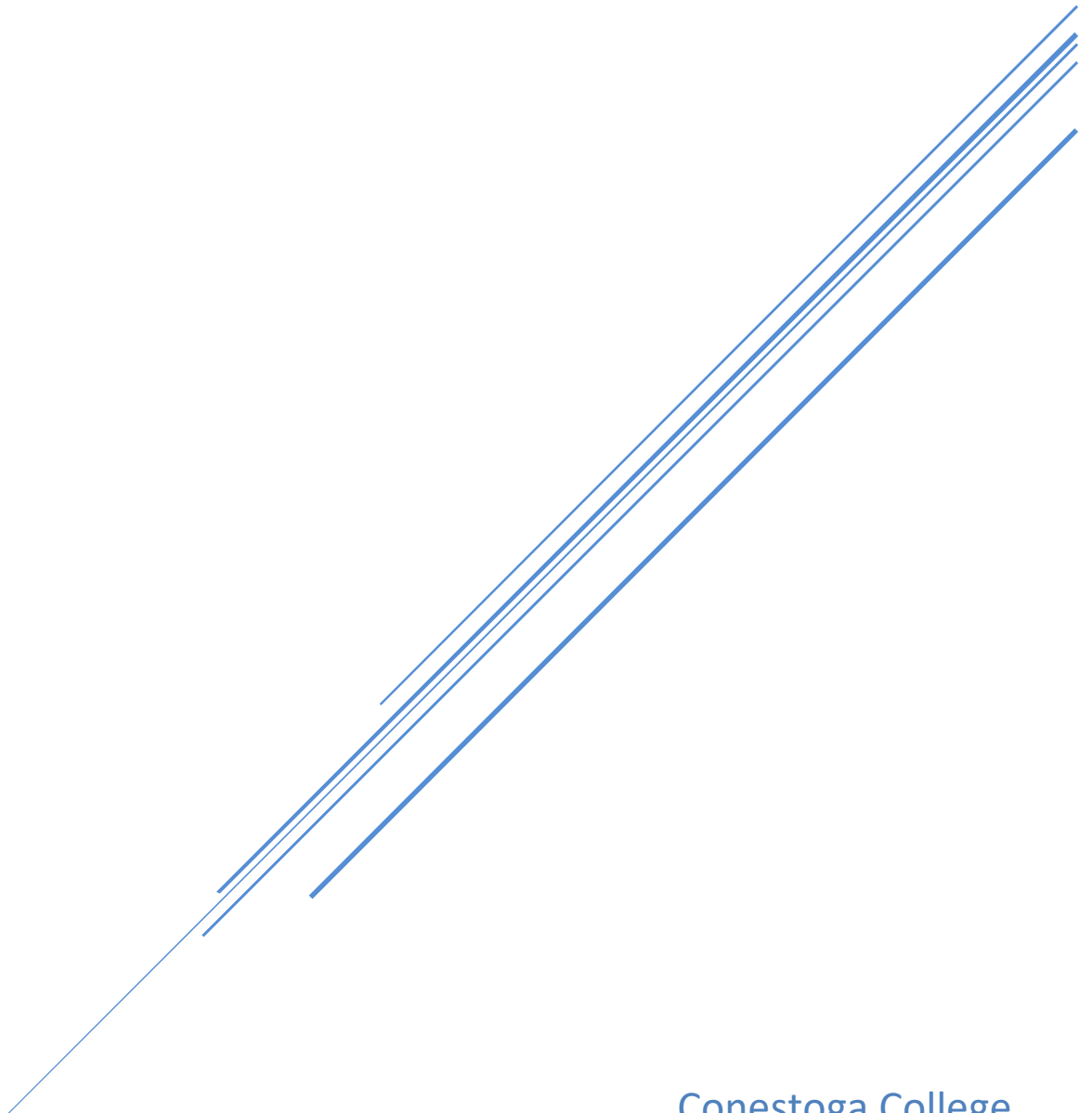


Assignment 2: Securing Communication with GPG

Name: Twinkle Akhilesh Mishra

Student Id : 8894858



Conestoga College	
Course	INFO8965: Computer and Network Security
Activity	Securing Communication with GPG
Student Name	Twinkle Akhilesh Mishra
Date performed	04-March-2025

Objectives

- Security transfer data using public key cryptography
- Authenticate Message Integrity and Sender Identity
- Manage and Exchange Cryptographic Keys

Resources

- PC / Laptop
- Virtual Box: <https://www.virtualbox.org/wiki/Downloads>
- Linux Kali
- GPG (GNU Privacy Guard), SSH

Note: In this lab activity, replace 'FirstName' with your actual first name (Twinkle), 'LastName(Mishra)' with your actual last name, and 'XYZ(858)' with the last three digits of your student ID.

