- Operating System: Ubuntu 22.04 LTS (64-bit)
- OpenSSH Server: Version 8.9p1 or higher
- Shell: Bash 5.1
- Additional Tools: `systemctl`, `sshd`, `scp`, `journalctl`, `ufw` (for optional testing)

This environment provides a balance between performance and simplicity, allowing us to focus on core SSH configuration and security.

2.2 Linux Distribution and Version

The system used for this project was **Ubuntu 22.04 LTS**, a stable, widely-used Linux distribution known for strong community support, active security patching, and compatibility with a range of administration tools. The Long-Term Support (LTS) release ensures ongoing updates and reliability, making it an ideal choice for secure SSH server deployment.

2.3 Tools and Utilities Used

The following tools and utilities were integral to the development and testing of the SSH setup:

- **OpenSSH Server** – Core service enabling remote secure login
- **ssh-keygen** – Used for generating RSA key pairs for secure authentication
- **ssh** – Client tool for connecting to the configured server
- **systemctl** – For managing the SSH service (start, restart, enable)
- **journalctl** – For viewing detailed system logs and troubleshooting SSH behavior
- **chmod, chown, mkdir** – File system tools to set proper SSH directory permissions
- **scp** – For securely copying files between systems during testing
- **ping, ifconfig / ip** – For network checks and host identification

These utilities enabled not only the setup and security of SSH but also ensured effective diagnostics and smooth testing.

3. Conceptual Overview

3.1 Key Concepts Related to the Project

This project is rooted in key Linux administration and cybersecurity principles. A brief overview of the main concepts:

- **SSH (Secure Shell):** A cryptographic network protocol that enables secure remote login and command execution over an unsecured network.
- **Public Key Authentication:** A method of authentication where a private key (kept secure by the user) matches a public key stored on the server, providing stronger security than passwords.
- **SSHD (SSH Daemon):** The background service that listens for and handles incoming SSH connections.
- **Hardening:** The practice of tightening security by disabling unused features, enforcing stricter access controls, and minimizing potential attack vectors.

This project brings these principles together by applying secure SSH configuration best practices in a real-world Linux environment.

3.2 Relevant System Components and Files

Several system components and configuration files were essential to this implementation:

- `/etc/ssh/sshd_config` – Main SSH server configuration file.
- `/home/kascit/.ssh/authorized_keys` – Stores public keys allowed to authenticate as user `kascit`.
- `/var/log/auth.log` – Log file for authentication attempts, useful for tracking SSH activity.
- `/etc/systemd/system/` – Manages SSH service through `systemctl`.
- `~/.ssh/id_rsa`, `~/.ssh/id_rsa.pub` – Private and public SSH key pair generated using `ssh-keygen`.

These files were properly edited, secured, and tested to ensure a working and secure setup.

3.3 Linux Commands and Services Involved

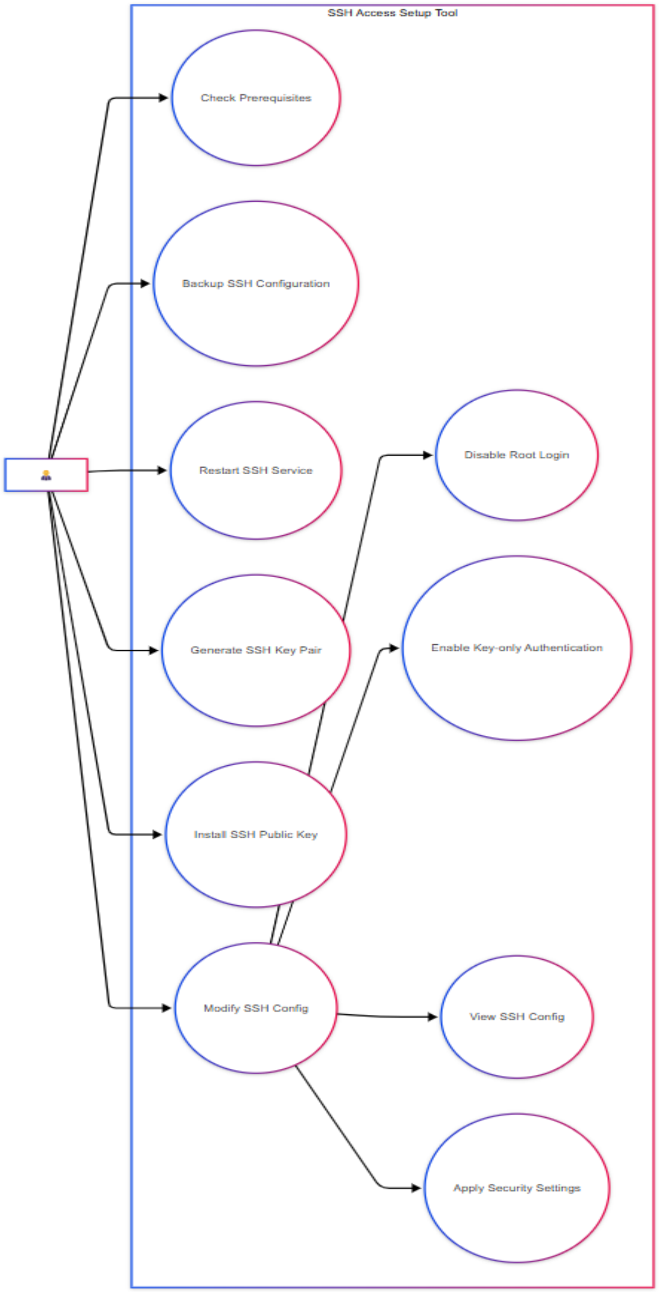
A combination of Linux services and administrative commands were used throughout the project:

- `sudo apt install openssh-server` – Installs the SSH server.
- `sudo systemctl enable --now ssh` – Enables and starts the SSH service immediately.
- `sudo systemctl status ssh` – Checks the status of the SSH daemon.
- `ssh-keygen -t rsa -b 4096` – Generates a strong RSA key pair.
- `ssh -i ~/.ssh/id_rsa kascit@localhost` – Tests login using key-based authentication.
- `chmod 700 ~/.ssh` and `chmod 600 ~/.ssh/authorized_keys` – Sets correct permissions for secure key usage.
- `journalctl -u ssh` – Reviews SSH logs for debugging or auditing.

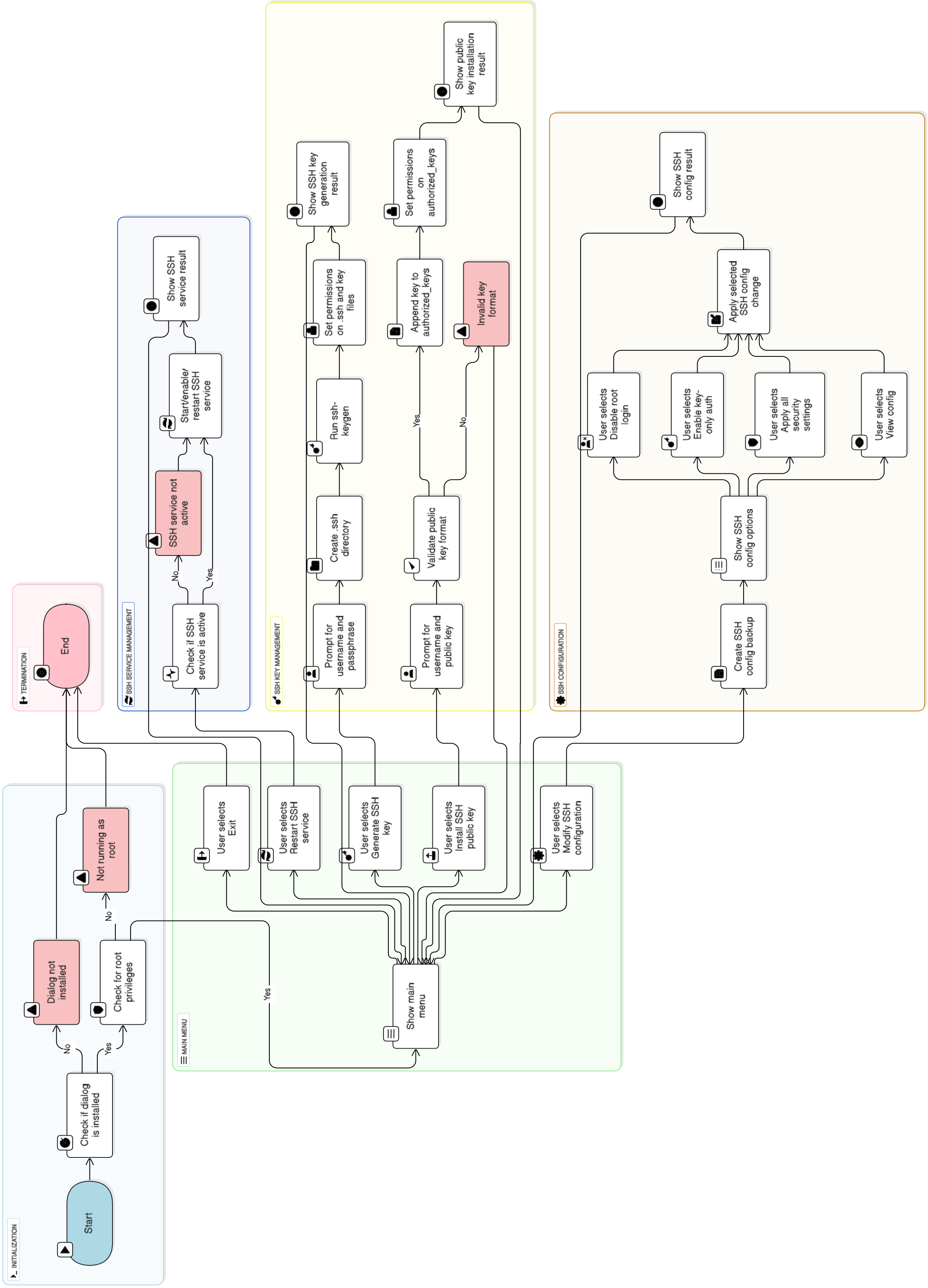
These commands and services were systematically applied and tested to ensure both security and usability.

