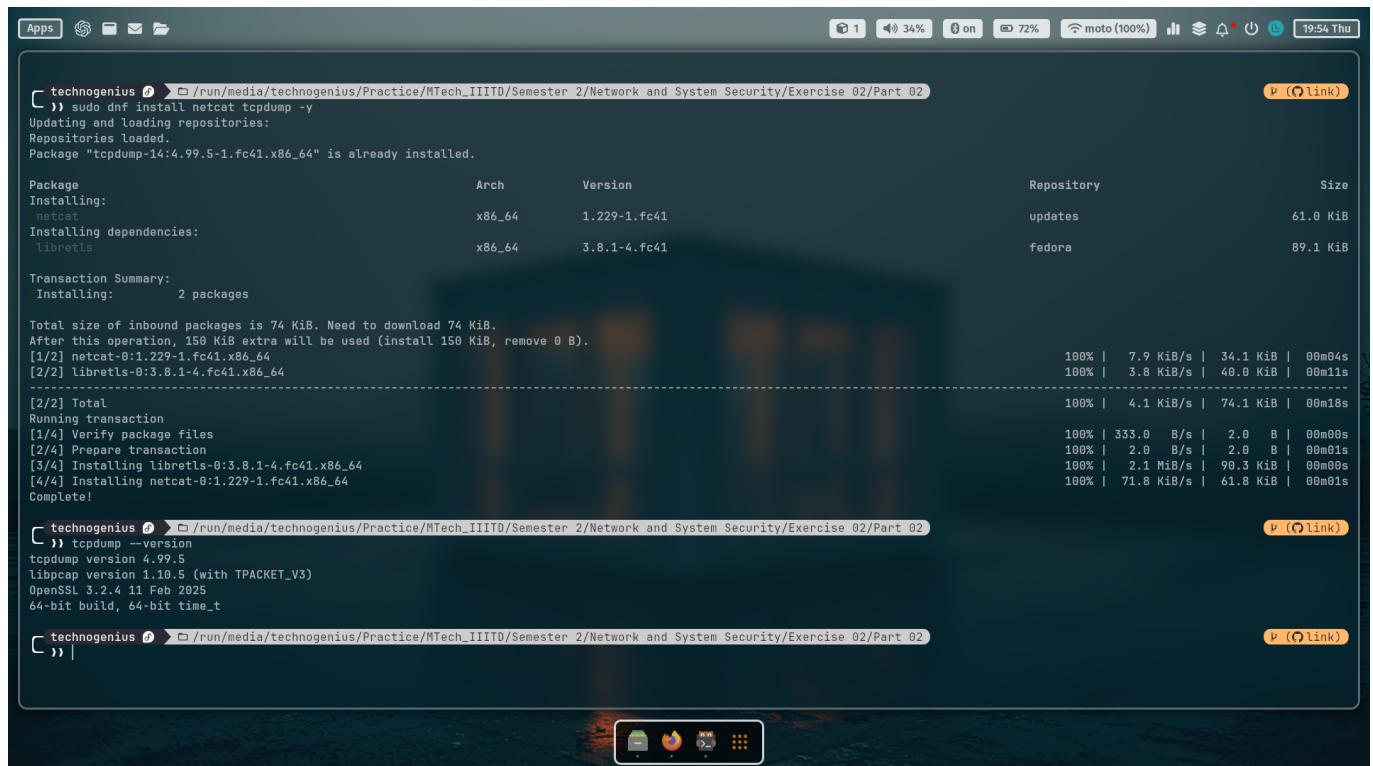


## AES-Enabled Echo Server and Client Using OpenSSL and Netcat

This report provides an in-depth explanation of the commands used in the provided shell scripts to build an AES-enabled echo server and client. The goal is to encrypt communication between the client and server using **AES-256 encryption** and capture the encrypted traffic using **tcpdump**. Additionally, this report includes advanced concepts, detailed descriptions, and placeholders for output images to help you understand the process better.

```
C technogenius ⚡ ▶ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
)) sudo dnf install openssl
[sudo] password for technogenius:
Updating and loading repositories:
Repositories loaded.
Package                                     Arch      Version          Repository      Size
Installing:
openssl                                     x86_64    1:3.2.4-1.fc41   updates           1.7 MiB
Transaction Summary:
Installing:           1 package
Total size of inbound packages is 1 MiB. Need to download 1 MiB.
After this operation, 2 MiB extra will be used (install 2 MiB, remove 0 B).
Is this ok [y/N]: y
[1/1] openssl-1:3.2.4-1.fc41.x86_64
-----[1/1] Total
Running transaction
[1/3] Verify package files
[2/3] Prepare transaction
[3/3] Installing openssl-1:3.2.4-1.fc41.x86_64
Complete!
C technogenius ⚡ ▶ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
)) |
```



```
C technogenius ⚡ ▶ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
)) sudo dnf install netcat tcpdump -y
Updating and loading repositories:
Repositories loaded.
Package "tcpdump-14:4.99.5-1.fc41.x86_64" is already installed.
Package                                     Arch      Version          Repository      Size
Installing:
netcat                                     x86_64    1.229-1.fc41   updates           61.0 KiB
Installing dependencies:
libretlts                                    x86_64    3.8.1-4.fc41   fedora            89.1 KiB
Transaction Summary:
Installing:           2 packages
Total size of inbound packages is 74 KiB. Need to download 74 KiB.
After this operation, 150 KiB extra will be used (install 150 KiB, remove 0 B).
[1/2] netcat-0:1.229-1.fc41.x86_64
[2/2] libretlts-0:3.8.1-4.fc41.x86_64
-----[2/2] Total
Running transaction
[1/4] Verify package files
[2/4] Prepare transaction
[3/4] Installing libretlts-0:3.8.1-4.fc41.x86_64
[4/4] Installing netcat-0:1.229-1.fc41.x86_64
Complete!
C technogenius ⚡ ▶ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
)) tcpdump --version
tcpdump version 4.99.5
libpcap version 1.10.5 (with TPACKET_V3)
OpenSSL 3.2.4 11 Feb 2025
64-bit build, 64-bit time_
C technogenius ⚡ ▶ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
)) |
```