(A) Create clones

-

(B) Create GPG Key Pair

- Page 2 of 2

```
twinkle858@vbox: ~  
File Actions Edit View Help  
GnuPG needs to construct a user ID to identify your key.  
  
Real name: Twinkle858  
Email address: twink858@domain.com  
You selected this USER-ID:  
"Twinkle858 <twink858@domain.com>"  
  
Change (N)ame, (E)mail, or (O)kay/(Q)uit? o  
We need to generate a lot of random bytes. It is a good idea to perform  
some other action (type on the keyboard, move the mouse, utilize the  
disks) during the prime generation; this gives the random number  
generator a better chance to gain enough entropy.  
gpg: agent_genkey failed: Timeout  
Key generation failed: Timeout  
  
[twinkle858@vbox: ~]  
$ gpg --gen-key  
gpg (GnuPG) 2.2.43 Copyright (C) 2024 g10 Code GmbH  
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 dialog.  
GnuPG needs to construct a user ID to identify your key.  
  
Real name: Twinkle858  
Email address: twink858@domain.com  
You selected this USER-ID:  
"Twinkle858 <twink858@domain.com>"  
  
Change (N)ame, (E)mail, or (O)kay/(Q)uit? o  
We need to generate a lot of random bytes. It is a good idea to perform  
some other action (type on the keyboard, move the mouse, utilize the  
disks) during the prime generation; this gives the random number  
generator a better chance to gain enough entropy.  
We need to generate a lot of random bytes. It is a good idea to perform  
some other action (type on the keyboard, move the mouse, utilize the  
disks) during the prime generation; this gives the random number  
generator a better chance to gain enough entropy.  
gpg: /home/twinkle/.gnupg/trustdb.gpg: trustdb created  
gpg: directory '/home/twinkle/.gnupg/openpgp-revocs.d' created  
gpg: revocation certificate stored as '/home/twinkle/.gnupg/openpgp-revocs.d/41A89E66180988E7DBA35EBFE696218D888E62B1.rev'  
public and secret key created and signed.  
  
pub rsa3072 2025-03-01 [SC] [expires: 2028-02-29]  
41A89E66180988E7DBA35EBFE696218D888E62B1  
uid  
Twinkle858 <twink858@domain.com>  
sub rsa3072 2025-03-01 [E] [expires: 2028-02-29]  
  
[twinkle858@vbox: ~]
```

➤ On the Receiver VM (Mishra858)

- Open the terminal and type:
gpg --gen-key
- Enter the following details when prompted:
 - Name: Mishra858
 - Email: mishra.858@domain.com
 - Algorithm: RSA and RSA (default)
 - Key Size: 4096
 - Expiration: Press Enter (default).
 - Passphrase: Set a secure passphrase.
- Wait for the system to generate the key.

The screenshot shows a terminal window titled 'mishra858 [Running] - Oracle VM VirtualBox'. The user is at the 'twinkle@vbox: ~' prompt. They have run 'gpg --gen-key', which has prompted them for their real name, email address, and user ID. The user has entered 'Mishra858', 'mishra.858@domain.com', and 'Mishra858 <mishra.858@domain.com>'. The terminal shows the progress of key generation, including a warning about entropy and a confirmation that the key pair has been created. The output of the key generation is as follows:

```
pub rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
3EDBE8AAFAFD4B2E8CC8D3F895FC0423D3504
uid Mishra858 <mishra.858@domain.com>
sub rsa3072 2025-03-01 [E] [expires: 2028-02-29]
```

An arrow points from the 'pub' line to a text box that says: 'GPG key pair with RSA encryption and signature features was successfully created for Mishra858 (Receiver)'.

(C) List and Export Public Keys

On both VMs, list the keys with:

gpg --list-keys

Capture a screenshot of each key list and add it to **Output # 2**.