4. UML Diagrams

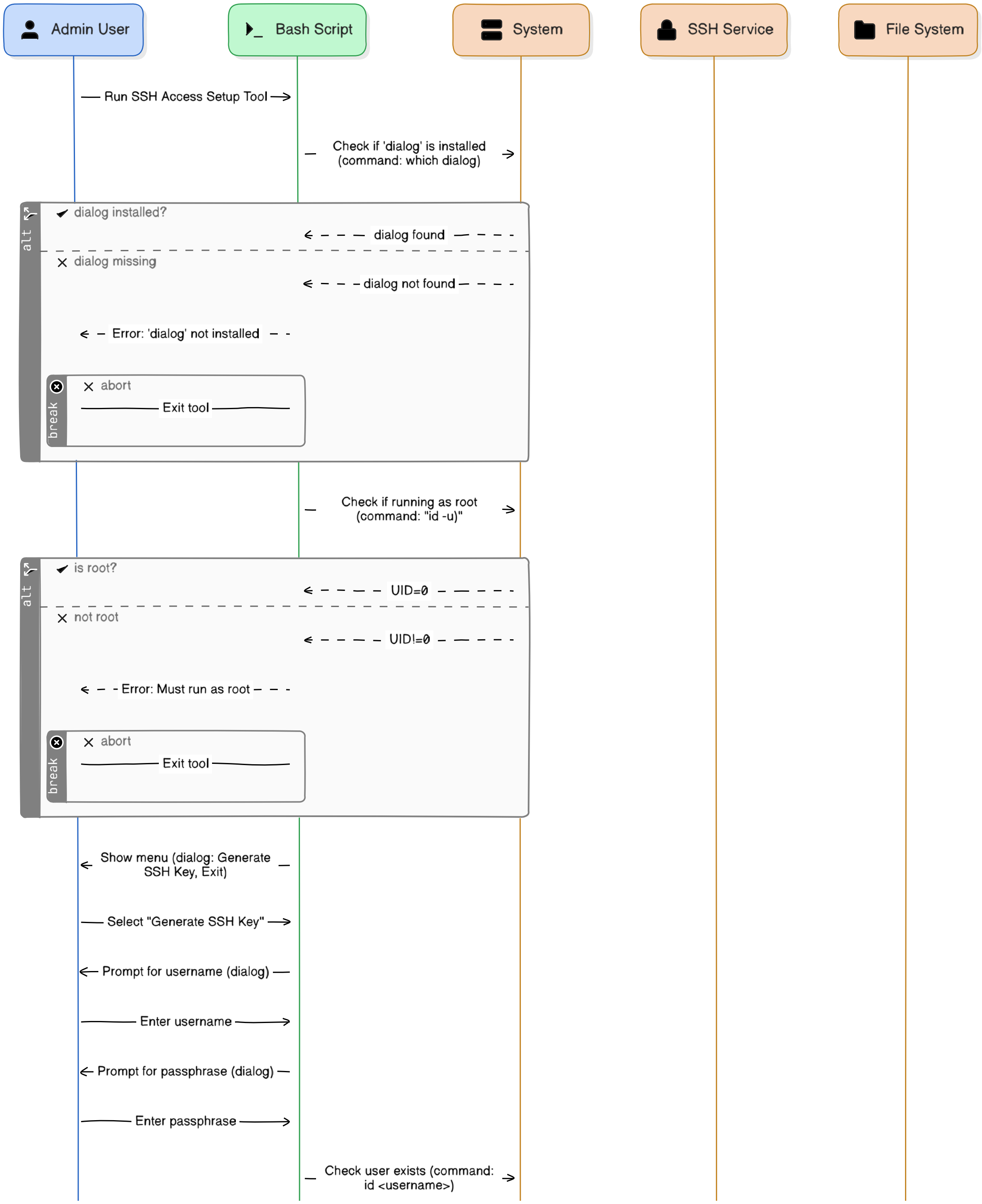
4.1 Use Case Diagram

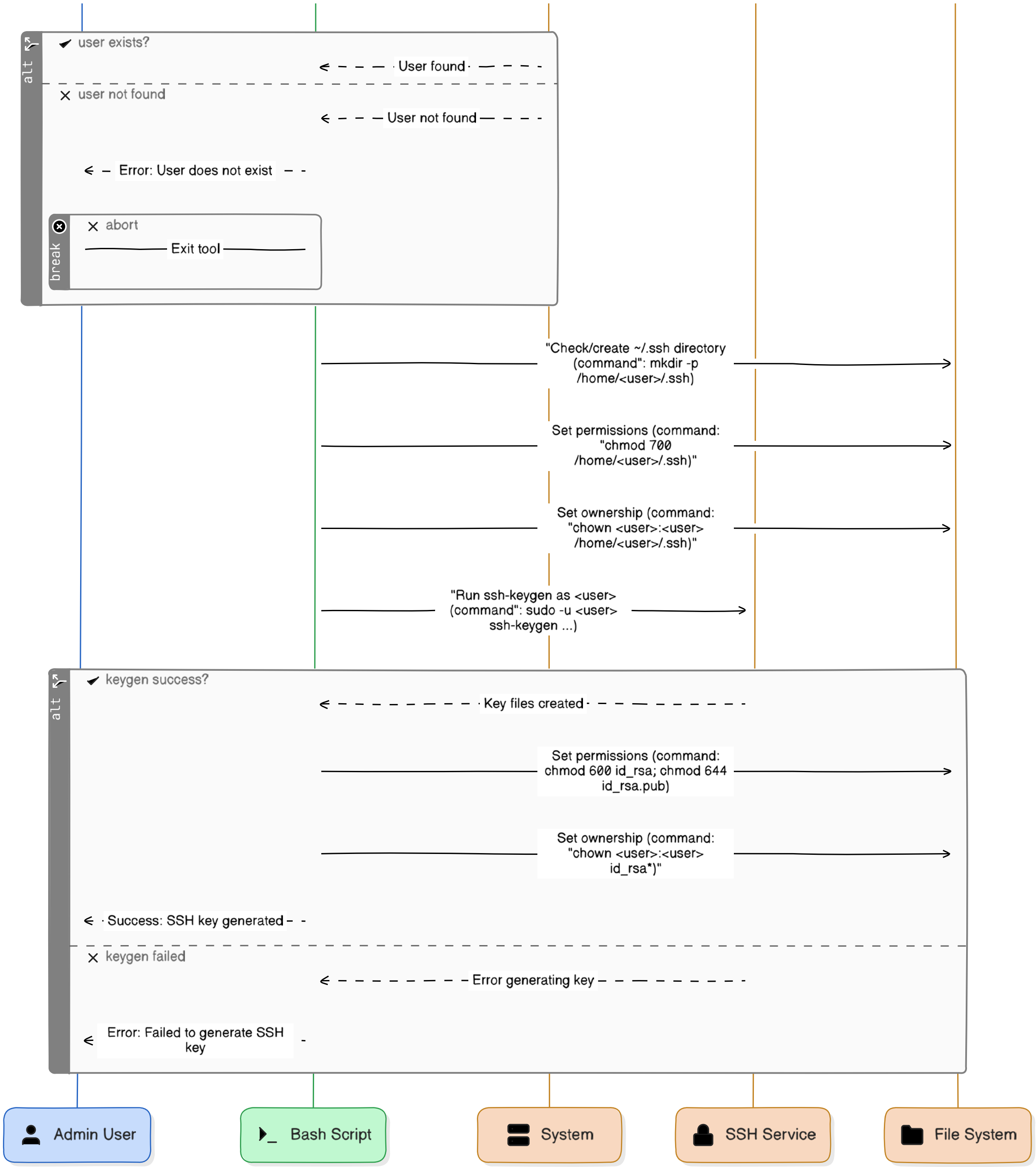


4.2 Activity Diagram


INITIALIZATION
Start
Check if dialog is installed
No
Dialog not installed
Yes
Check for root privileges
No
Not running as root
Yes
TERMINATION
End
MAIN MENU
Show main menu
User selects Exit
User selects Restart SSH service
User selects Generate SSH key
User selects Install SSH public key
User selects Modify SSH configuration
SSH SERVICE MANAGEMENT
Check if SSH service is active
No
SSH service not active
Yes
Start/enable/ restart SSH service
Show SSH service result
SSH KEY MANAGEMENT
Prompt for username and passphrase
Create .ssh directory
Run ssh-keygen
Set permissions on .ssh and key files
Show SSH key generation result
Prompt for username and public key
Validate public key format
Yes
Append key to authorized_keys
Set permissions on authorized_keys
No
Invalid key format
Show public key installation result
SSH CONFIGURATION
Create SSH config backup
Show SSH config options
User selects Disable root login
User selects Enable key-only auth
User selects Apply all security settings
User selects View config
Apply selected SSH config change
Show SSH config result

4.3 Sequence Diagram


Admin User
Bash Script
System
SSH Service
File System
Run SSH Access Setup Tool
Check if 'dialog' is installed (command: which dialog)
alt
dialog installed?
dialog found
dialog missing
dialog not found
Error: 'dialog' not installed
break
abort
Exit tool
Check if running as root (command: "id -u)"
alt
is root?
UID=0
not root
UID!=0
Error: Must run as root
break
abort
Exit tool
Show menu (dialog: Generate SSH Key, Exit)
Select "Generate SSH Key"
Prompt for username (dialog)
Enter username
Prompt for passphrase (dialog)
Enter passphrase
Check user exists (command: id <username>)


alt
user exists?
User found
user not found
User not found
Error: User does not exist
break
abort
Exit tool
"Check/create ~/.ssh directory (command": mkdir -p /home/<user>/.ssh)
Set permissions (command: "chmod 700 /home/<user>/.ssh)"
Set ownership (command: "chown <user>:<user> /home/<user>/.ssh)"
"Run ssh-keygen as <user> (command": sudo -u <user> ssh-keygen ...)
alt
keygen success?
Key files created
Set permissions (command: chmod 600 id_rsa; chmod 644 id_rsa.pub)
Set ownership (command: "chown <user>:<user> id_rsa*)"
Success: SSH key generated
keygen failed
Error generating key
Error: Failed to generate SSH key
Admin User
Bash Script
System
SSH Service
File System

5. Implementation Details

5.1 Step-by-Step Configuration/Development

The project was carried out through a structured series of steps to ensure clarity and control throughout the SSH hardening process:

Step 1: Install OpenSSH Server

The OpenSSH server was installed using the system package manager:

```
sudo apt install openssh-server
```

Step 2: Generate SSH Keys for User `kascit`

Key-based authentication was set up by generating a 4096-bit RSA key:

```
ssh-keygen -t rsa -b 4096 -f /home/kascit/.ssh/id_rsa
```

The public key was then copied to `~/.ssh/authorized_keys`.

Step 3: Secure SSH Directory and Files

To prevent unauthorized access, strict permissions were applied:

- `chmod 700 ~/.ssh`
- `chmod 600 ~/.ssh/authorized_keys`
- Ownership confirmed with: `chown -R kascit:kascit ~/.ssh`

Step 4: Harden SSH Configuration (`sshd_config`)

Key security parameters were configured in `/etc/ssh/sshd_config`:

- `PasswordAuthentication no`
- `PermitRootLogin no`
- `ChallengeResponseAuthentication no`
- `PubkeyAuthentication yes`
- `ClientAliveInterval 300`
- `ClientAliveCountMax 2`
- `MaxAuthTries 3`
- `LoginGraceTime 60`

These reduce brute-force attack risks and close common vulnerabilities.

Step 5: Restart and Test SSH Service

After changes, the service was restarted to apply updates:

```
sudo systemctl restart ssh
```

Testing was done locally using:

```
ssh -i /home/kascit/.ssh/id_rsa kascit@localhost
```

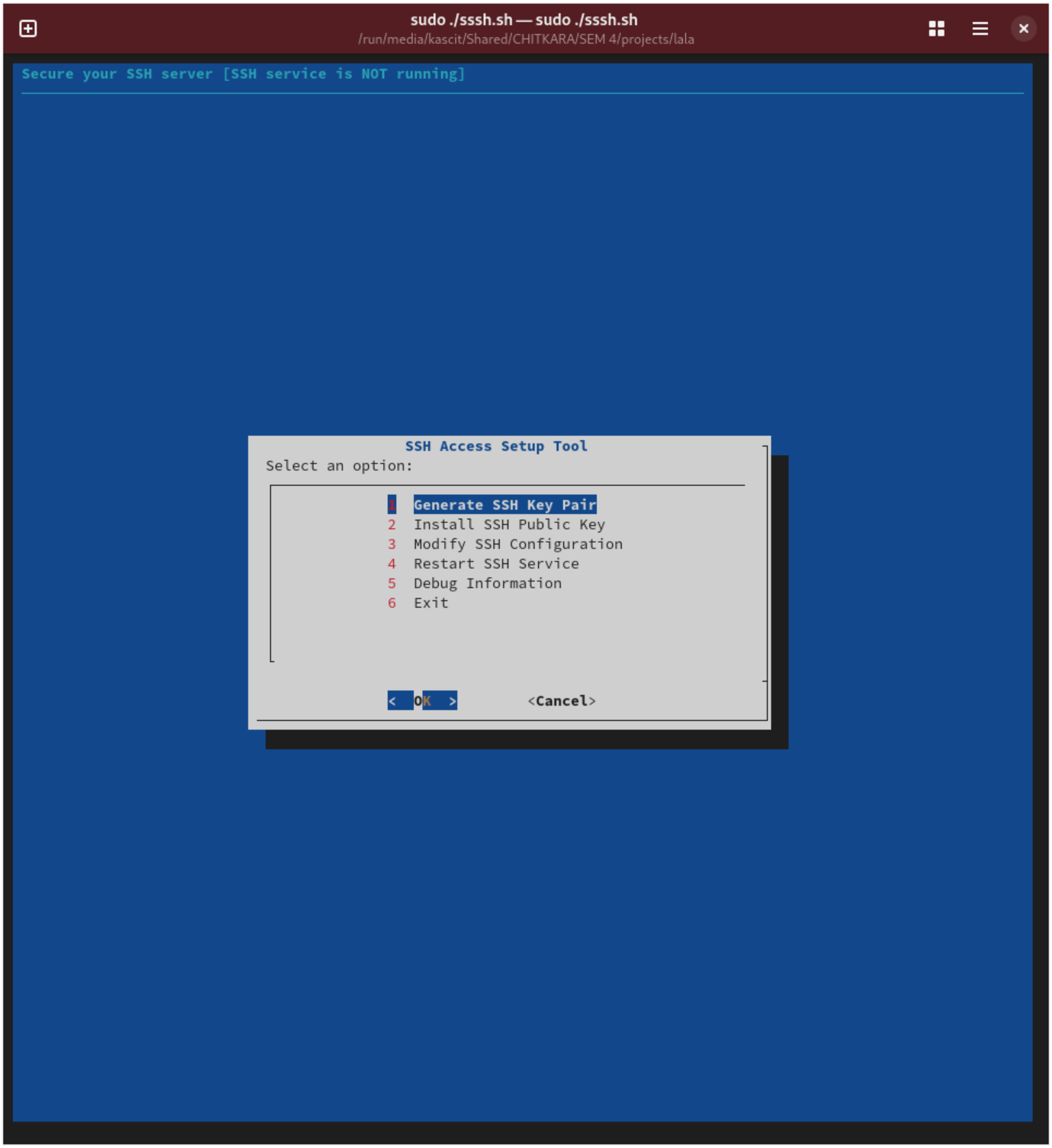
5.2 Commands and Scripts Used

Commands were primarily shell-based, with no external scripts used. However, the project can be easily extended with automation scripts using Bash or Python for future scaling.

Examples:

- `sudo systemctl status ssh` – Validate that SSH is running.
- `ssh -v` – Run SSH in verbose mode for troubleshooting.
- `tail -f /var/log/auth.log` – Live log monitoring of login attempts.

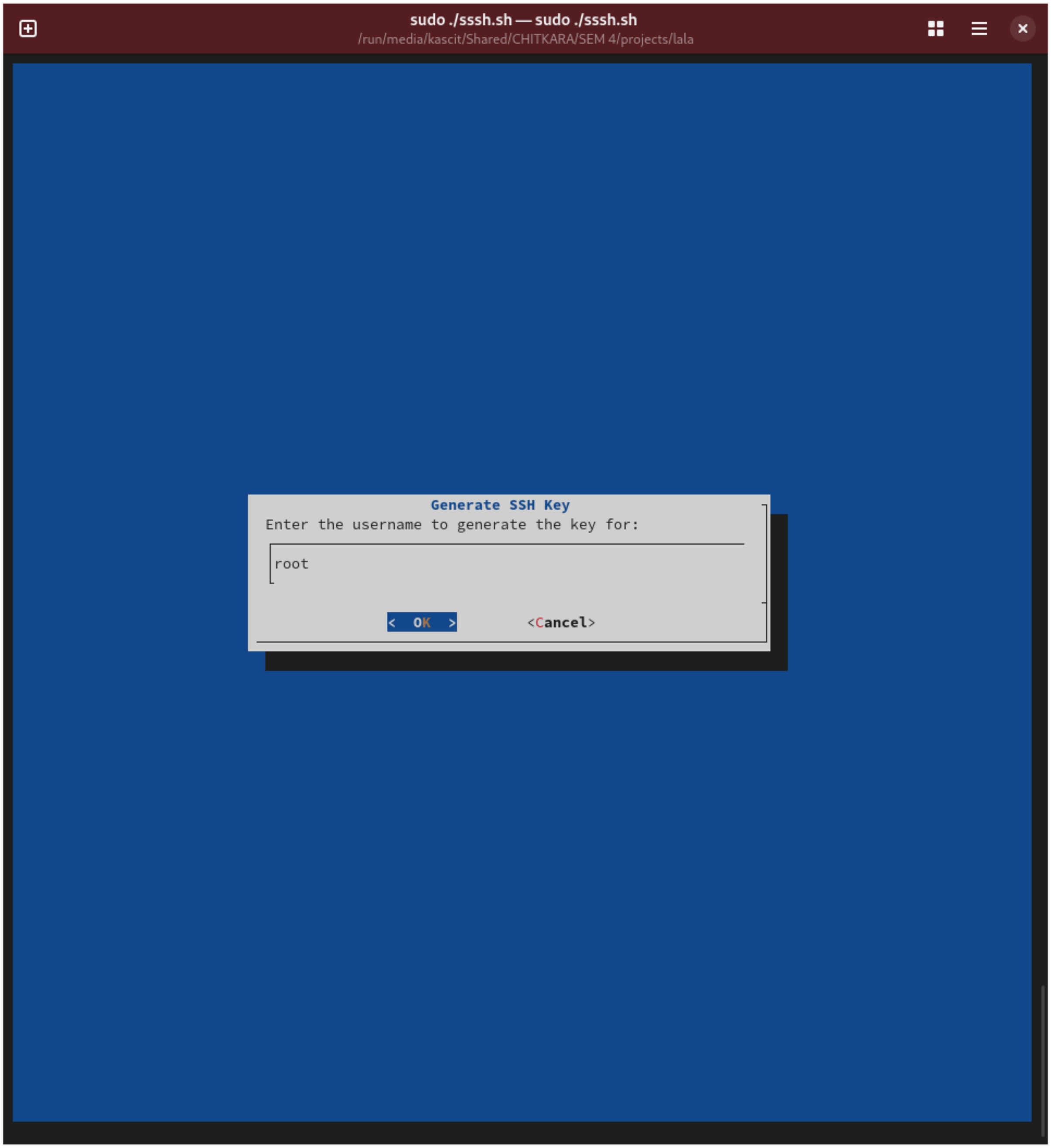
5.3 Screenshots and Outputs



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

```
Secure your SSH server [SSH service is NOT running]

                SSH Access Setup Tool
Select an option:

          1  Generate SSH Key Pair
          2  Install SSH Public Key
          3  Modify SSH Configuration
          4  Restart SSH Service
          5  Debug Information
          6  Exit

          <  OK  >          <Cancel>
```

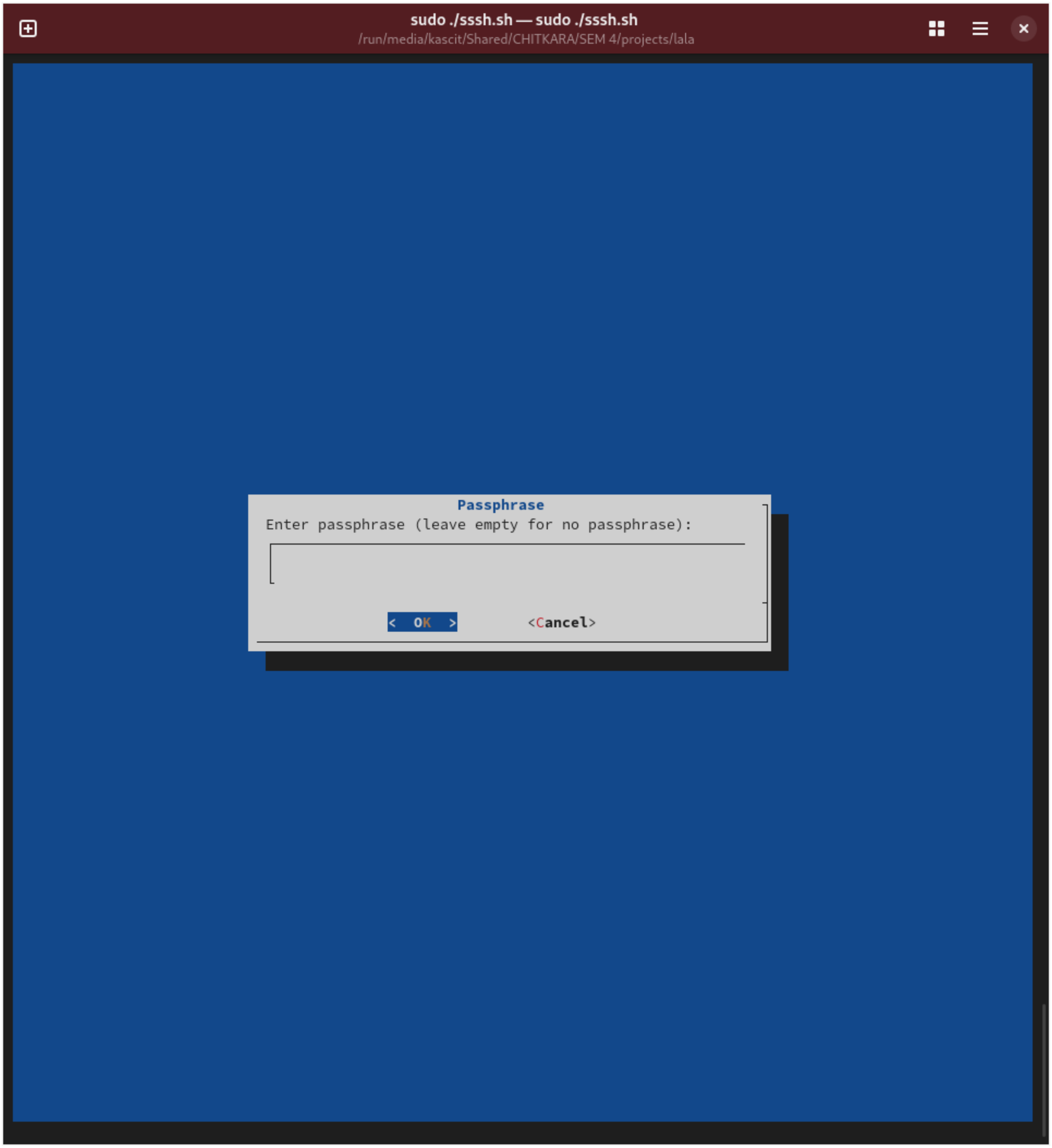
(main menu)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

```
Generate SSH Key
Enter the username to generate the key for:

root

< OK >          <Cancel>
```

