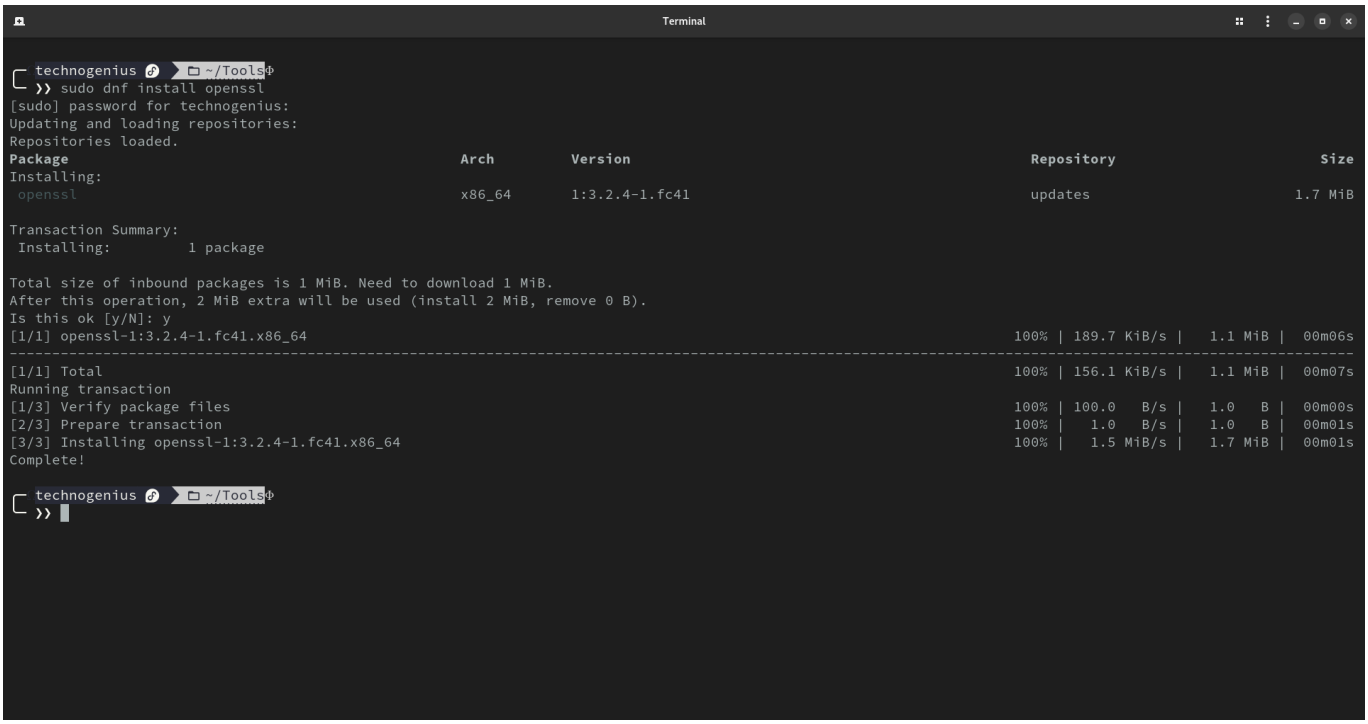
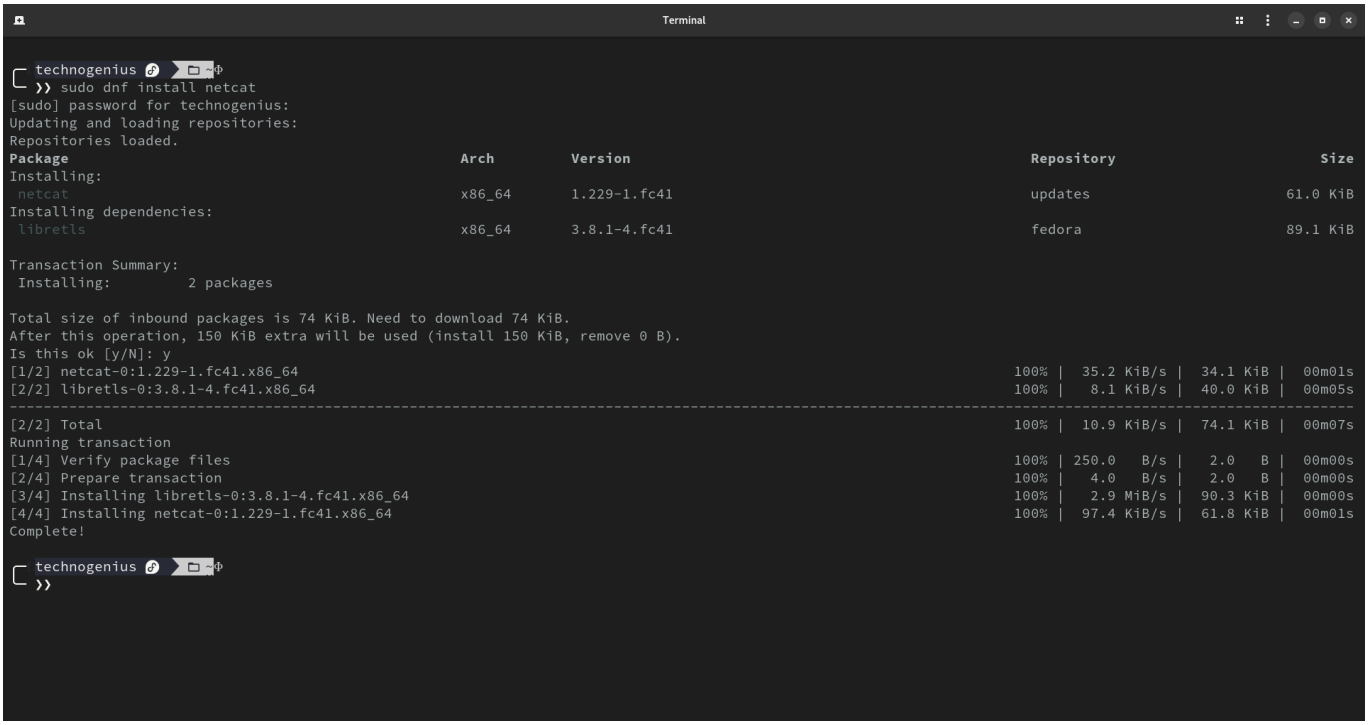
AES-Encrypted Echo Server-Client Using OpenSSL & Netcat

