Lab Exercise 4- Signed Commits in Git and GitHub

Objective:

To configure Git to sign commits with GPG, push them to GitHub, and verify commit authenticity for secure code contribution.

Prerequisites:

- Git installed on your system
- GPG (GNU Privacy Guard) installed and configured
- GitHub account with a repository (you own or have write access to)
- Basic knowledge of Git commands

Step 1 – Generate or Use an Existing GPG Key

1. **Check for existing keys**

```
gpg --list-secret-keys --keyid-format=long
```

2. **If no key exists, generate a new one**

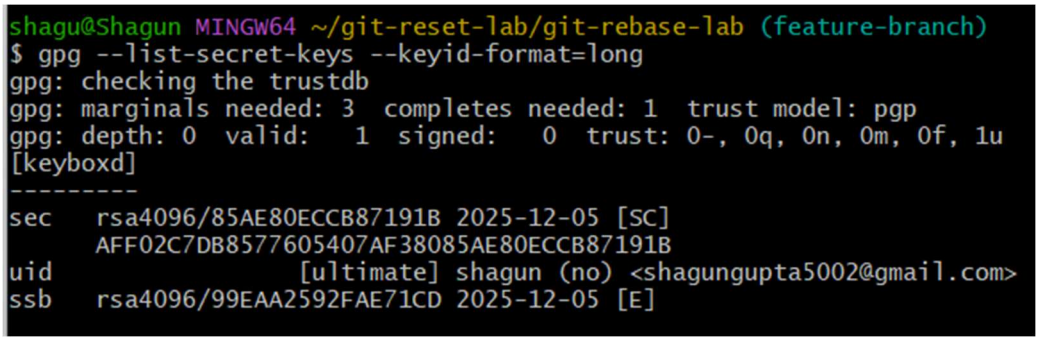
```
gpg --full-generate-key
```

   - Select **RSA and RSA**
   - Key size: **4096**
   - Expiration: **0** (never) or a fixed date
   - Enter your **GitHub-registered name and email**

3. **Get your key ID**

```
gpg --list-secret-keys --keyid-format=long
```

Output:



Step 2 – Add GPG Key to GitHub

1. Export your public key:

```
gpg --armor --export YOUR_KEY_ID
```

2. Copy the output.

3. Go to **GitHub → Settings → SSH and GPG Keys → New GPG Key**.

4. Paste your key and save.

Step 3 – Configure Git for Signed Commits

1. Tell Git which key to use:

```
git config --global user.signingkey YOUR_KEY_ID
```

2. Enable signing for all commits:

```
git config --global commit.gpgsign true
```

Step 4 – Make a Signed Commit

1. Clone your repo (or use an existing one):

```
git clone https://github.com/<username>/<repository>.git

cd <repository>
```