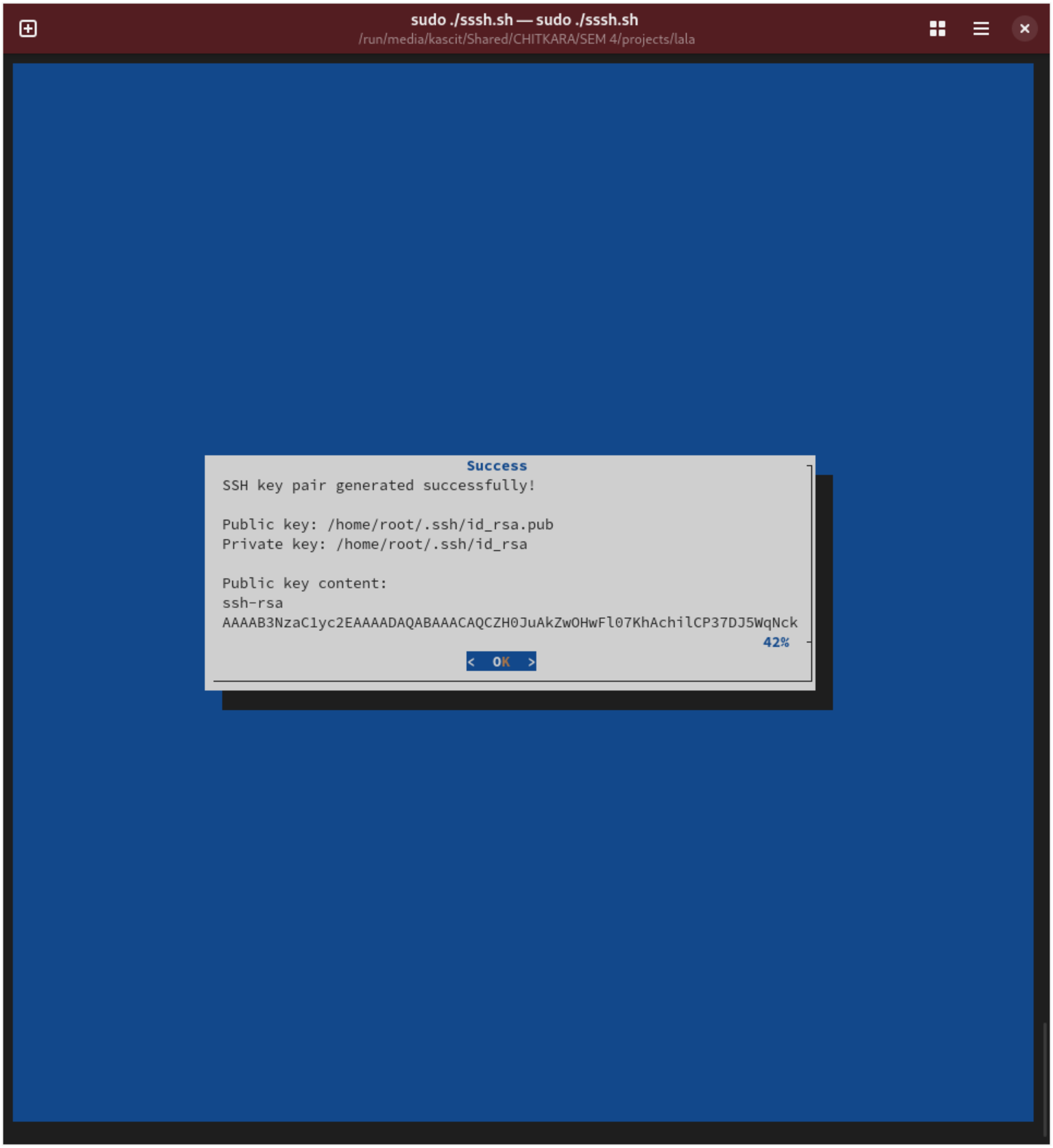
(generate SSH key pair #1)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

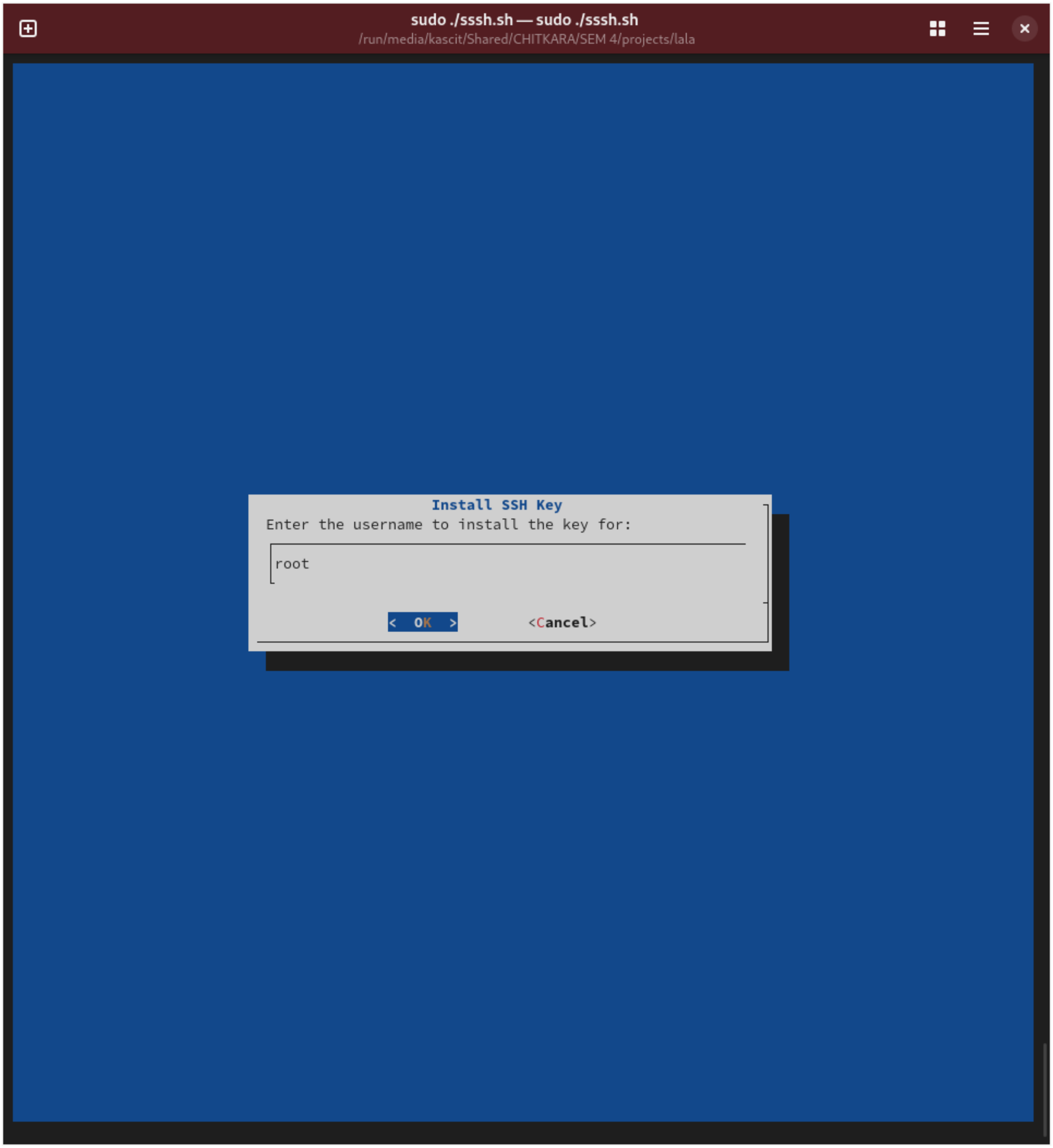
```
Passphrase
Enter passphrase (leave empty for no passphrase):

< OK >          <Cancel>
```

(generate SSH key pair #2)


sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

```
                              Success
SSH key pair generated successfully!

Public key: /home/root/.ssh/id_rsa.pub
Private key: /home/root/.ssh/id_rsa

Public key content:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQCZH0JuAkZwOHwFl07KhAchilCP37DJ5WqNck
                                                                42%
                           <  OK  >
```

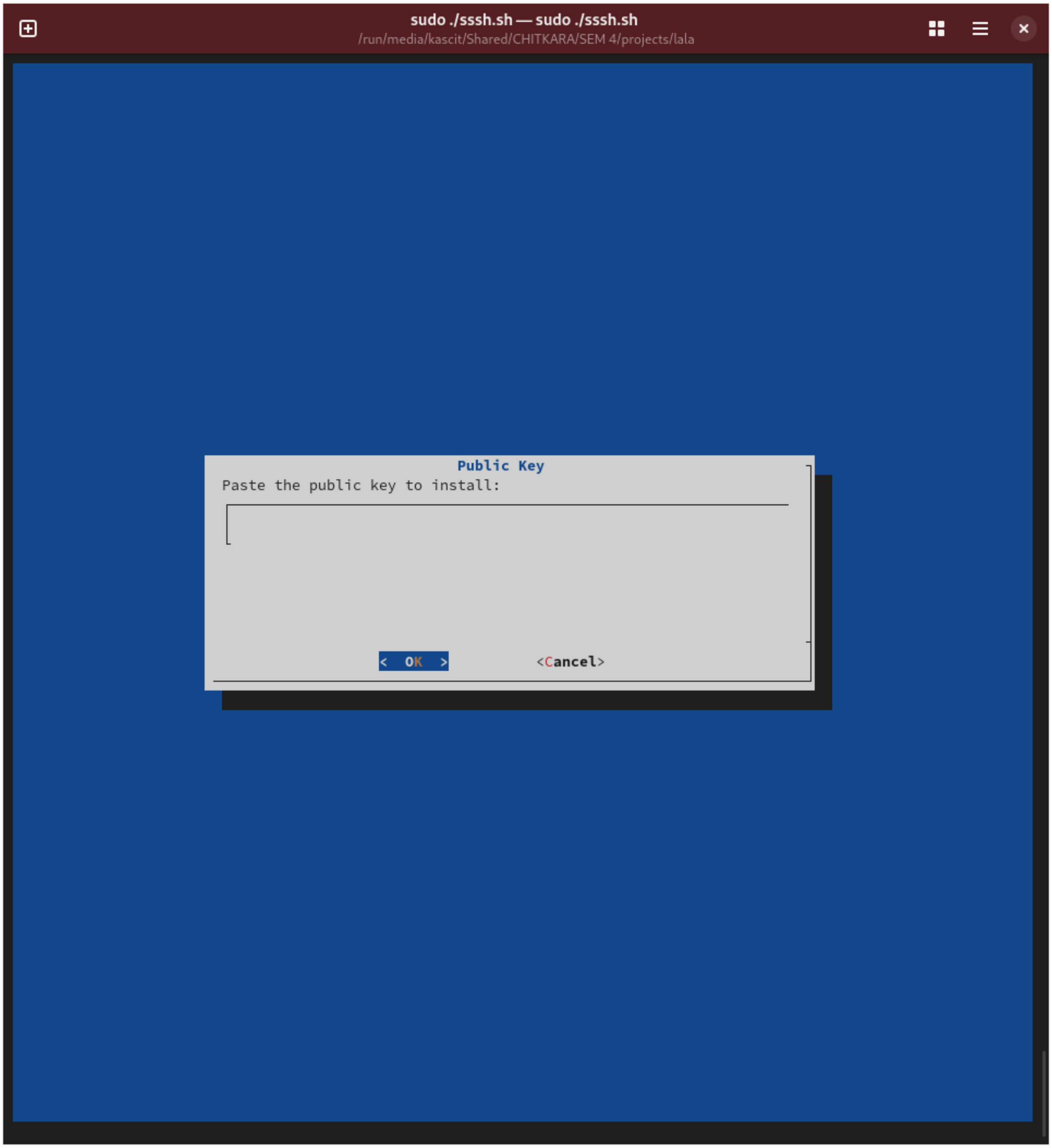
(generate SSH key pair #3)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

```
Install SSH Key
Enter the username to install the key for:

root

< OK >          <Cancel>
```

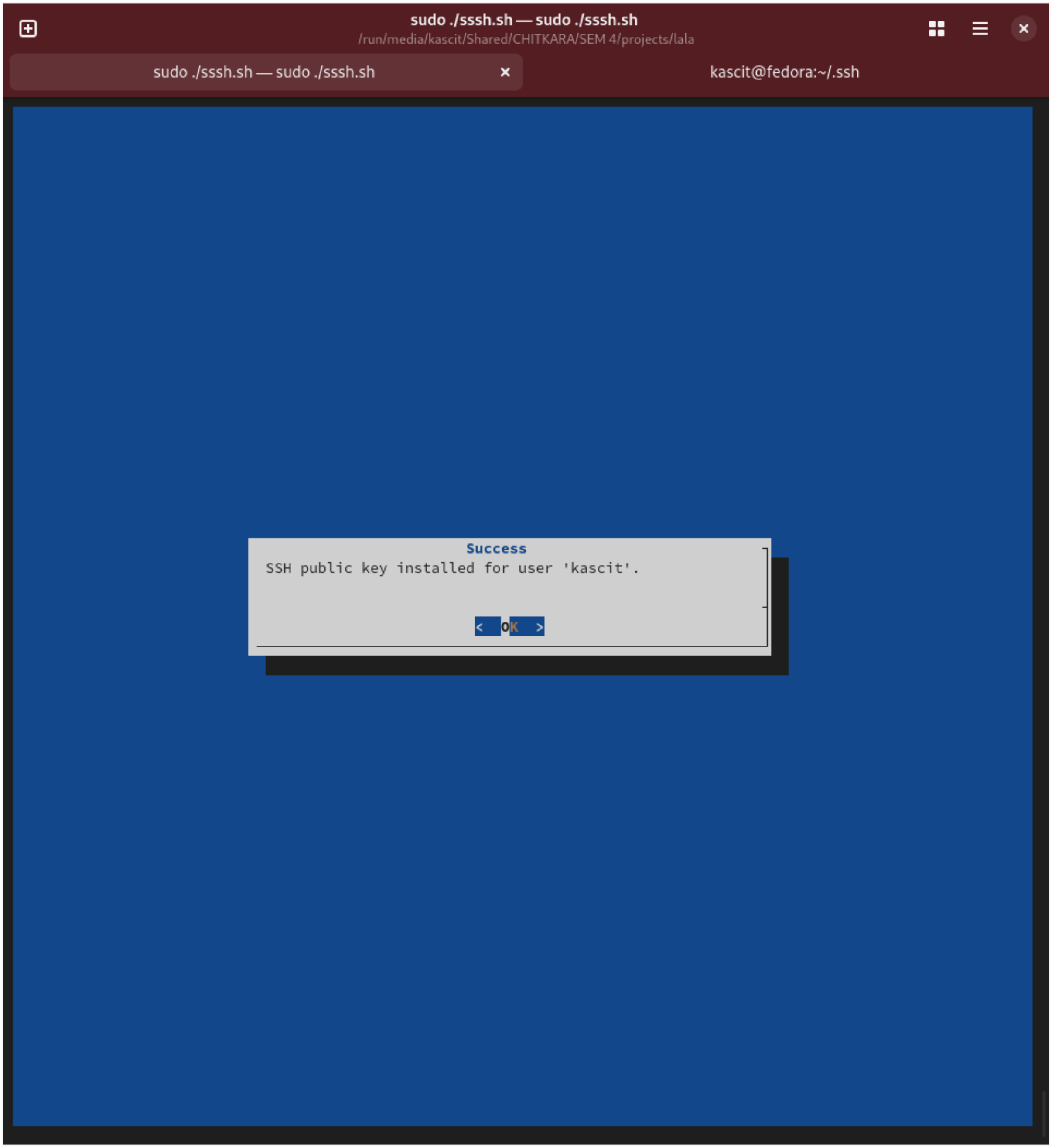
(install SSH public key #1)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

```
Public Key
Paste the public key to install:
```