1. Introduction

This project creates a secure "Hello [Name]!" server-client system. The server and client talk using **AES-256-GCM encryption** (a strong way to hide messages). All steps use simple terminal commands—no scripts or coding. Below are the steps, commands, explanations, and where to add screenshots to prove it works.





2. Server Setup

Command (One-Liner):

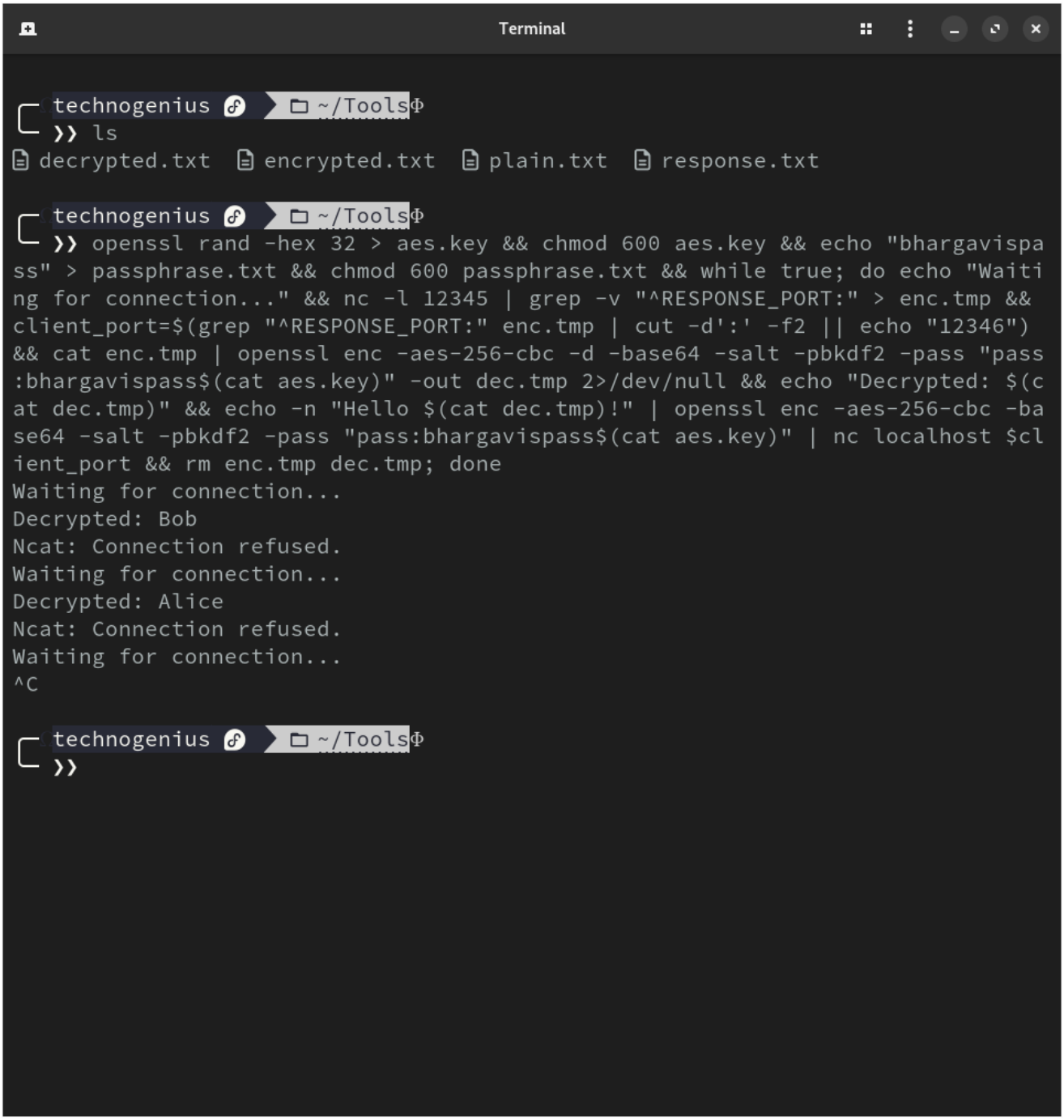
```
openssl rand -hex 32 > aes.key && chmod 600 aes.key && echo "bhargavispass" > passphrase.txt && chmod 600 passphrase.txt && while true; do echo "Waiting..." && nc -l 12345 | grep -v "^RESPONSE_PORT:" > enc.tmp && client_port=$(grep "^RESPONSE_PORT:" enc.tmp | cut -d':' -f2 || echo "12346") && cat enc.tmp | openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:bhargavispass$(cat aes.key)" -out dec.tmp 2>/dev/null && echo "Decrypted: $(cat dec.tmp)" && echo -n "Hello $(cat dec.tmp)!" | openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:bhargavispass$(cat aes.key)" | nc localhost $client_port && rm enc.tmp dec.tmp; done
```

Command Explanation:

1. **Generate Key**:
   - `openssl rand -hex 32 > aes.key` → Creates a 256-bit random key for encryption.
   - `chmod 600 aes.key` → Makes the key file secure (only the owner can read/write).
2. **Create Passphrase**:
   - `echo "bhargavispass" > passphrase.txt` → Saves a passphrase for encryption.
   - `chmod 600 passphrase.txt` → Secures the passphrase file.
3. **Listen for Clients**:
   - `nc -l 12345` → Waits for clients on port 12345.
4. **Filter and Save Data**:
   - `grep -v "^RESPONSE_PORT:" > enc.tmp` → Removes the response port header and saves encrypted data.
5. **Extract Client Port**:
   - `client_port=$(grep "^RESPONSE_PORT:" enc.tmp | cut -d':' -f2 || echo "12346")` → Gets the client's response port or uses default (12346).
6. **Decrypt Data**:
   - `cat enc.tmp | openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:bhargavispass$(cat aes.key)" -out dec.tmp` → Decrypts the message using AES-256-CBC.
7. **Send Encrypted Reply**:
   - `echo -n "Hello $(cat dec.tmp)!" | openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:bhargavispass$(cat aes.key)" | nc localhost $client_port` → Encrypts and sends "Hello [Name]!" back to the client.