The screenshot shows a terminal window titled 'twinkle858 (Linked Base for twinklexyz and mishra858) [Running] - Oracle VM VirtualBox'. The user is at the 'twinkle@vbox: ~' prompt. They have run 'gpg --list-key', which has prompted them to check the trust database. The output of the command is as follows:

```
pub rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
41A896661809886708435E8F696218D888E62B1
uid [ultimate] Twinkle858 <twinkle.858@domain.com>
sub rsa3072 2025-03-01 [E] [expires: 2028-02-29]
```

```
twinkle@vbox:~$ gpg --list-key
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 2 signed: 0 trust: 0-, 0q, 0n, 0f, 2u
gpg: next trustdb check due at 2025-02-29
/home/twinkle/.gnupg/pubring.kbx

pub rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
+1AB9E66108988E70B435BF896218D688E62B1
uid [ultimate] Twinkle858 <twinkle.858@domain.com>
sub rsa3072 2025-03-01 [E] [expires: 2028-02-29]

pub rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
3EDBEAAFAFAFD4B2ECCB03FA95FCB423D3504
uid [ultimate] Mishra858 <mishra.858@domain.com>
sub rsa3072 2025-03-01 [E] [expires: 2028-02-29]
```

twinkle@vbox:~\$

↓

GPG key listing on Twinkle858 (Sender) and Mishra858 (Receiver), confirming successful key generation.

On the Sender VM, export the public key:

`gpg --export --armor --output sender_pubkey.asc "Twinkle858"`

Transfer `sender_pubkey.asc` to the Receiver VM using SCP (Secure Copy Protocol) command. Refer to online resources for more information about SCP command.

```
twinkle@vbox:~$ gpg --export --armor --output sender_pubkey.asc "Twinkle858"
File 'sender_pubkey.asc' exists. Overwrite? (y/n) y

twinkle@vbox:~$ ls -l sender_pubkey.asc
-rw-rw-r-- 1 twinkle twinkle 2456 Mar  4 13:43 sender_pubkey.asc

twinkle@vbox:~$ scp sender_pubkey.asc twinkle858:/home/twinkle/
```

→

Exported the GPG public key from Twinkle858 (Sender) to sender_pubkey.asc

The screenshot shows a terminal window for a VM named 'twinkle858'. The user is at the prompt 'twinkle@vbox: ~'. They run the command 'scp sender_pubkey.asc twinkler192.168.0.199:/home/twinkle'. The terminal output shows the SSH connection process, including host key fingerprint warnings and a confirmation to add the host to the known hosts list. The transfer progress bar shows 100% completion. An arrow points from the command line to a text box on the right.

sender_pubkey.asc was successfully transferred over SCP from Twinkle858 (Sender) to Mishra858 (Receiver)

On the Receiver VM, export the public key:

```
gpg --export --armor --output receiver_pubkey.asc "Mishra858"
```

Transfer scp receiver_pubkey.asc to the Sender VM using SCP command.

The screenshot shows a terminal window for a VM named 'twinkle858'. The user is at the prompt 'twinkle@vbox: /home/twinkle'. They run the command 'gpg --export --armor --output receiver_pubkey.asc "Mishra858"'. The terminal output shows the key export process. Then, they run the command 'scp receiver_pubkey.asc twinkler192.168.0.198:/home/twinkle'. The terminal output shows the SSH connection process, including host key fingerprint warnings and a confirmation to add the host to the known hosts list. The transfer progress bar shows 100% completion. An arrow points from the command line to a text box on the right.

Using SCP, receiver_pubkey.asc has been successfully transferred from Mishra858 (Receiver) to Twinkle858 (Sender)

(D) Import public keys

Output#3

- On the Sender VM, import the Receiver's public key:
 - `gpg --import receiver_pubkey.asc`
- Verify the imported key with:
 - `gpg --list-keys`

```
(twinkle@vbox)-[~]
$ gpg --import receiver_pubkey.asc
gpg: key F095FC0423D3504: public key "Mishra858 <mishra.858@domain.com>" imported
gpg: Total number processed: 1
gpg:      imported: 1

(twinkle@vbox)-[~]
$ gpg --list-keys
/home/twinkle/.gnupg/pubring.kbx

pub   rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
41A89E66180988E7D0A35E8F696218D888E62B1
uid   [ultimate] Twinkle858 <twinkle.858@domain.com>
sub   rsa3072 2025-03-01 [E] [expires: 2028-02-29]

pub   rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
3ED8E0AFAAFD4B2E8CC8D3F995FC0423D3504
uid   [unknown] Mishra858 <mishra.858@domain.com>
sub   rsa3072 2025-03-01 [E] [expires: 2028-02-29]

(twinkle@vbox)-[~]
```