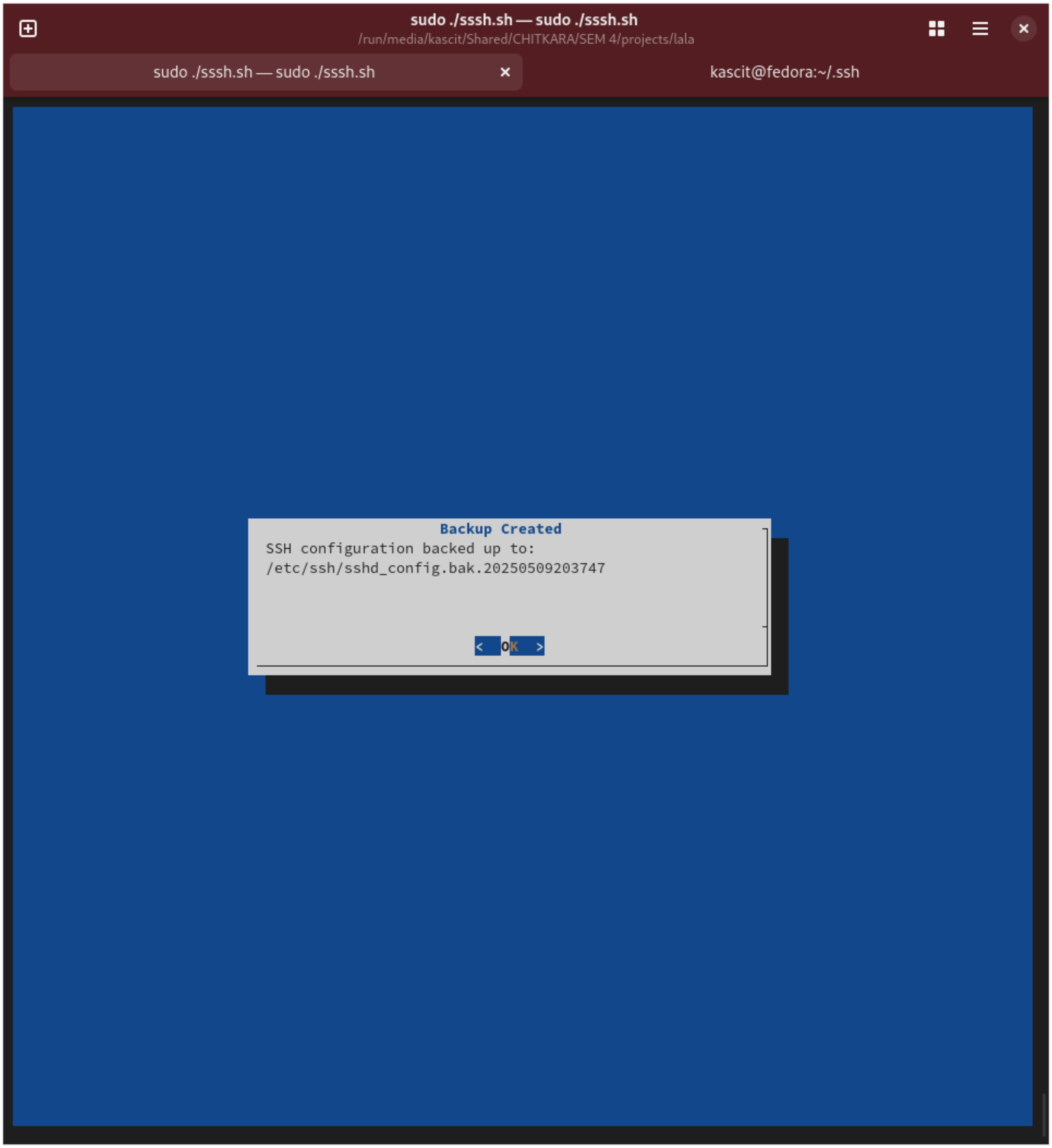
(install SSH public key #2)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala
sudo ./sssh.sh — sudo ./sssh.sh
kascit@fedora:~/.ssh

```
Success
SSH public key installed for user 'kascit'.

< OK >
```

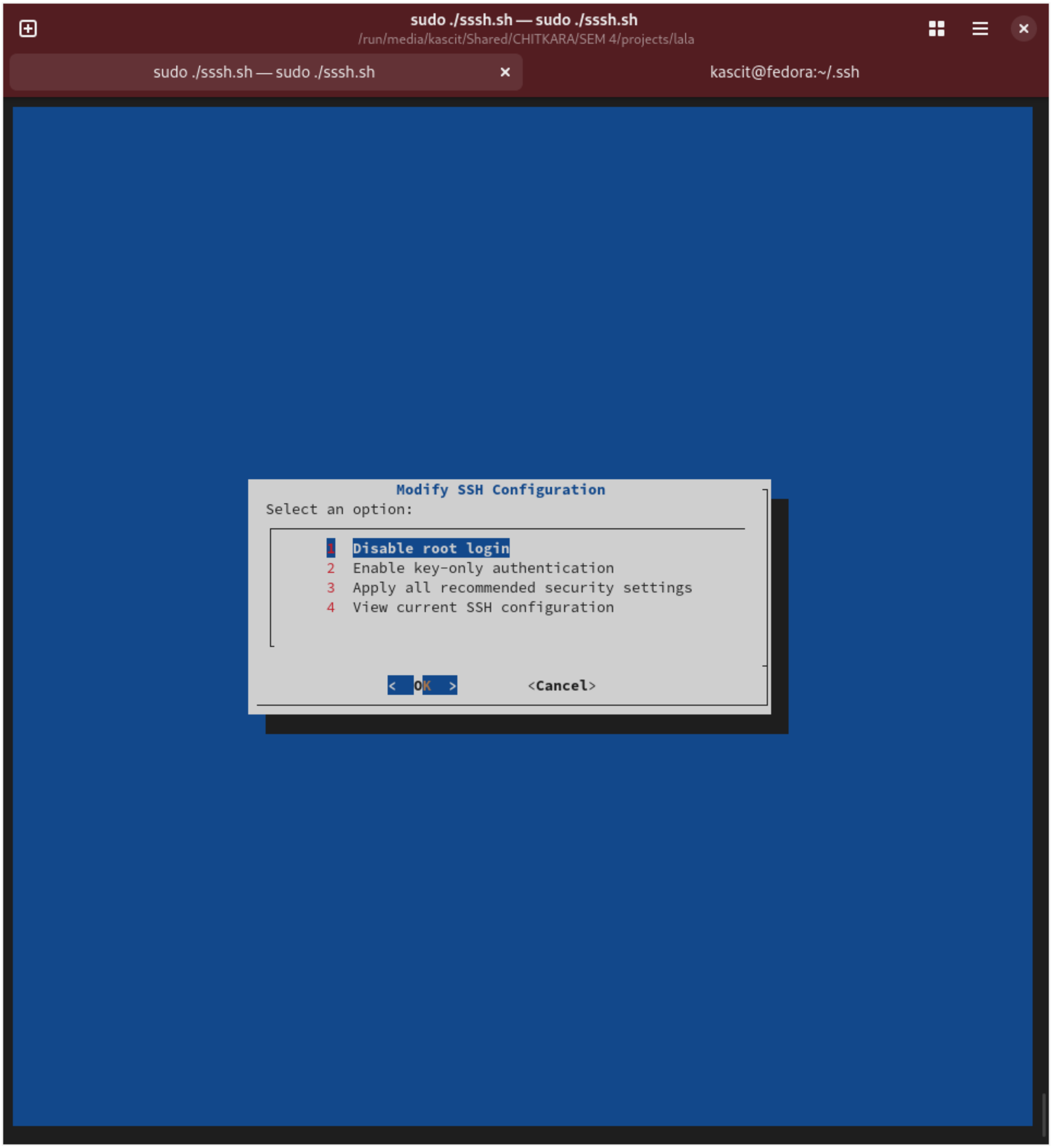
(install SSH public key #3)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala
sudo ./sssh.sh — sudo ./sssh.sh
kascit@fedora:~/.ssh

```
Backup Created
SSH configuration backed up to:
/etc/ssh/sshd_config.bak.20250509203747
```

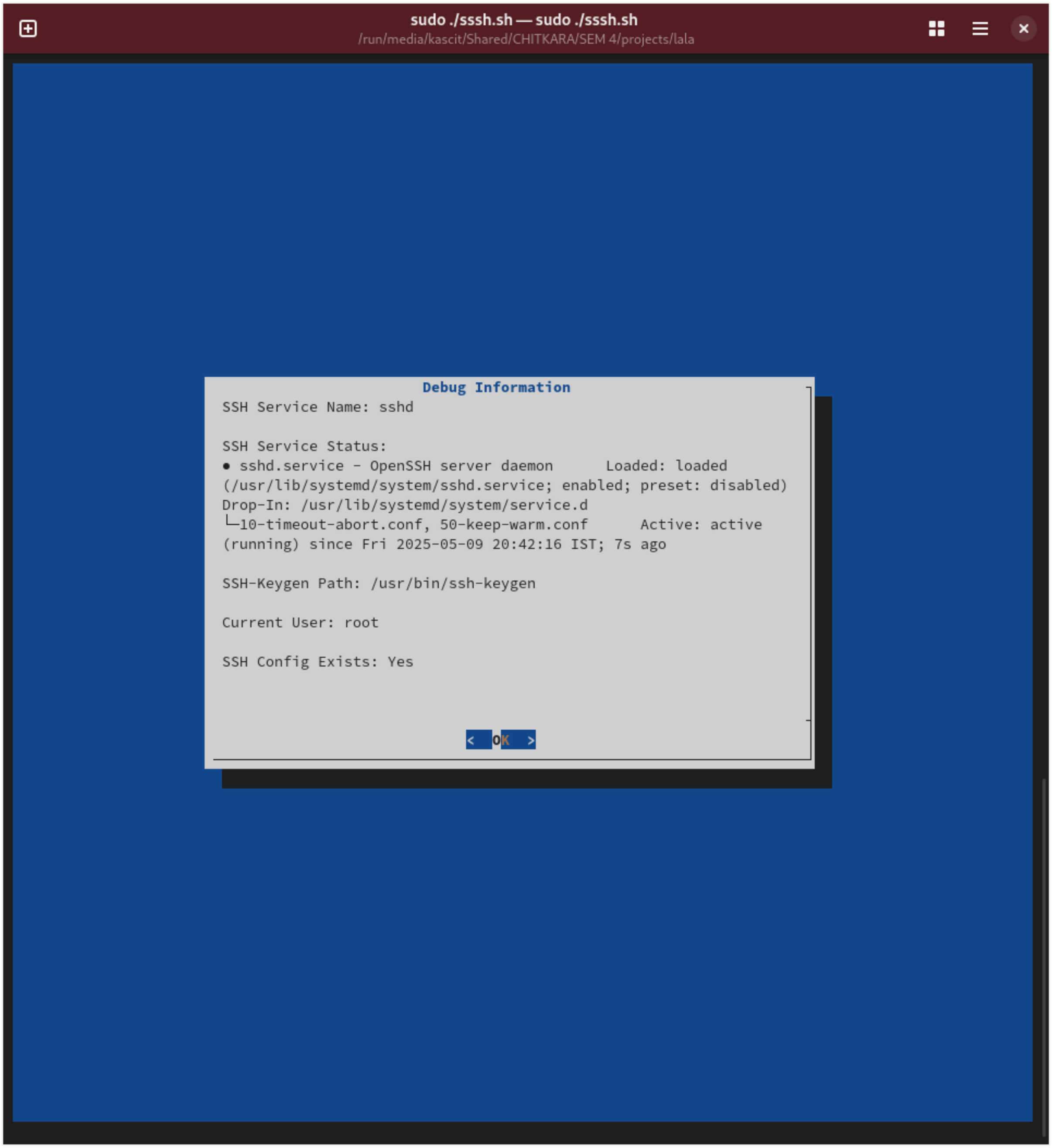
(auto backup before modifying SSH config)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala
sudo ./sssh.sh — sudo ./sssh.sh
kascit@fedora:~/.ssh

```
                 Modify SSH Configuration
 Select an option:

        1   Disable root login
        2   Enable key-only authentication
        3   Apply all recommended security settings
        4   View current SSH configuration


              <  OK  >           <Cancel>
```

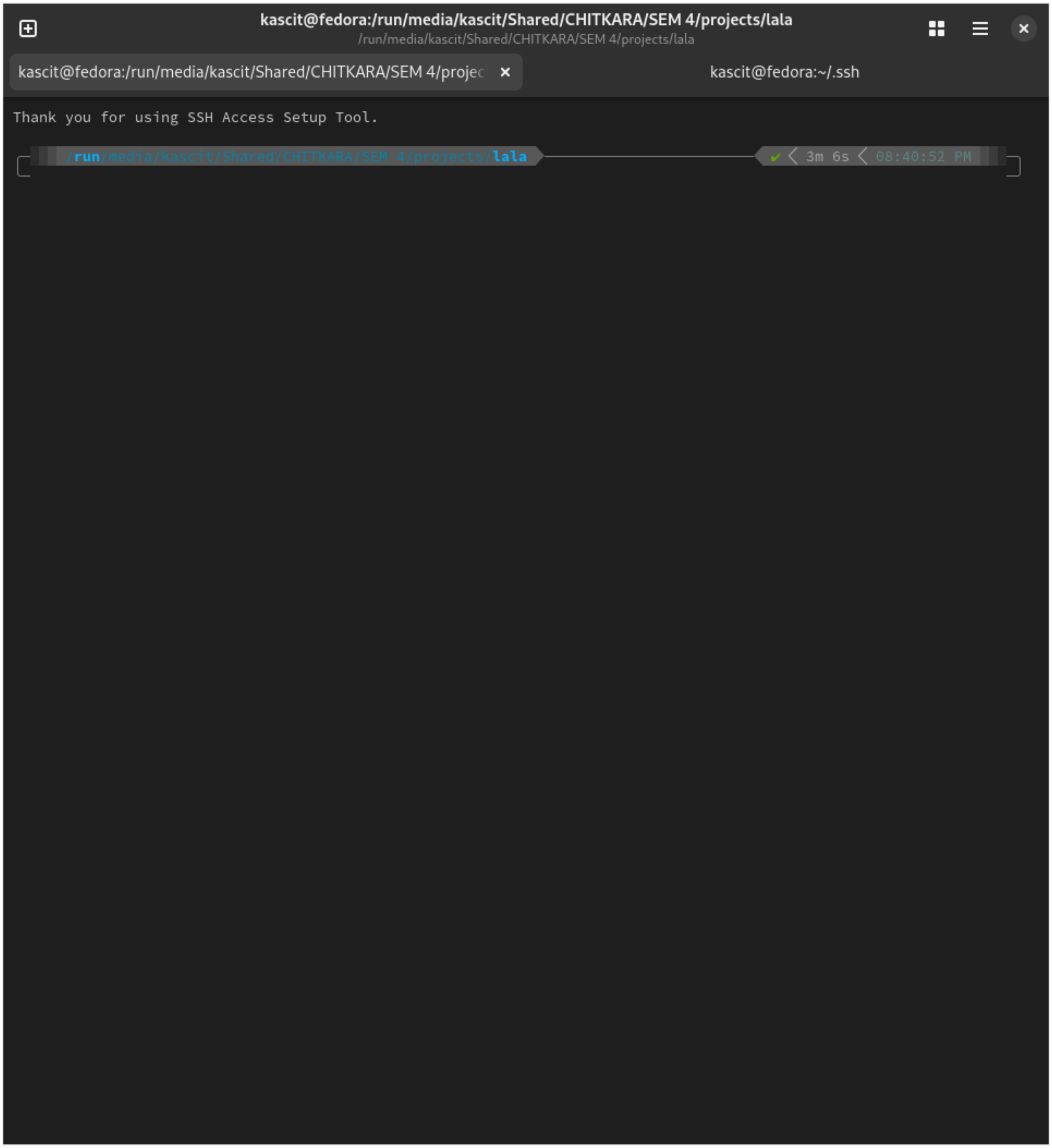
(modify SSH config menu)



sudo ./sssh.sh — sudo ./sssh.sh
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala

```
                      Debug Information
SSH Service Name: sshd

SSH Service Status:
● sshd.service - OpenSSH server daemon      Loaded: loaded
(/usr/lib/systemd/system/sshd.service; enabled; preset: disabled)
Drop-In: /usr/lib/systemd/system/service.d
└─10-timeout-abort.conf, 50-keep-warm.conf      Active: active
(running) since Fri 2025-05-09 20:42:16 IST; 7s ago

SSH-Keygen Path: /usr/bin/ssh-keygen

Current User: root

SSH Config Exists: Yes
```