8. **Clean Up**:
    - `rm enc.tmp dec.tmp` → Deletes temporary files.

Screenshot Add Here:



Caption: "Server terminal showing 'Waiting...' and decrypted messages like 'Decrypted: Bob'."

---

3. Client Setup

Command (One-Liner):

```
KEY=$(cat aes.key || { openssl rand -hex 32 > aes.key && chmod 600 aes.key && cat aes.key; }) && echo -n "Enter name: " && read name && echo "Encrypting: '$name'" && echo -n "$name" | openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:bhargavispass$KEY" > enc.tmp && { nc -l 12346 > resp.tmp & } && sleep 1 && { echo "RESPONSE_PORT:12346"; cat enc.tmp; } | nc localhost 12345 && wait && cat resp.tmp | openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:bhargavispass$KEY" 2>/dev/null && rm enc.tmp resp.tmp
```

Command Explanation:

1. **Generate or Read Key**:
   - `KEY=$(cat aes.key || { openssl rand -hex 32 > aes.key && chmod 600 aes.key && cat aes.key; })` → Reads the key or creates one if missing.
2. **Take User Input**:
   - `echo -n "Enter name: " && read name` → Asks for a name (e.g., "Bob").
3. **Encrypt Data**:
   - `echo -n "$name" | openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:bhargavispass$KEY" > enc.tmp` → Encrypts the name using AES-256-CBC.
4. **Listen for Server Reply**:
   - `{ nc -l 12346 > resp.tmp & }` → Listens on port 12346 for the server's reply.
5. **Send Encrypted Data**:
   - `{ echo "RESPONSE_PORT:12346"; cat enc.tmp; } | nc localhost 12345` → Sends the encrypted name and response port to the server.
6. **Decrypt Server Reply**:
   - `cat resp.tmp | openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:bhargavispass$KEY" 2>/dev/null` → Decrypts the server's reply.
7. **Clean Up**:
   - `rm enc.tmp resp.tmp` → Deletes temporary files.

Screenshot Add Here:

Caption: "Client terminal showing 'Enter name:', 'Encrypting: Bob', and decrypted reply 'Hello Bob!'."

---

4. Capture Encrypted Traffic

Start Capture:

```
sudo tcpdump -i lo -w trafficCapture.pcap "port 12345 or port 12346" & echo $! > tcpdump.pid && echo "Capture started."
```