Imported receiver_pubkey.asc into Twinkle858's GPG keyring successfully

On the Receiver VM, import the Sender's public key:

- `gpg --import sender_pubkey.asc`

Verify the imported key:

- `gpg --list-keys`

```
(twinkle@vbox)-[~]
$ gpg --import sender_pubkey.asc
gpg: key E096218D888E62B1: "Twinkle858 <twinkle.858@domain.com>" not changed
gpg: Total number processed: 1
gpg:      unchanged: 1

(twinkle@vbox)-[~]
$ gpg --list-keys
option "--list" is ambiguous

(twinkle@vbox)-[~]
$ gpg --list-keys
/home/twinkle/.gnupg/pubring.kbx

pub   rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
41A89E66180988E7D0A35E8F696218D888E62B1
uid   [ultimate] Twinkle858 <twinkle.858@domain.com>
sub   rsa3072 2025-03-01 [E] [expires: 2028-02-29]

pub   rsa3072 2025-03-01 [SC] [expires: 2028-02-29]
3ED8E0AFAAFD4B2E8CC8D3F995FC0423D3504
uid   [ultimate] Mishra858 <mishra.858@domain.com>
sub   rsa3072 2025-03-01 [E] [expires: 2028-02-29]

(twinkle@vbox)-[~]
```

Imported sender_pubkey.asc into Mishra858's GPG keyring successfully

(E) Sign Receiver's public key

- On the Sender VM, sign the Receiver's public key to verify its authenticity:
 - `gpg --sign-key "Mishra858"`
- Capture a screenshot showing that the key has been signed and add it to **Output # 4**.


```
(twinkle@vbox):~$ gpg --sign-key "Mishra858"
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 1 signed: 1 trust: 0-, 0q, 0n, 0e, 0f, 1u
gpg: depth: 1 valid: 1 signed: 0 trust: 1-, 0q, 0n, 0e, 0f, 0u
gpg: next trustdb check due at 2028-02-29
pub rsa3072/F095FC00423D3584
   created: 2025-03-01 expires: 2028-02-29 usage: SC
   trust: unknown validity: full
sub rsa3072/0F47537718BF813E
   created: 2025-03-01 expires: 2028-02-29 usage: E
   [ full ] (1). Mishra858 <mishra.858@domain.com>

"Mishra858 <mishra.858@domain.com>" was already signed by key E696218D0808E62B1
Nothing to sign with key E696218D0808E62B1
Key not changed so no update needed.

(twinkle@vbox):~$
```

Successfully signed Mishra858's public key on Twinkle858 to check authenticity.

On the Receiver VM, sign the Sender's public key:

- **gpg --sign-key "Twinkle858"**

Capture a screenshot showing the signed key and add it to **Output # 4**.

```
(twinkle@vbox):~$ gpg --sign-key "Twinkle858"
sec rsa3072/E696218D0808E62B1
   created: 2025-03-01 expires: 2028-02-29 usage: SC
   trust: ultimate validity: ultimate
ssb rsa3072/76C1796E2B71E54F
   created: 2025-03-01 expires: 2028-02-29 usage: E
   [ultimate] (1). Twinkle858 <twinkle.858@domain.com>

"Twinkle858 <twinkle.858@domain.com>" was already signed by key E696218D0808E62B1
Nothing to sign with key E696218D0808E62B1
Key not changed so no update needed.

(twinkle@vbox):~$
```

Successfully signed Twinkle858's public key on Mishra858 to verify authenticity.

- Explain the advantage of signing receiver's public key in this context. [2 mark]
- Signing a public key ensures that:
 - ✓ Authenticity is Verified
 - It confirms that the key truly belongs to the claimed identity.
 - Prevents man-in-the-middle attacks where someone intercepts and replaces keys.
 - ✓ Trust Establishment
 - The recipient can see that you have verified and trusted this key.
 - Helps build a web of trust between users.
 - ✓ Prevents Spoofing
 - No one can impersonate the key owner without a valid signature.