(debug information)



kascit@fedora:/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala
/run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala
kascit@fedora:/run/media/kascit/Shared/CHITKARA/SEM 4/projec
kascit@fedora:~/.ssh

```
Thank you for using SSH Access Setup Tool.

  /run/media/kascit/Shared/CHITKARA/SEM 4/projects/lala ──────────── ✔ < 3m 6s < 08:40:52 PM
```

(exit)

These outputs confirmed the effectiveness of the implemented configuration.

6. Security and Optimization

6.1 Hardening Measures Taken

Security was at the core of this project. The following best practices were implemented to harden the OpenSSH server:

- **Key-Based Authentication Only**: Disabled password login entirely by setting `PasswordAuthentication no` and enforcing `PubkeyAuthentication yes`. This prevents brute-force password attacks.
- **Disabled Root Login**: `PermitRootLogin no` ensures the root account is not directly accessible over SSH.
- **Idle Session Timeout**: Configured `ClientAliveInterval 300` and `ClientAliveCountMax 2` to disconnect idle sessions after 10 minutes, reducing the window for unauthorized access if a session is left open.
- **Limited Login Attempts**: `MaxAuthTries 3` restricts brute-force attempts.
- **Shortened Login Grace Period**: `LoginGraceTime 60` forces timely login, minimizing lingering unauthenticated sessions.
- **Disabled Challenge-Response Authentication**: Prevents legacy insecure authentication methods.
- **File Permissions**: Proper permissions (`chmod 700 ~/.ssh` and `chmod 600 ~/.ssh/authorized_keys`) were applied to prevent unauthorized access to key files.

6.2 Performance Tuning and Efficiency

Although SSH is lightweight by default, minor tuning was done for responsiveness and maintainability:

- **Reduced DNS Resolution Delay**: `UseDNS no` (commented by default, can be enabled) improves connection speed by skipping reverse DNS lookup.
- **Protocol Optimization**: Ensured `Protocol 2` (modern, secure SSH protocol) was explicitly used for safer key exchange and encryption.
- **Minimal Attack Surface**: By disabling root login, password authentication, and challenge-response methods, the number of potential entry points is significantly reduced.

6.3 Backup and Recovery Measures

- **Backup of SSH Keys**: The generated private and public keys were backed up securely on an external drive.
- **Config Snapshot**: The original `sshd_config` was backed up (`cp sshd_config sshd_config.bak`) before making modifications.
- **Recovery Login Method**: In case of misconfiguration or accidental lockout, a root user with physical or local console access can revert changes from TTY.
- **Audit Logs**: Login attempts and authentication failures were monitored using `/var/log/auth.log`, aiding forensic analysis if needed.

7. Testing and Validation

7.1 Test Scenarios and Expected Results

To ensure the SSH configuration changes were effective and secure, the following test cases were performed:

- **Key-Based SSH Login (Success Case)**
  - **Scenario**: Connect using the generated private key from the user account `kascit`.
  - **Command**: `ssh -i /home/kascit/.ssh/id_rsa kascit@localhost`
  - **Expected Result**: Successful login without a password prompt.
- **Password Authentication (Failure Case)**
  - **Scenario**: Attempt to login without an SSH key.
  - **Expected Result**: Login rejected due to `PasswordAuthentication no`.
- **Root Login (Failure Case)**
  - **Scenario**: Try to SSH as root user.
  - **Expected Result**: Login denied due to `PermitRootLogin no`.
- **Idle Session Timeout**
  - **Scenario**: Leave session idle for more than 10 minutes.
  - **Expected Result**: Session automatically disconnected.
- **Excessive Login Attempts**
  - **Scenario**: Deliberately input wrong key multiple times.
  - **Expected Result**: Connection dropped after 3 failed attempts due to `MaxAuthTries 3`.
- **Key Permissions Check**
  - **Scenario**: Test login with incorrect permissions on key files.
  - **Expected Result**: SSH denies connection, logs permission errors.

7.2 Troubleshooting Techniques

Common debugging and recovery approaches included:

- **Verbose SSH Output**
  - Used `ssh -v` to see step-by-step authentication process.
  - Helpful in diagnosing key loading or permission issues.
- **System Logs**
  - Checked `/var/log/auth.log` for login attempts and errors.
  - Used `journalctl -u ssh` (or `sshd`) for service-level logs.
- **Service Restart Verification**
  - Every config change was followed by:

    ```
    sudo systemctl restart sshd
    ```
  - Verified with:

    ```
    sudo systemctl status sshd
    ```
- **Recovery Access**
  - If locked out remotely, physical or local terminal access allowed reverting the configuration safely.

7.3 Logs and Monitoring Tools

- **Authentication Logs**:
  - `/var/log/auth.log` and `/var/log/secure` were continuously monitored.
- **Login Monitoring**:
  - `last` and `who` commands were used to verify login history.
- **Future Plans**:
  - Considered enabling fail2ban to block repeated failed SSH attempts from the same IP.

8. Challenges and Limitations

8.1 Problems Faced During Implementation

Several practical challenges emerged during the course of the project:

- **SSH Key Permissions**: OpenSSH is highly sensitive to key file permissions. Even a slight deviation (like `~/.ssh/id_rsa` not having `600`) caused access denials. Ensuring the right ownership and restrictive permissions was crucial.
- **Missing SSH Service Alias**: Some systems use `sshd` instead of `ssh` as the service name. The initial command `sudo systemctl status ssh` failed until the correct service name (`sshd`) was used.
- **Local vs. Remote Configuration**: Testing on localhost didn’t fully simulate real-world network behavior. Some configurations (e.g., hostname resolution, port forwarding) behaved slightly differently over the LAN.
- **Firewall and Port Accessibility**: Though out of project scope, in a LAN or remote setting, port 22 had to be explicitly allowed via the firewall. This was observed but not deeply addressed.
- **Service Restart Confusion**: After editing config files, forgetting to restart the SSH service (`sudo systemctl restart sshd`) resulted in unchanged behavior, causing brief confusion.

8.2 Workarounds and Fixes

- Created a checklist to validate key permissions (`chmod 700 ~/.ssh`, `chmod 600 ~/.ssh/id_rsa`) after every regeneration.
- Identified and confirmed the correct SSH service name using `systemctl list-units | grep ssh`.
- Used verbose flags (`-v`, `-vv`) for SSH to trace connection issues and differentiate client-side and server-side errors.
- Leveraged loopback and LAN setups to mimic real-world testing without public exposure.

8.3 Known Issues or Constraints

- **LAN Testing Only**: The project was implemented and tested in a local environment (localhost and LAN). Real internet exposure, dynamic IP handling, and DNS configuration were not part of this phase.
- **User Key Management Not Automated**: Users must manually generate and manage their SSH keys. No key deployment automation or centralized key revocation logic was implemented.
- **No Intrusion Detection/Brute Force Mitigation Yet**: Tools like `fail2ban` or auditd, which add dynamic protection against repeated login attempts, were noted but not set up in this round.

9. Conclusion and Future Work

9.1 Summary of Accomplishments

This project successfully achieved the goal of configuring and securing an OpenSSH server on a Linux system. The SSH server was set up with strong security practices, including key-based authentication, disabled root login, and disabled password authentication. Idle session timeouts were implemented, and logging controls were enforced to ensure system integrity. All changes were thoroughly tested using SSH tools, and configurations were validated in both localhost and LAN environments. Through this process, a secure and efficient SSH setup was established that can be used as a foundation for future Linux system security enhancements.

9.2 Learnings from the Project

- **In-depth Understanding of SSH Security**: The project provided valuable insight into SSH security mechanisms, such as key-based authentication and server hardening. It reinforced the importance of correct file permissions and the risks of insecure configurations like root login.
- **Linux System Administration Skills**: Throughout the project, the need for knowledge of Linux system services, such as `systemctl` for managing the SSH service, was crucial. The configuration and testing steps improved understanding of Linux server environments, including service management and security controls.
- **Problem-Solving and Debugging**: The challenges faced and resolved during implementation helped hone troubleshooting skills, particularly with SSH connectivity issues, configuration errors, and service management on Linux.
- **Security Awareness**: By configuring SSH with best security practices and emphasizing access control, the project emphasized the importance of proactive system security measures in preventing unauthorized access.

9.3 Future Enhancements

While the project successfully achieved its goals, several future improvements can be made:

- **Automated User Key Management**: Implement a system for automatically managing and distributing user SSH keys to simplify user onboarding and management.
- **Enhanced Intrusion Detection**: Integrate intrusion detection and mitigation tools like `fail2ban` to further protect the SSH server from brute-force attacks and repeated login attempts.
- **Remote Access and Firewall Setup**: Expand the scope to configure SSH for remote access over the internet, including firewall configurations, dynamic DNS, and IP whitelisting for better access control.
- **Centralized Key Revocation System**: Build a mechanism to revoke user access centrally by updating the `authorized_keys` file dynamically and securely, providing more granular control over who can access the server.
- **Automation of SSH Configuration**: Develop an automation script to deploy SSH hardening configurations across multiple systems, ensuring consistent application of best practices on a larger scale.

10. References

- 5 Best Practices for Securing SSH
- The Best Ways to Secure Your SSH Server
- SSH Security Best Practices
- How to secure SSH best practices
- Eight ways to protect SSH access on your system
- Dialog Manual
- systemd - Why not ssh.service but sshd.service?
- centos - SELinux preventing ssh via public key

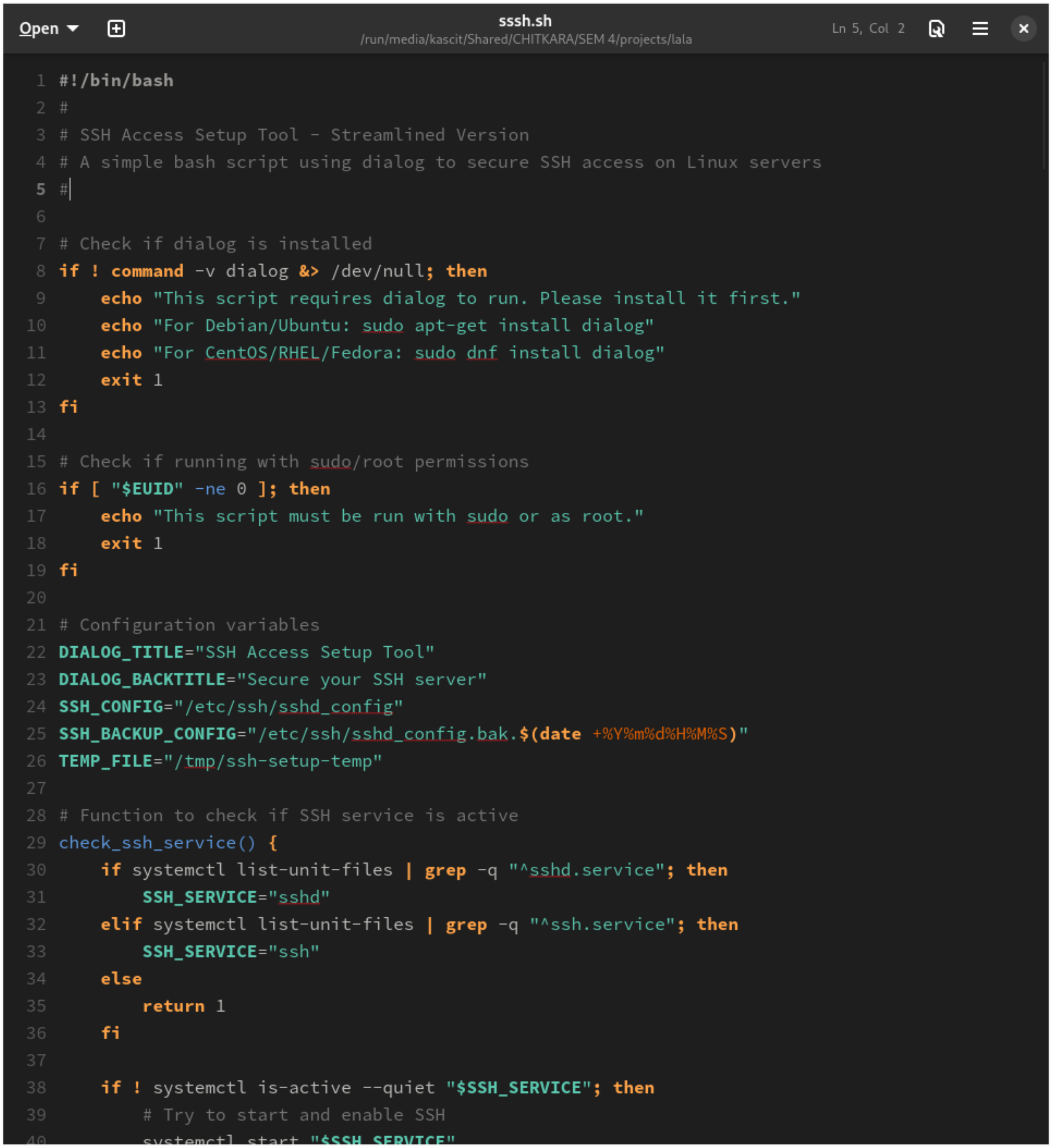
11. Appendices

11.1 Configuration Files

Change in configuration after applying recommended security settings:

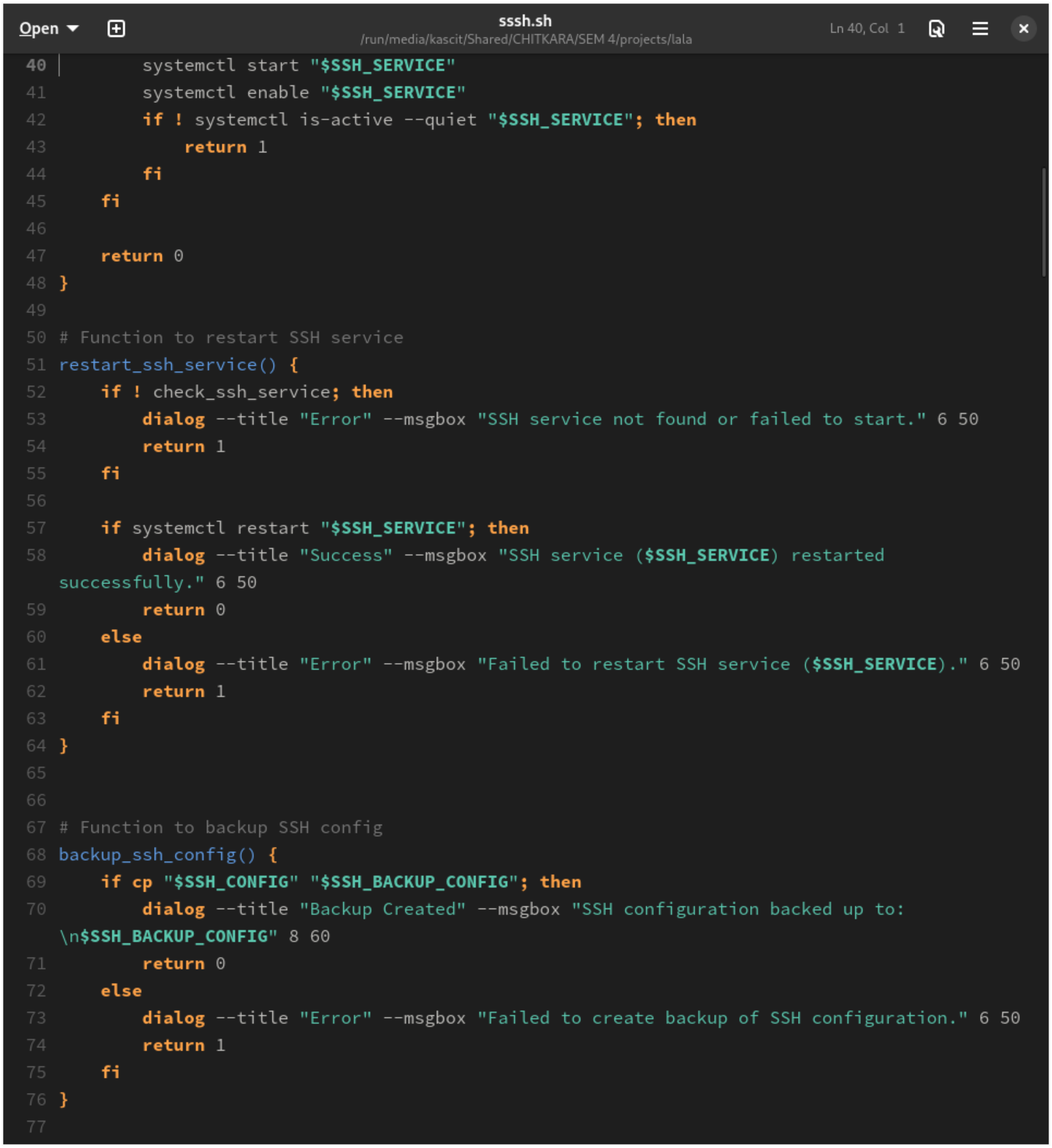
```
# /etc/ssh/sshd_config
PermitRootLogin no
PasswordAuthentication no
ChallengeResponseAuthentication no
PubkeyAuthentication yes
Protocol 2
X11Forwarding no
ClientAliveInterval 300
ClientAliveCountMax 2
MaxAuthTries 3
LoginGraceTime 60
```

11.2 Script Screenshots



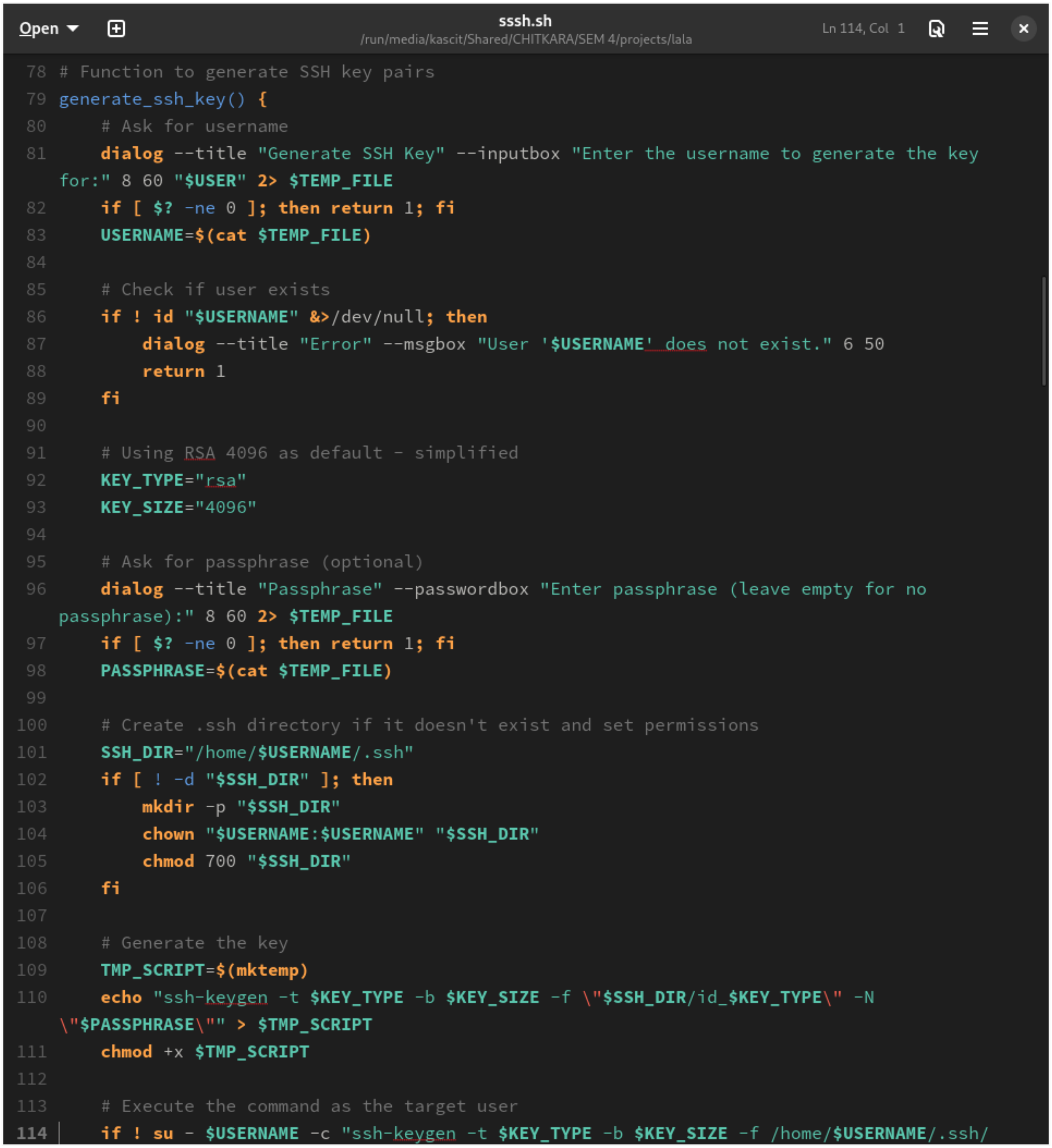
```
#!/bin/bash
#
# SSH Access Setup Tool - Streamlined Version
# A simple bash script using dialog to secure SSH access on Linux servers
#

# Check if dialog is installed
if ! command -v dialog &> /dev/null; then
    echo "This script requires dialog to run. Please install it first."
    echo "For Debian/Ubuntu: sudo apt-get install dialog"
    echo "For CentOS/RHEL/Fedora: sudo dnf install dialog"
    exit 1
fi

# Check if running with sudo/root permissions
if [ "$EUID" -ne 0 ]; then
    echo "This script must be run with sudo or as root."
    exit 1
fi

# Configuration variables
DIALOG_TITLE="SSH Access Setup Tool"
DIALOG_BACKTITLE="Secure your SSH server"
SSH_CONFIG="/etc/ssh/sshd_config"
SSH_BACKUP_CONFIG="/etc/ssh/sshd_config.bak.$(date +%Y%m%d%H%M%S)"
TEMP_FILE="/tmp/ssh-setup-temp"

# Function to check if SSH service is active
check_ssh_service() {
    if systemctl list-unit-files | grep -q "^sshd.service"; then
        SSH_SERVICE="sshd"
    elif systemctl list-unit-files | grep -q "^ssh.service"; then
        SSH_SERVICE="ssh"
    else
        return 1
    fi

    if ! systemctl is-active --quiet "$SSH_SERVICE"; then
        # Try to start and enable SSH
        systemctl start "$SSH_SERVICE"
```

(screen #1)



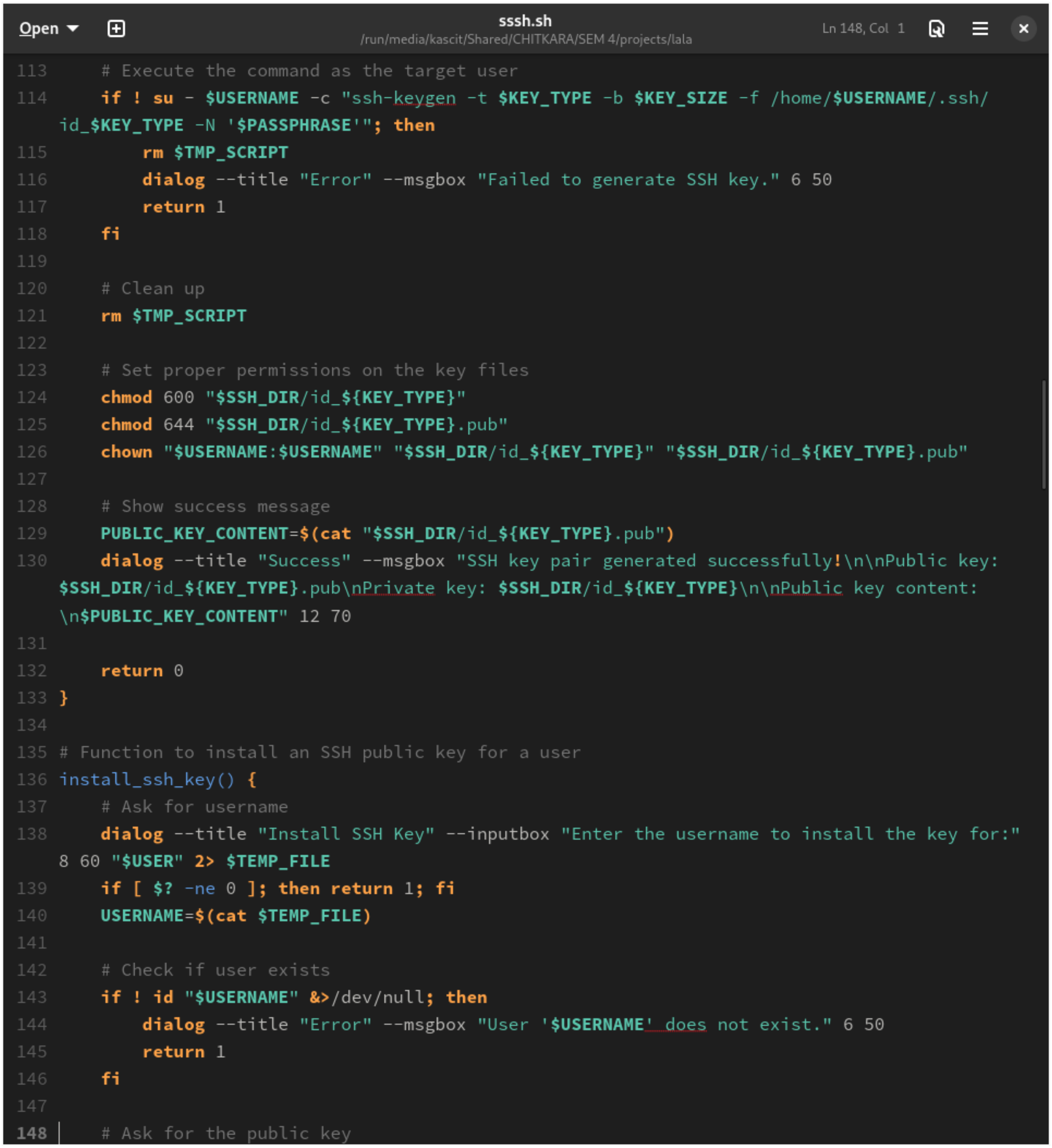
```
        systemctl start "$SSH_SERVICE"
        systemctl enable "$SSH_SERVICE"
        if ! systemctl is-active --quiet "$SSH_SERVICE"; then
            return 1
        fi
    fi

    return 0
}

# Function to restart SSH service
restart_ssh_service() {
    if ! check_ssh_service; then
        dialog --title "Error" --msgbox "SSH service not found or failed to start." 6 50
        return 1
    fi

    if systemctl restart "$SSH_SERVICE"; then
        dialog --title "Success" --msgbox "SSH service ($SSH_SERVICE) restarted successfully." 6 50
        return 0
    else
        dialog --title "Error" --msgbox "Failed to restart SSH service ($SSH_SERVICE)." 6 50
        return 1
    fi
}


# Function to backup SSH config
backup_ssh_config() {
    if cp "$SSH_CONFIG" "$SSH_BACKUP_CONFIG"; then
        dialog --title "Backup Created" --msgbox "SSH configuration backed up to:\n$SSH_BACKUP_CONFIG" 8 60
        return 0
    else
        dialog --title "Error" --msgbox "Failed to create backup of SSH configuration." 6 50
        return 1
    fi
}

```

(screen #2)