2. Edit or create a file:

```
echo "Secure commit test" >> secure.txt
```

```
git add secure.txt
```

3. Commit with signing:

```
git commit -S -m "Add secure commit test file"
```

4. Enter your GPG passphrase when prompted.

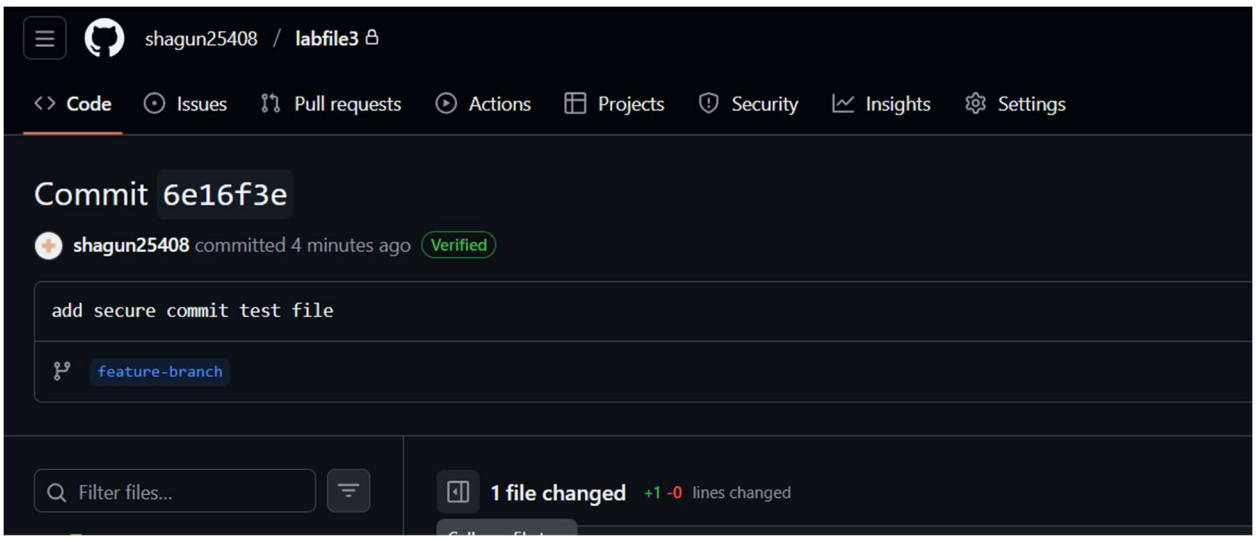
Step 5 – Push and Verify on GitHub

1. Push the commit:

```
git push origin main
```

2. Go to your repository on GitHub → Click the commit → You should see a **green "Verified" badge**.

Output:

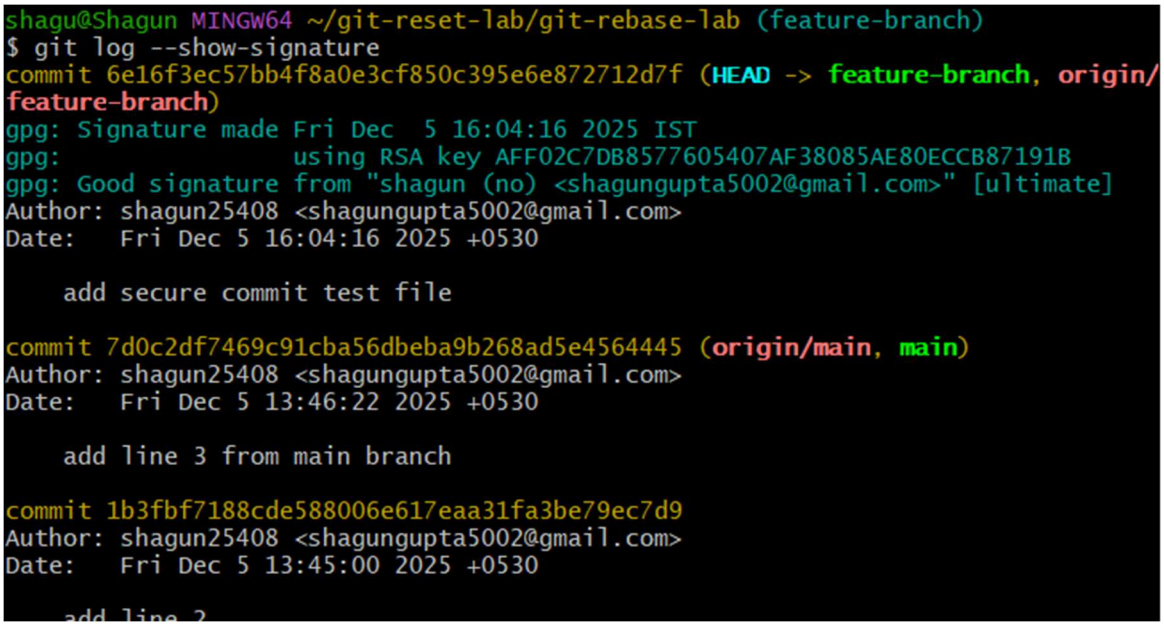


Step 6 – Local Verification of Commit

```
git log --show-signature
```

This will display the GPG verification details locally.

Output:



Use Case

Signed commits prevent identity spoofing in collaborative projects, ensuring only verified authors can make trusted changes in critical codebases.