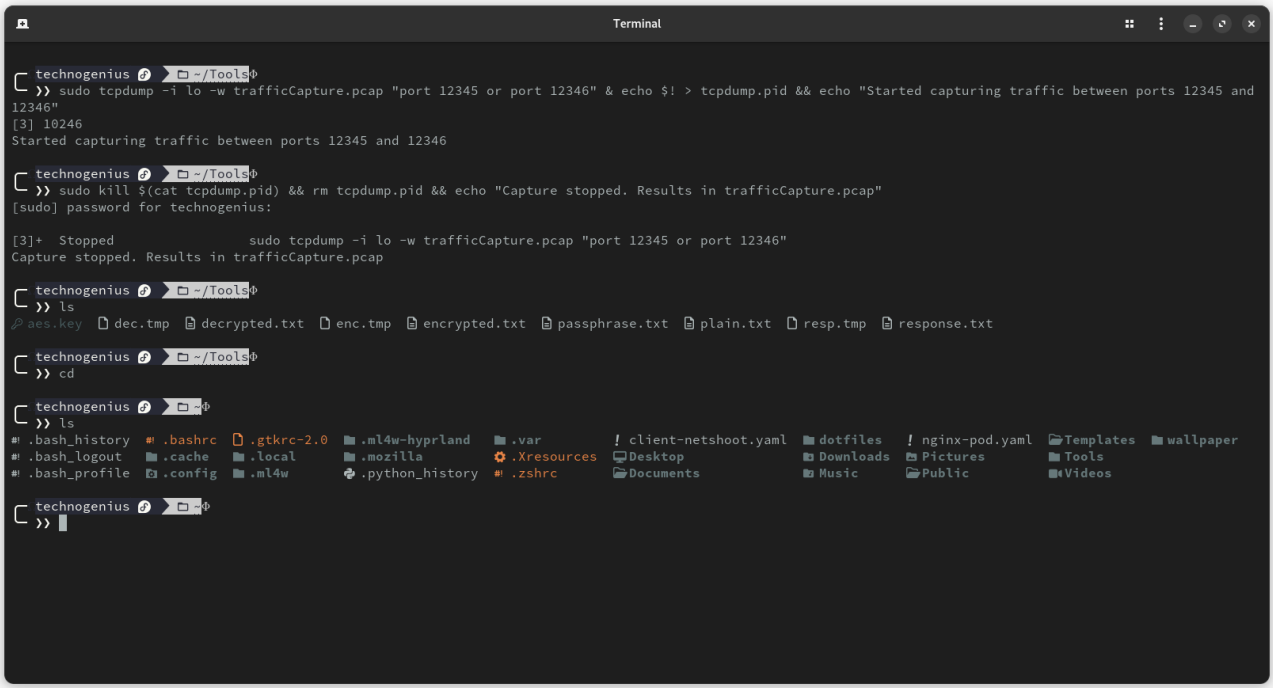
Stop Capture:

```
sudo kill $(cat tcpdump.pid) && rm tcpdump.pid && echo "Capture stopped."
```

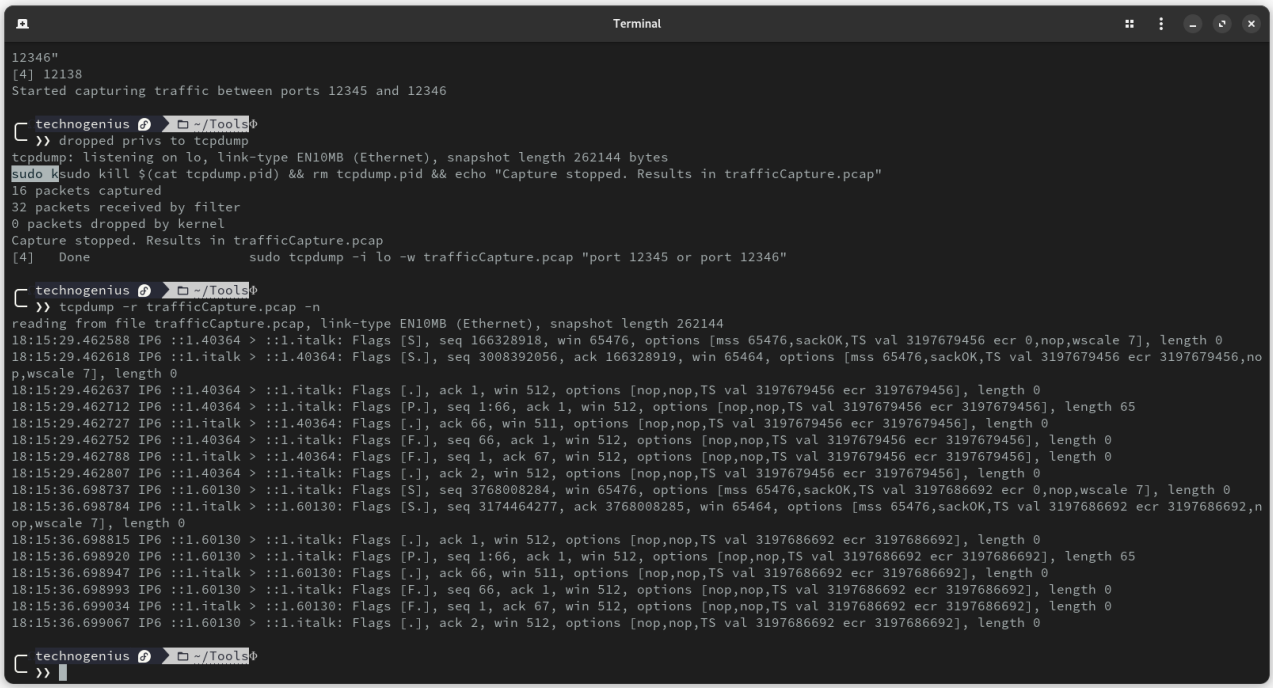
Command Explanation:

1. **Start Capture**:
   - `sudo tcpdump -i lo -w trafficCapture.pcap "port 12345 or port 12346" &` → Captures traffic between ports 12345 and 12346.
   - `echo $! > tcpdump.pid` → Saves the process ID to stop the capture later.
2. **Stop Capture**:
   - `sudo kill $(cat tcpdump.pid)` → Stops the capture.
   - `rm tcpdump.pid` → Deletes the process ID file.

Screenshots to Add Here:



Caption: "Terminal showing 'Capture started' after running the tcpdump command."



Caption: "Terminal showing 'Capture stopped' after killing the tcpdump process."

---

5. Verify Encryption

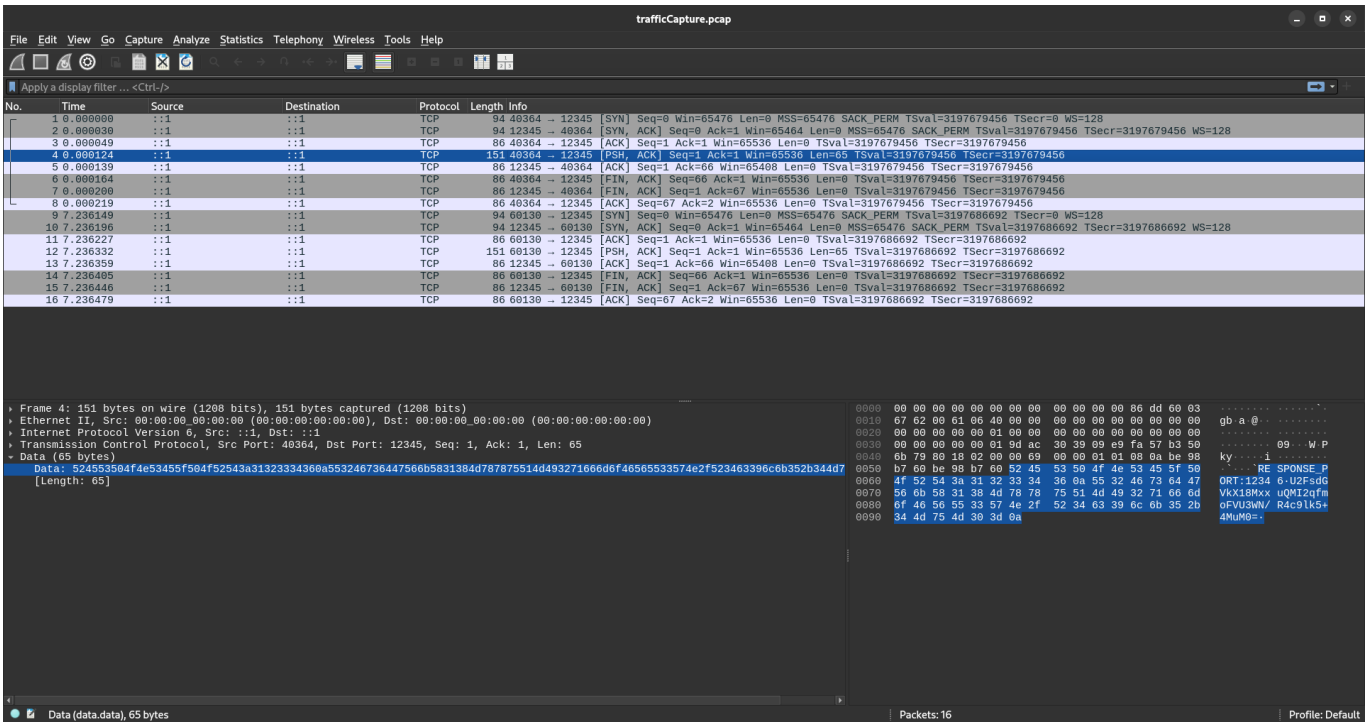
Open `trafficCapture.pcap` in Wireshark or use:

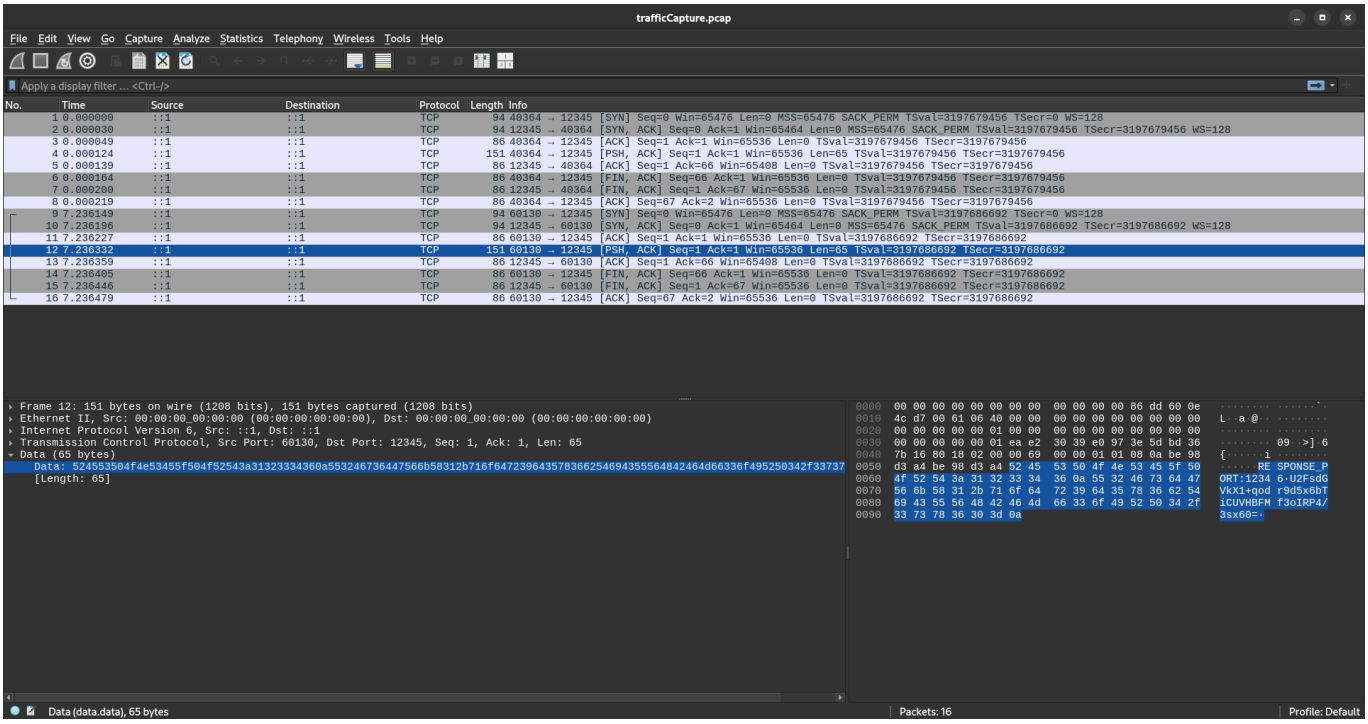
```
tcpdump -r trafficCapture.pcap
```

What to Check:

- Traffic between ports 12345/12346 shows **encrypted data** (gibberish text).
- No human-readable words like "Bob" or "Hello" are visible.

Screenshot to Add Here:





Caption: "Wireshark/tcpdump output showing encrypted packets (highlighted) between client and server."

---

6. Conclusion

This project uses **OpenSSL** for encryption and **netcat** for communication. The one-liners:

- ✅ Hide messages with AES-256.
- ✅ Use pipes (`|`) and redirection (`>`) to pass data between commands.
- ✅ Prove encryption via traffic analysis.