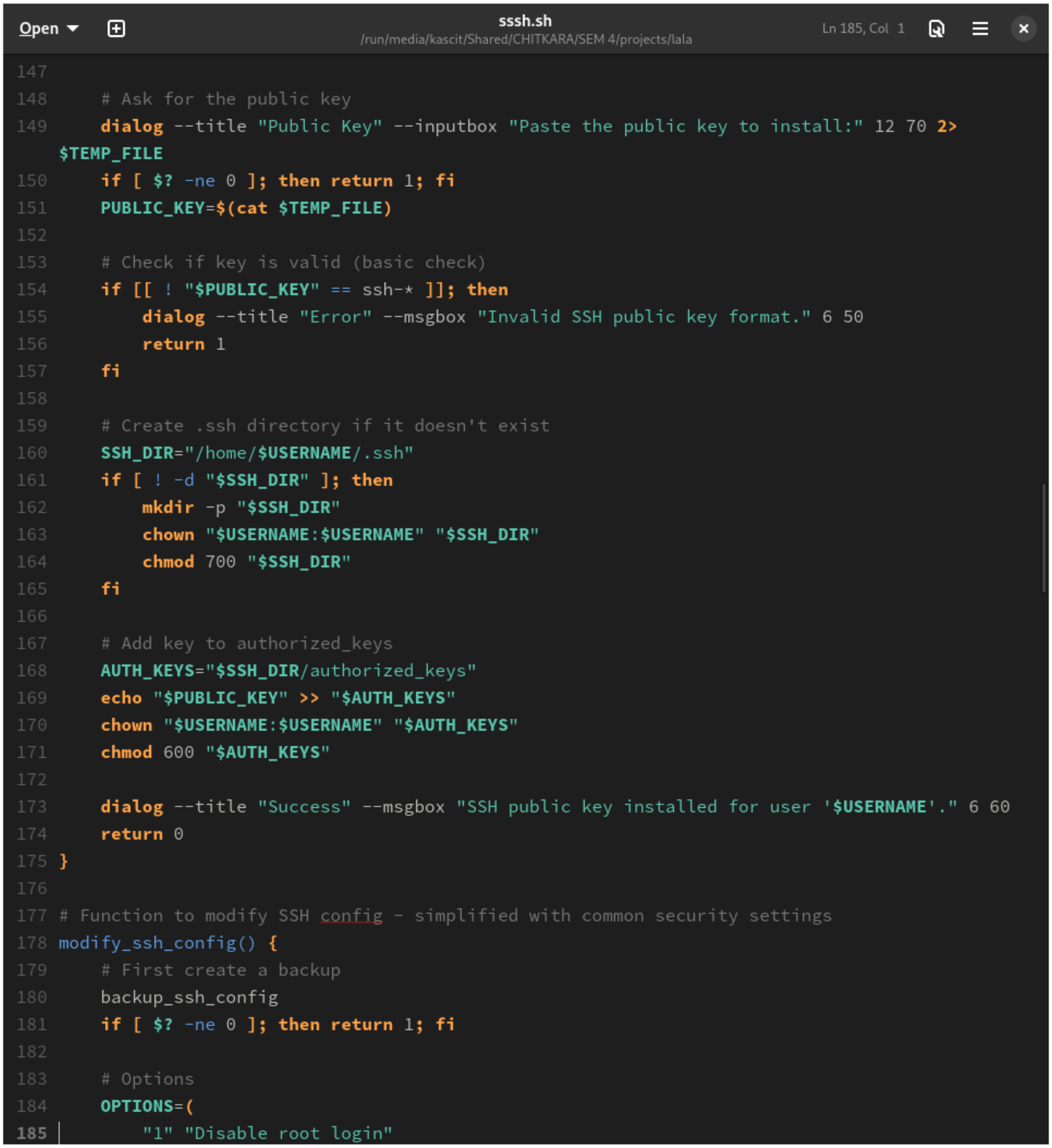
```
# Function to generate SSH key pairs
generate_ssh_key() {
    # Ask for username
    dialog --title "Generate SSH Key" --inputbox "Enter the username to generate the key for:" 8 60 "$USER" 2> $TEMP_FILE
    if [ $? -ne 0 ]; then return 1; fi
    USERNAME=$(cat $TEMP_FILE)

    # Check if user exists
    if ! id "$USERNAME" &>/dev/null; then
        dialog --title "Error" --msgbox "User '$USERNAME' does not exist." 6 50
        return 1
    fi

    # Using RSA 4096 as default - simplified
    KEY_TYPE="rsa"
    KEY_SIZE="4096"

    # Ask for passphrase (optional)
    dialog --title "Passphrase" --passwordbox "Enter passphrase (leave empty for no passphrase):" 8 60 2> $TEMP_FILE
    if [ $? -ne 0 ]; then return 1; fi
    PASSPHRASE=$(cat $TEMP_FILE)

    # Create .ssh directory if it doesn't exist and set permissions
    SSH_DIR="/home/$USERNAME/.ssh"
    if [ ! -d "$SSH_DIR" ]; then
        mkdir -p "$SSH_DIR"
        chown "$USERNAME:$USERNAME" "$SSH_DIR"
        chmod 700 "$SSH_DIR"
    fi

    # Generate the key
    TMP_SCRIPT=$(mktemp)
    echo "ssh-keygen -t $KEY_TYPE -b $KEY_SIZE -f \"$SSH_DIR/id_$KEY_TYPE\" -N \"$PASSPHRASE\"" > $TMP_SCRIPT
    chmod +x $TMP_SCRIPT

    # Execute the command as the target user
    if ! su - $USERNAME -c "ssh-keygen -t $KEY_TYPE -b $KEY_SIZE -f /home/$USERNAME/.ssh/
```

(screen #3)



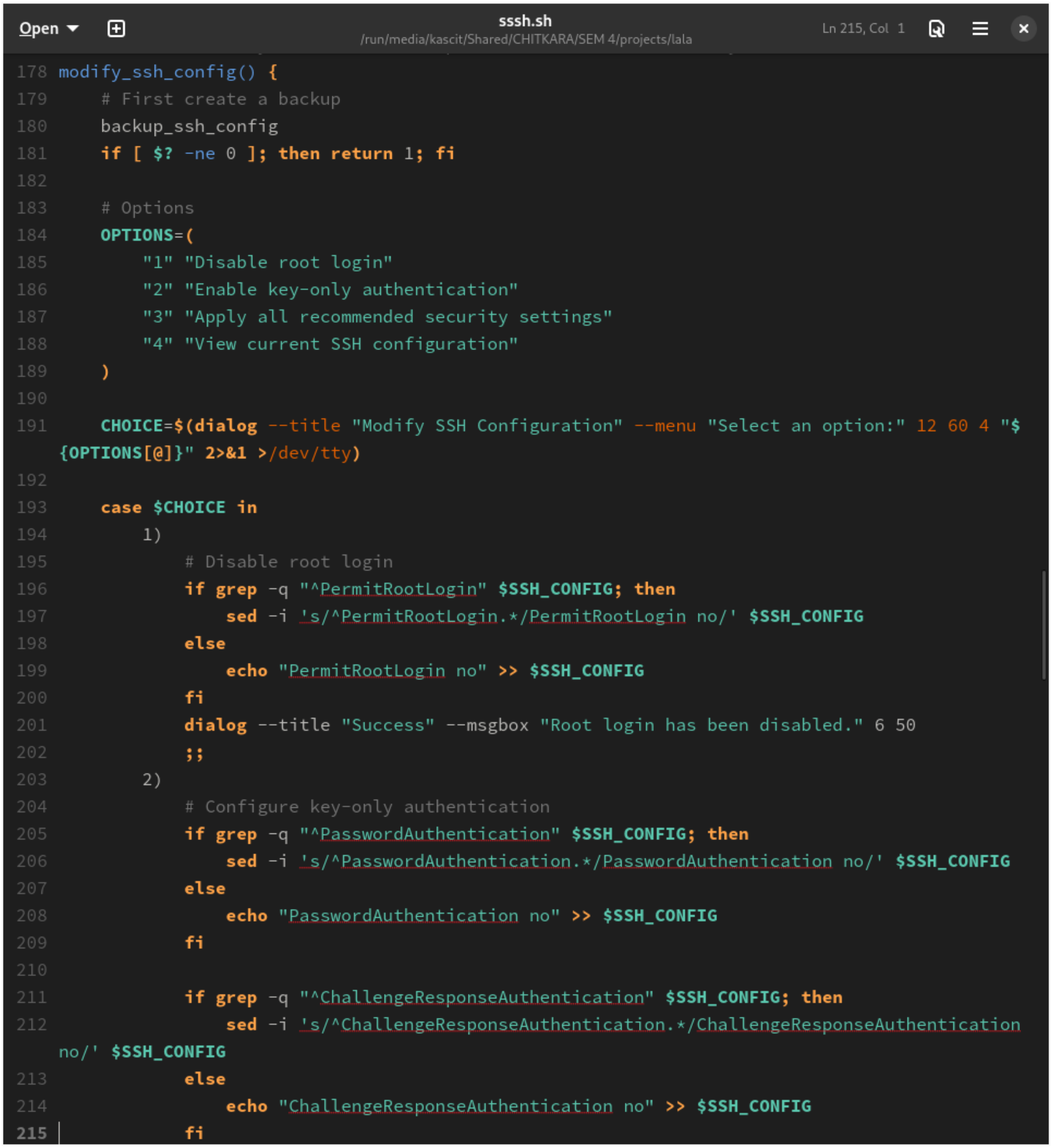
```
    # Execute the command as the target user
    if ! su - $USERNAME -c "ssh-keygen -t $KEY_TYPE -b $KEY_SIZE -f /home/$USERNAME/.ssh/id_$KEY_TYPE -N '$PASSPHRASE'"; then
        rm $TMP_SCRIPT
        dialog --title "Error" --msgbox "Failed to generate SSH key." 6 50
        return 1
    fi

    # Clean up
    rm $TMP_SCRIPT

    # Set proper permissions on the key files
    chmod 600 "$SSH_DIR/id_${KEY_TYPE}"
    chmod 644 "$SSH_DIR/id_${KEY_TYPE}.pub"
    chown "$USERNAME:$USERNAME" "$SSH_DIR/id_${KEY_TYPE}" "$SSH_DIR/id_${KEY_TYPE}.pub"

    # Show success message
    PUBLIC_KEY_CONTENT=$(cat "$SSH_DIR/id_${KEY_TYPE}.pub")
    dialog --title "Success" --msgbox "SSH key pair generated successfully!\n\nPublic key: $SSH_DIR/id_${KEY_TYPE}.pub\nPrivate key: $SSH_DIR/id_${KEY_TYPE}\n\nPublic key content:\n$PUBLIC_KEY_CONTENT" 12 70

    return 0
}

# Function to install an SSH public key for a user
install_ssh_key() {
    # Ask for username
    dialog --title "Install SSH Key" --inputbox "Enter the username to install the key for:" 8 60 "$USER" 2> $TEMP_FILE
    if [ $? -ne 0 ]; then return 1; fi
    USERNAME=$(cat $TEMP_FILE)

    # Check if user exists
    if ! id "$USERNAME" &>/dev/null; then
        dialog --title "Error" --msgbox "User '$USERNAME' does not exist." 6 50
        return 1
    fi

    # Ask for the public key
```

(screen #4)



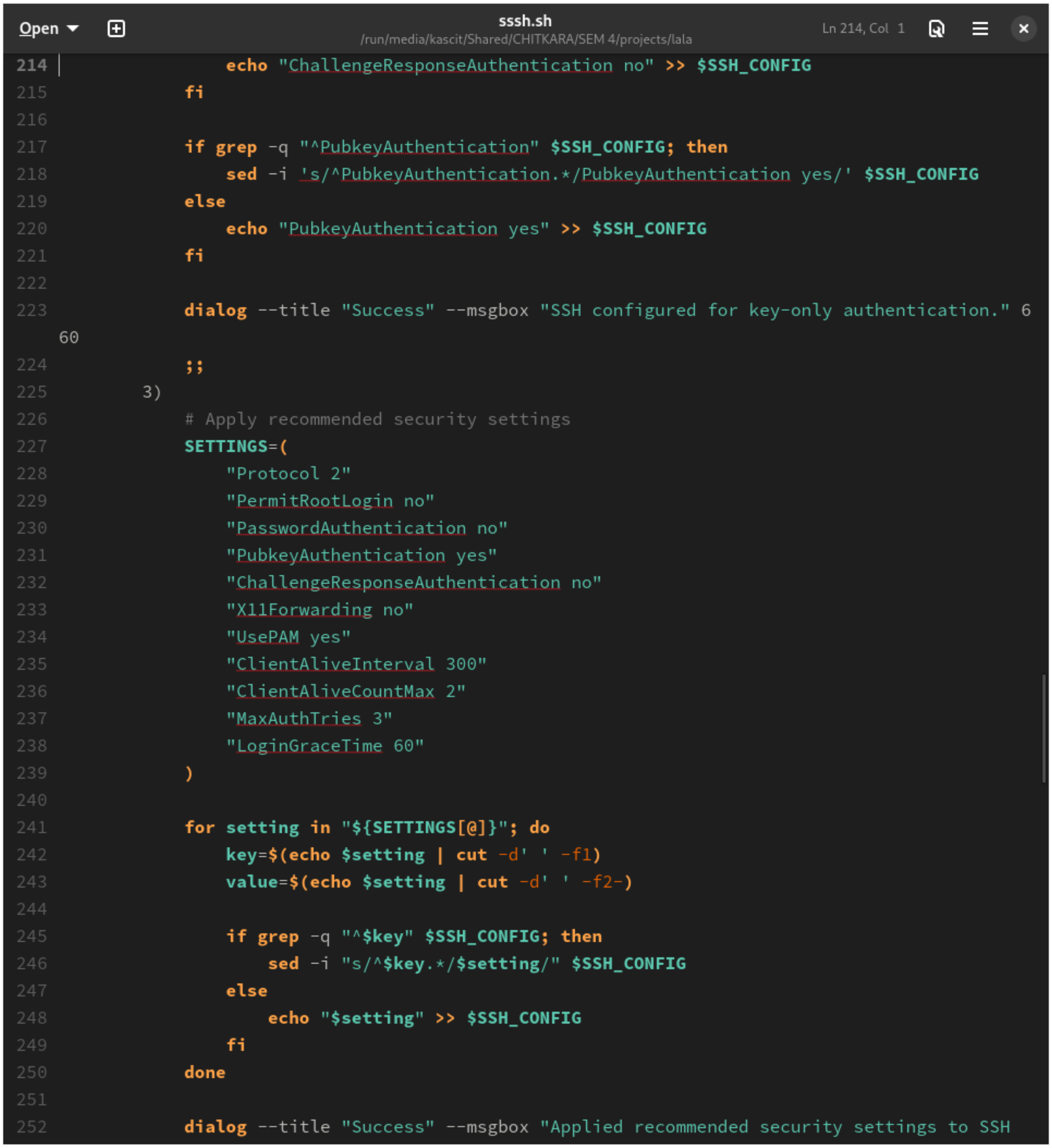
```

    # Ask for the public key
    dialog --title "Public Key" --inputbox "Paste the public key to install:" 12 70 2> $TEMP_FILE
    if [ $? -ne 0 ]; then return 1; fi
    PUBLIC_KEY=$(cat $TEMP_FILE)

    # Check if key is valid (basic check)
    if [[ ! "$PUBLIC_KEY" == ssh-* ]]; then
        dialog --title "Error" --msgbox "Invalid SSH public key format." 6 50
        return 1
    fi

    # Create .ssh directory if it doesn't exist
    SSH_DIR="/home/$USERNAME/.ssh"
    if [ ! -d "$SSH_DIR" ]; then
        mkdir -p "$SSH_DIR"
        chown "$USERNAME:$USERNAME" "$SSH_DIR"
        chmod 700 "$SSH_DIR"
    fi

    # Add key to authorized_keys
    AUTH_KEYS="$SSH_DIR/authorized_keys"
    echo "$PUBLIC_KEY" >> "$AUTH_KEYS"
    chown "$USERNAME:$USERNAME" "$AUTH_KEYS"
    chmod 600 "$AUTH_KEYS"

    dialog --title "Success" --msgbox "SSH public key installed for user '$USERNAME'." 6 60
    return 0
}

# Function to modify SSH config - simplified with common security settings
modify_ssh_config() {
    # First create a backup
    backup_ssh_config
    if [ $? -ne 0 ]; then return 1; fi

    # Options
    OPTIONS=(
        "1" "Disable root login"
```

(screen #5)