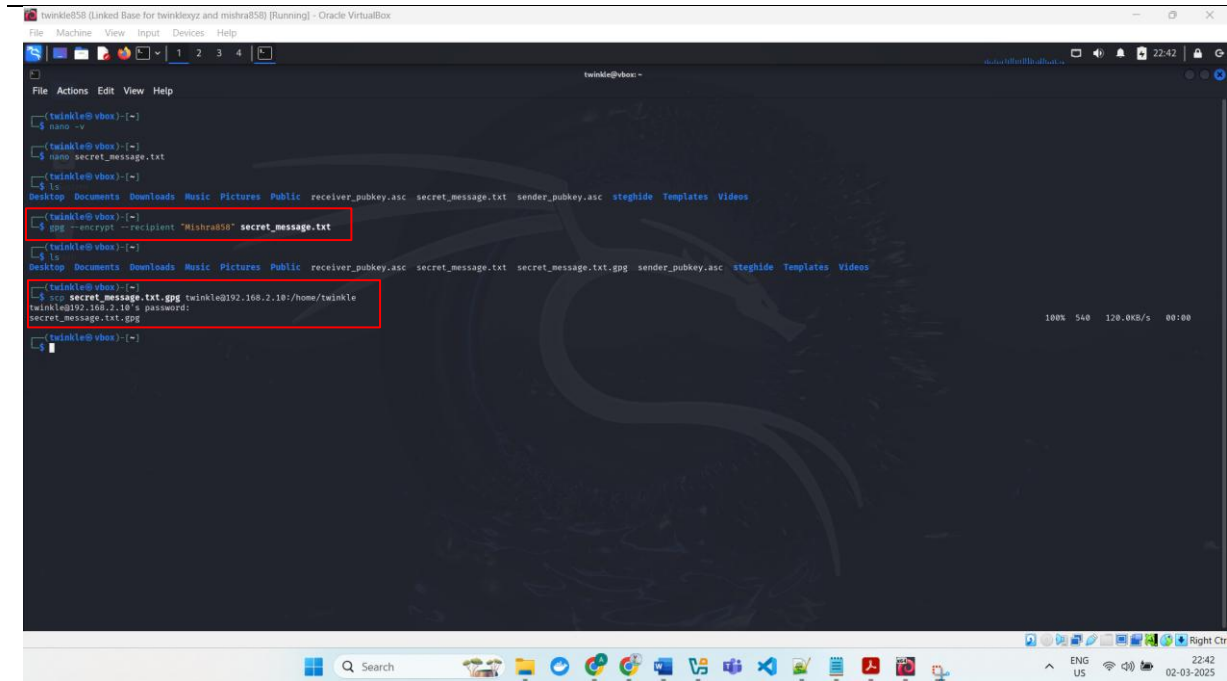
Signing with public key certifies its legitimacy and establishes a trust network. It will help to protect against man-in-the-middle attacks and ensures secure communication between the sender and receiver.

(F) Encrypt and Decrypt a File

On the Sender VM, create a text file named secret_message.txt. Add some confidential message to this file. Encrypt the file using the Receiver's public key:

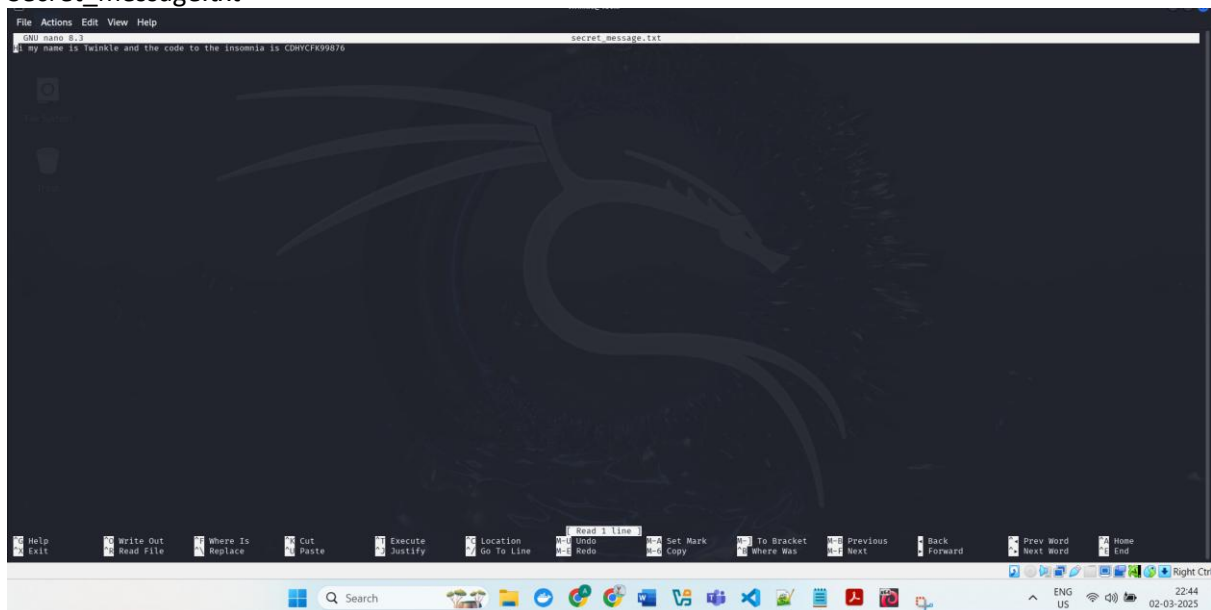
- **gpg --encrypt --recipient "Mishra858" secret_message.txt**

This will create an encrypted file secret_message.txt.gpg. Capture a screenshot of the encrypted file and add it to **Output # 5**.



```
twinkle@vbox:~$ nano -v
twinkle@vbox:~$ nano secret_message.txt
twinkle@vbox:~$ ls
Desktop Documents Downloads Music Pictures Public receiver_pubkey.asc secret_message.txt sender_pubkey.asc steghide Templates Videos
twinkle@vbox:~$ gpg --encrypt --recipient "Mishra858" secret_message.txt
twinkle@vbox:~$ ls
Desktop Documents Downloads Music Pictures Public receiver_pubkey.asc secret_message.txt secret_message.txt.gpg sender_pubkey.asc steghide Templates Videos
twinkle@vbox:~$ scp secret_message.txt.gpg twink1e@192.168.2.10:/home/twinkle
twink1e@192.168.2.10's password:
secret_message.txt.gpg
100% 540 120.0KB/s 00:00
```

Secret_message.txt



```
File Actions Edit View Help
GNU nano 2.9.3 secret_message.txt
My name is Twinkle and the code to the insomnia is CDMYCFK99876
```

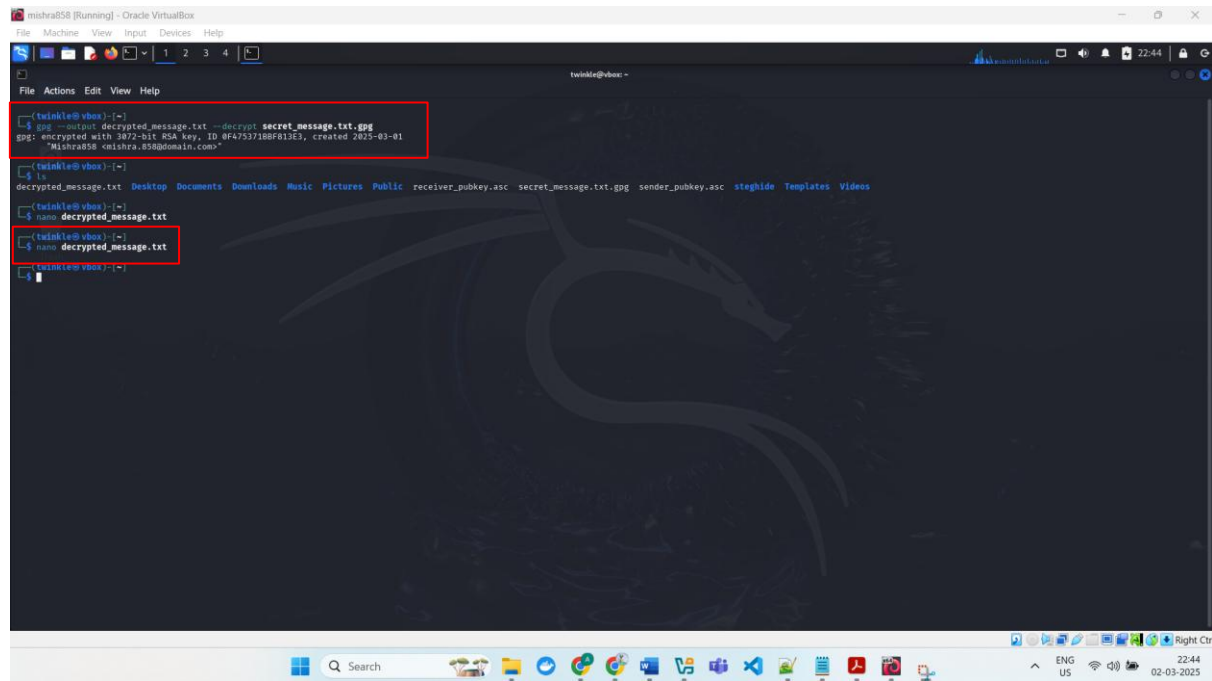
Transfer `secret_message.txt.gpg` from the Sender VM to the Receiver VM using SCP.