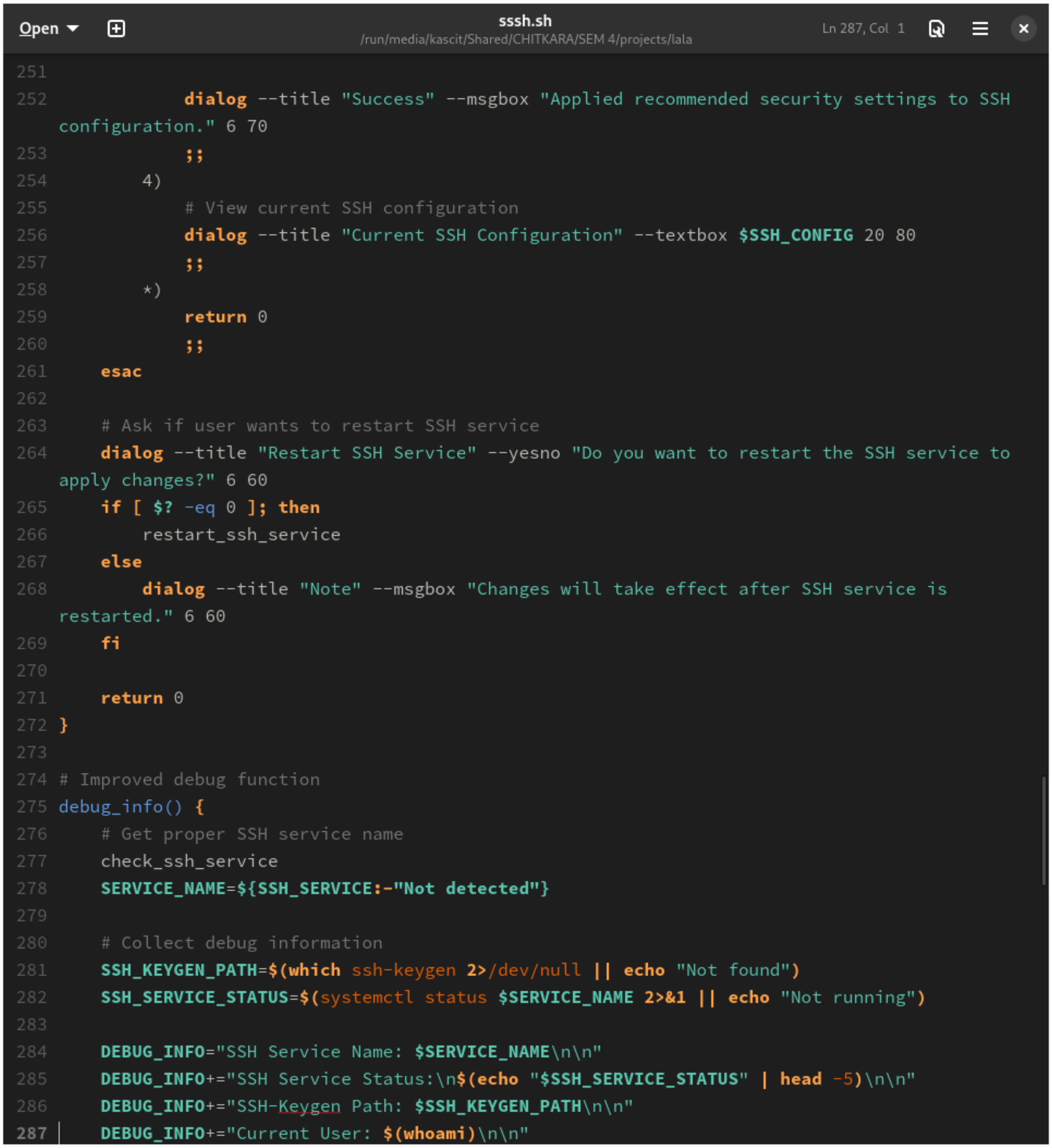
```
modify_ssh_config() {
    # First create a backup
    backup_ssh_config
    if [ $? -ne 0 ]; then return 1; fi

    # Options
    OPTIONS=(
        "1" "Disable root login"
        "2" "Enable key-only authentication"
        "3" "Apply all recommended security settings"
        "4" "View current SSH configuration"
    )

    CHOICE=$(dialog --title "Modify SSH Configuration" --menu "Select an option:" 12 60 4 "${OPTIONS[@]}" 2>&1 >/dev/tty)

    case $CHOICE in
        1)
            # Disable root login
            if grep -q "^PermitRootLogin" $SSH_CONFIG; then
                sed -i 's/^PermitRootLogin.*/PermitRootLogin no/' $SSH_CONFIG
            else
                echo "PermitRootLogin no" >> $SSH_CONFIG
            fi
            dialog --title "Success" --msgbox "Root login has been disabled." 6 50
            ;;
        2)
            # Configure key-only authentication
            if grep -q "^PasswordAuthentication" $SSH_CONFIG; then
                sed -i 's/^PasswordAuthentication.*/PasswordAuthentication no/' $SSH_CONFIG
            else
                echo "PasswordAuthentication no" >> $SSH_CONFIG
            fi

            if grep -q "^ChallengeResponseAuthentication" $SSH_CONFIG; then
                sed -i 's/^ChallengeResponseAuthentication.*/ChallengeResponseAuthentication no/' $SSH_CONFIG
            else
                echo "ChallengeResponseAuthentication no" >> $SSH_CONFIG
            fi
```

(screen #6)



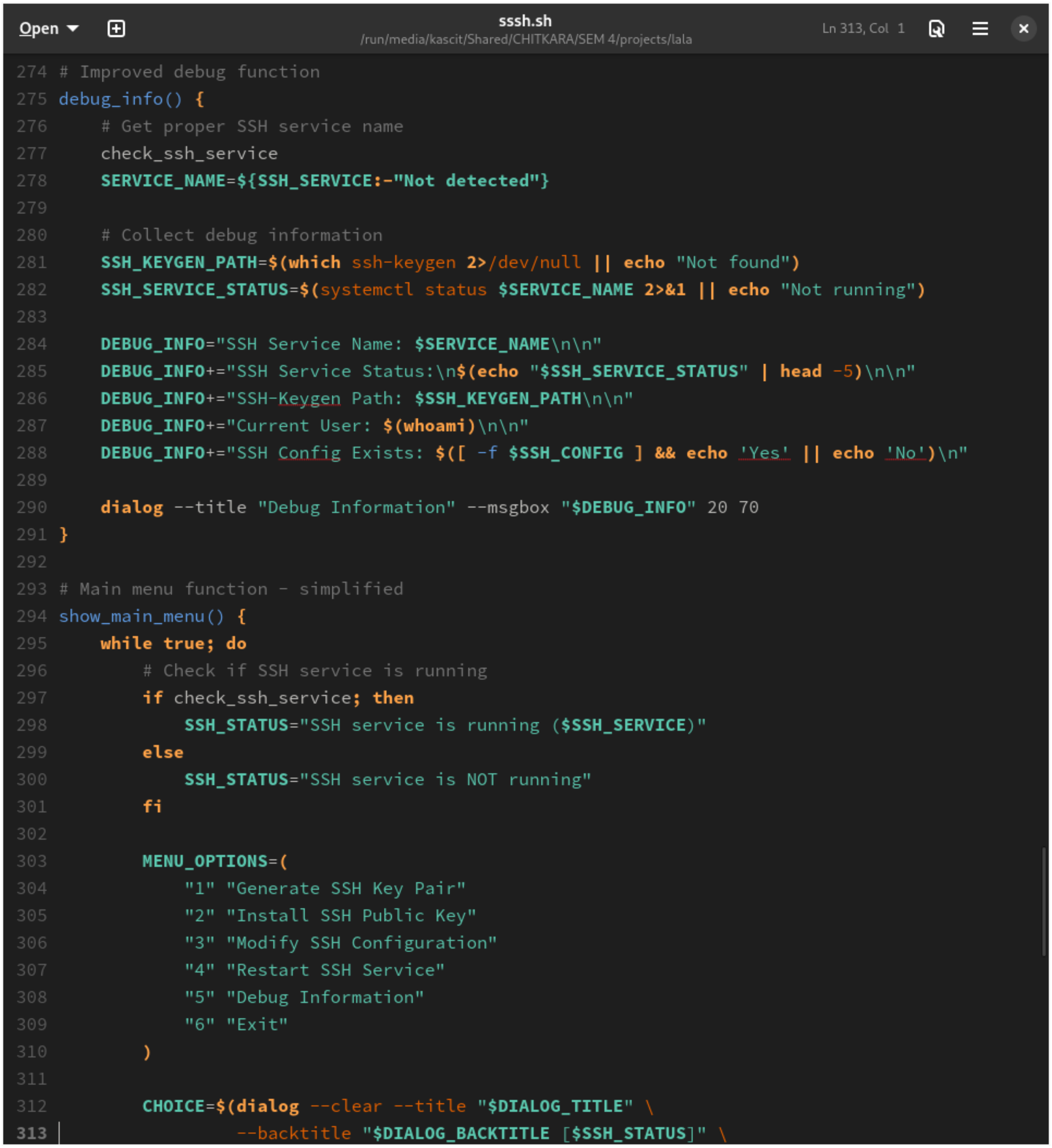
```
                echo "ChallengeResponseAuthentication no" >> $SSH_CONFIG
            fi

            if grep -q "^PubkeyAuthentication" $SSH_CONFIG; then
                sed -i 's/^PubkeyAuthentication.*/PubkeyAuthentication yes/' $SSH_CONFIG
            else
                echo "PubkeyAuthentication yes" >> $SSH_CONFIG
            fi

            dialog --title "Success" --msgbox "SSH configured for key-only authentication." 6 60
            ;;
        3)
            # Apply recommended security settings
            SETTINGS=(
                "Protocol 2"
                "PermitRootLogin no"
                "PasswordAuthentication no"
                "PubkeyAuthentication yes"
                "ChallengeResponseAuthentication no"
                "X11Forwarding no"
                "UsePAM yes"
                "ClientAliveInterval 300"
                "ClientAliveCountMax 2"
                "MaxAuthTries 3"
                "LoginGraceTime 60"
            )

            for setting in "${SETTINGS[@]}"; do
                key=$(echo $setting | cut -d' ' -f1)
                value=$(echo $setting | cut -d' ' -f2-)

                if grep -q "^$key" $SSH_CONFIG; then
                    sed -i "s/^$key.*/$setting/" $SSH_CONFIG
                else
                    echo "$setting" >> $SSH_CONFIG
                fi
            done

            dialog --title "Success" --msgbox "Applied recommended security settings to SSH
```

(screen #7)



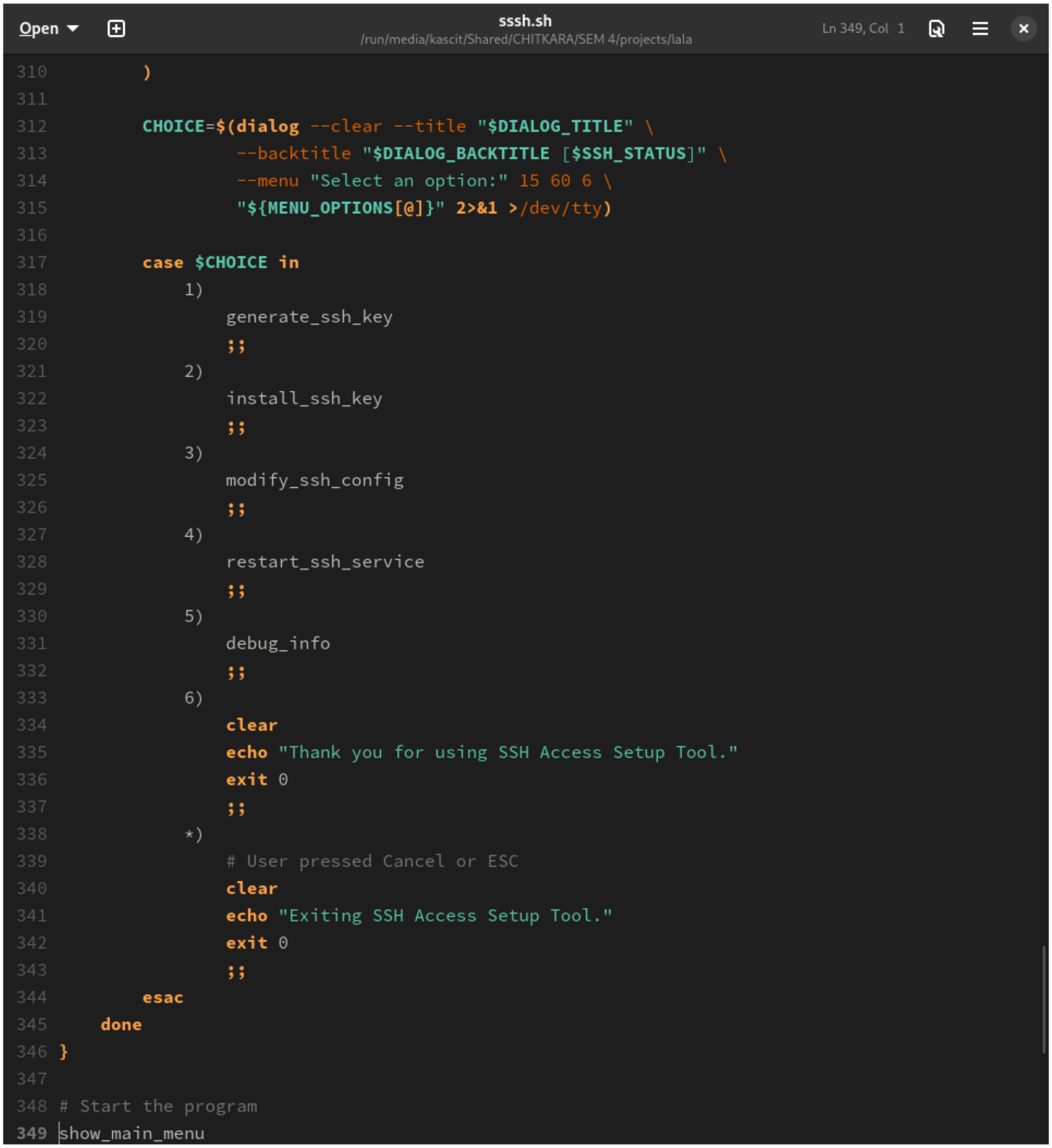
```

            dialog --title "Success" --msgbox "Applied recommended security settings to SSH configuration." 6 70
            ;;
        4)
            # View current SSH configuration
            dialog --title "Current SSH Configuration" --textbox $SSH_CONFIG 20 80
            ;;
        *)
            return 0
            ;;
    esac

    # Ask if user wants to restart SSH service
    dialog --title "Restart SSH Service" --yesno "Do you want to restart the SSH service to apply changes?" 6 60
    if [ $? -eq 0 ]; then
        restart_ssh_service
    else
        dialog --title "Note" --msgbox "Changes will take effect after SSH service is restarted." 6 60
    fi

    return 0
}

# Improved debug function
debug_info() {
    # Get proper SSH service name
    check_ssh_service
    SERVICE_NAME=${SSH_SERVICE:-"Not detected"}

    # Collect debug information
    SSH_KEYGEN_PATH=$(which ssh-keygen 2>/dev/null || echo "Not found")
    SSH_SERVICE_STATUS=$(systemctl status $SERVICE_NAME 2>&1 || echo "Not running")

    DEBUG_INFO="SSH Service Name: $SERVICE_NAME\n\n"
    DEBUG_INFO+="SSH Service Status:\n$(echo "$SSH_SERVICE_STATUS" | head -5)\n\n"
    DEBUG_INFO+="SSH-Keygen Path: $SSH_KEYGEN_PATH\n\n"
    DEBUG_INFO+="Current User: $(whoami)\n\n"
```

(screen #8)



```
# Improved debug function
debug_info() {
    # Get proper SSH service name
    check_ssh_service
    SERVICE_NAME=${SSH_SERVICE:-"Not detected"}

    # Collect debug information
    SSH_KEYGEN_PATH=$(which ssh-keygen 2>/dev/null || echo "Not found")
    SSH_SERVICE_STATUS=$(systemctl status $SERVICE_NAME 2>&1 || echo "Not running")

    DEBUG_INFO="SSH Service Name: $SERVICE_NAME\n\n"
    DEBUG_INFO+="SSH Service Status:\n$(echo "$SSH_SERVICE_STATUS" | head -5)\n\n"
    DEBUG_INFO+="SSH-Keygen Path: $SSH_KEYGEN_PATH\n\n"
    DEBUG_INFO+="Current User: $(whoami)\n\n"
    DEBUG_INFO+="SSH Config Exists: $([ -f $SSH_CONFIG ] && echo 'Yes' || echo 'No')\n"

    dialog --title "Debug Information" --msgbox "$DEBUG_INFO" 20 70
}

# Main menu function - simplified
show_main_menu() {
    while true; do
        # Check if SSH service is running
        if check_ssh_service; then
            SSH_STATUS="SSH service is running ($SSH_SERVICE)"
        else
            SSH_STATUS="SSH service is NOT running"
        fi

        MENU_OPTIONS=(
            "1" "Generate SSH Key Pair"
            "2" "Install SSH Public Key"
            "3" "Modify SSH Configuration"
            "4" "Restart SSH Service"
            "5" "Debug Information"
            "6" "Exit"
        )

        CHOICE=$(dialog --clear --title "$DIALOG_TITLE" \
                 --backtitle "$DIALOG_BACKTITLE [$SSH_STATUS]" \
```

(screen #9)



```bash
        )

        CHOICE=$(dialog --clear --title "$DIALOG_TITLE" \
                 --backtitle "$DIALOG_BACKTITLE [$SSH_STATUS]" \
                 --menu "Select an option:" 15 60 6 \
                 "${MENU_OPTIONS[@]}" 2>&1 >/dev/tty)

        case $CHOICE in
            1)
                generate_ssh_key
                ;;
            2)
                install_ssh_key
                ;;
            3)
                modify_ssh_config
                ;;
            4)
                restart_ssh_service
                ;;
            5)
                debug_info
                ;;
            6)
                clear
                echo "Thank you for using SSH Access Setup Tool."
                exit 0
                ;;
            *)
                # User pressed Cancel or ESC
                clear
                echo "Exiting SSH Access Setup Tool."
                exit 0
                ;;
        esac
    done
}

# Start the program
show_main_menu
```

(screen #10)