On the Receiver VM, decrypt the file with the command:

- `gpg --output decrypted_message.txt --decrypt secret_message.txt.gpg`

Capture a screenshot of the decrypted file content and add it to **Output # 5** [2 marks]

Receiver VM



The screenshot shows a terminal window titled "twinkle@vbox: ~" with a dark background and a dragon logo. The terminal displays the following commands and output:

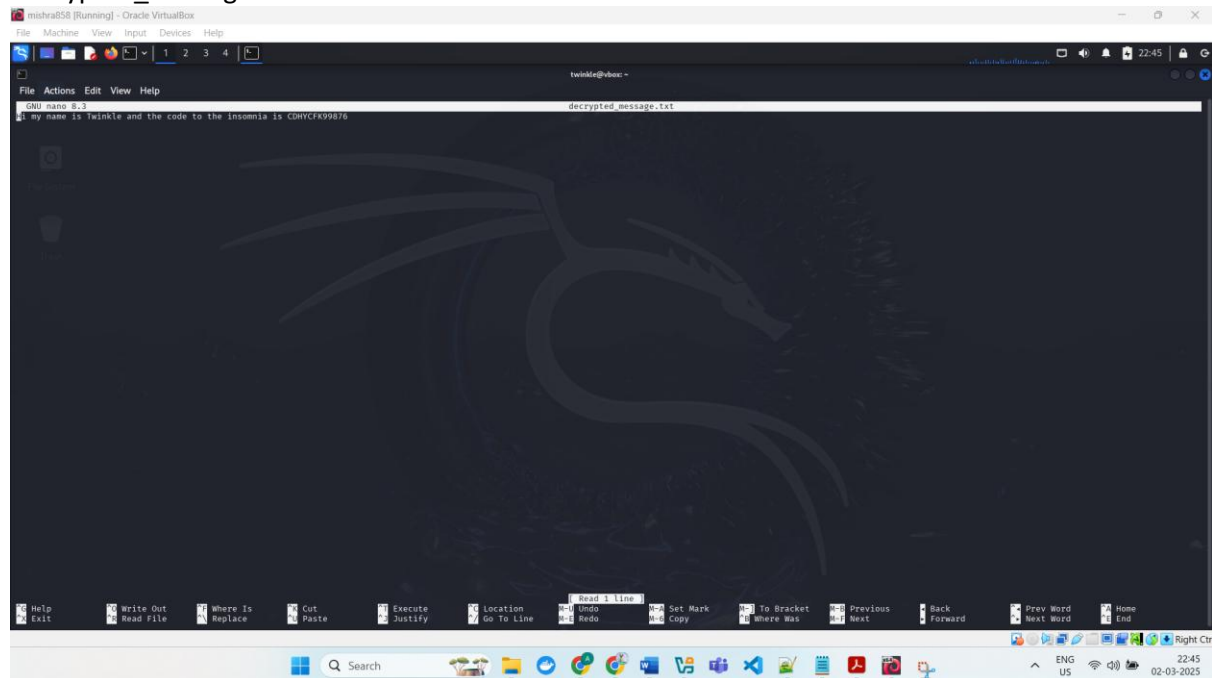
```
twinkle@vbox:~$ gpg --output decrypted_message.txt --decrypt secret_message.txt.gpg
gpg: encrypted with 2072-bit RSA key, ID 0F475716B613E3, created 2025-03-01
"mishra858-mishra-858@gmail.com"

twinkle@vbox:~$ ls
decrypted_message.txt  Desktop  Documents  Downloads  Music  Pictures  Public  receiver_pubkey.asc  secret_message.txt.gpg  sender_pubkey.asc  steghide  Templates  Videos

twinkle@vbox:~$ nano decrypted_message.txt
twinkle@vbox:~$ nano decrypted_message.txt
twinkle@vbox:~$
```

The Windows taskbar at the bottom shows the search bar, taskbar icons, and system tray with the date 02-03-2025 and time 22:44.

Decrypted_message.txt



The screenshot shows a terminal window titled "twinkle@vbox: ~" with a dark background and a dragon logo. The terminal displays the following content:

```
GNU nano 2.9.3 decrypted_message.txt
my name is twinkle and the code to the insomnia is CDHYCFK99870
```

The terminal window includes a menu bar with options like Help, Write Out, Where Is, Cut, Paste, Execute, Location, Go To Line, Undo, Redo, Set Mark, Copy, To Bracket, Previous Next, Back Forward, Prev Word, Next Word, and Home End. The Windows taskbar at the bottom shows the search bar, taskbar icons, and system tray with the date 02-03-2025 and time 22:45.

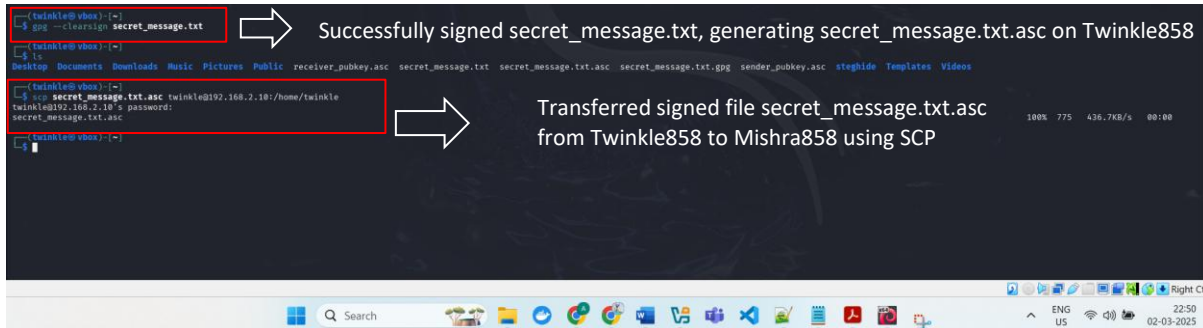
(F) Verify File Signature

- On the Sender VM, sign the secret_message.txt file:

- **gpg --clearsign secret_message.txt**

Transfer the signed file secret_message.txt.asc to the Receiver VM.

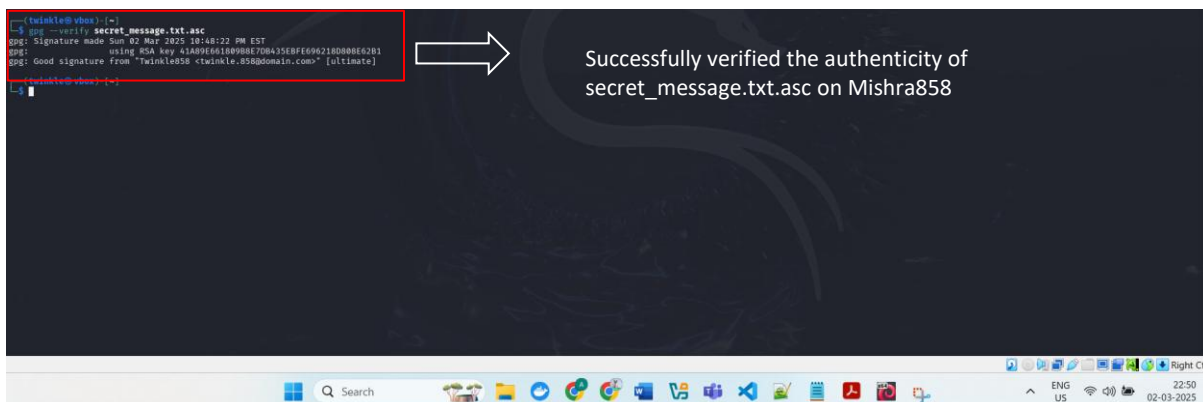
Sender VM



- On the Receiver VM, verify the signature:

- **gpg --verify secret_message.txt.asc**

Capture a screenshot showing successful verification of the signature and add it to **Output # 6**.



- Explain the purpose and reason of verifying file signatures in this context. [2 marks]

The following are the purpose and reason of verifying file signatures:

- ✓ Ensures File Integrity
 - Verifying the signature confirms that file has not been modified during transfer.
- ✓ Authenticates the Sender
 - It ensures that Twinkle858 (Sender) is the actual creator of the signed file.
- ✓ Preventing Attacks and Tampering
 - If the file is not verified, the attacker may change or replace it without anyone knowledge.

Verifying file signatures ensures the sender's validity as well as the file's integrity. It avoids manipulation and certifies that the file was not changed during transmission.