## 1. Advanced Explanation of Commands

### a. aesClient.sh

```

GNU nano 8.1                               aesClient.sh
#!/bin/bash
# aesClient.sh - Improved AES client with proper connection handling

source aesConfig.sh

if [ ! -f "$KEY_FILE" ]; then
    echo "Error: Key file not found. Run genKey.sh first."
    exit 1
fi

KEY=$(cat "$KEY_FILE")

# Get input
echo -n "Enter your name: "
read name

echo "Encrypting: '$name'"

# Create temporary file for the message
msg_file=$(mktemp)
echo -n "$name" > "$msg_file"

# Encrypt the message using passphrase + key
enc_file=$(mktemp)
openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -in "$msg_file" -out "$enc_file" 2>/dev/null

if [ $? -ne 0 ]; then
    echo "Error: Failed to encrypt the message."
    rm "$msg_file" "$enc_file"
    exit 1
fi

echo "Sending encrypted data to server..."

# Start client listener for response BEFORE sending data
resp_file=$(mktemp)

^G Help      ^O Write Out   ^F Where Is   ^K Cut        ^T Execute   ^C Location   M-U Undo   M-A Set Mark   M-J To Bracket   M-B Previous   . Back
^X Exit      ^R Read File   ^R Replace   ^U Paste     ^J Justify   ^L Go To Line  M-E Redo   M-C Copy     M-B Where Was  M-F Next    ^ . Forward

```

```

GNU nano 8.1                               aesClient.sh
count=0
while [ ! -s "$resp_file" ] && [ $count -lt $timeout ]; do
    sleep 1
    ((count++))
done

# Check if we got a response
if [ ! -s "$resp_file" ]; then
    echo "Error: No response received from server within $timeout seconds."
    kill $nc_pid 2>/dev/null
    rm "$msg_file" "$enc_file" "$resp_file"
    exit 1
fi

echo "Received encrypted response."

# Extract actual encrypted data (removing any headers)
grep -v '^RESPONSE_PORT:' "$resp_file" > "${resp_file}.enc"

# Decrypt the response using passphrase + key
dec_resp_file=$(mktemp)
openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -in "${resp_file}.enc" -out "$dec_resp_file" 2>/dev/null

if [ $? -ne 0 ]; then
    echo "Error: Failed to decrypt the server response."
    rm "$msg_file" "$enc_file" "${resp_file}.enc" "$dec_resp_file"
    exit 1
fi

# Display the response
response=$(cat "$dec_resp_file")
echo "Server response: $response"

# Clean up temporary files
rm "$msg_file" "$enc_file" "$resp_file" "${resp_file}.enc" "$dec_resp_file"

^G Help      ^O Write Out   ^F Where Is   ^K Cut        ^T Execute   ^C Location   M-U Undo   M-A Set Mark   M-J To Bracket   M-B Previous   . Back
^X Exit      ^R Read File   ^R Replace   ^U Paste     ^J Justify   ^L Go To Line  M-E Redo   M-C Copy     M-B Where Was  M-F Next    ^ . Forward

```

This script is the client-side program that sends an encrypted message to the server and waits for an encrypted response.

## 1. `source aesConfig.sh`

- **Why:** This command loads the configuration file (`aesConfig.sh`) that contains variables like `SERVER_PORT`, `CLIENT_PORT`, `KEY_FILE`, etc.
- **What it does:** It ensures the client knows which ports to use and where to find the encryption key.

- **Advanced:** The `source` command is used to include the configuration file in the current script, making the variables available for use.

## 2. `if [ ! -f "$KEY_FILE" ]; then ...`

- **Why:** This checks if the key file exists. If not, the script stops because encryption cannot happen without a key.
- **What it does:** It ensures the key file is present before proceeding.
- **Advanced:** The `-f` flag checks if the file exists and is a regular file. This is a common practice in shell scripting to validate file existence.

## 3. `KEY=$(cat "$KEY_FILE")`

- **Why:** This reads the AES key from the key file.
- **What it does:** The key is stored in the `KEY` variable for later use in encryption.
- **Advanced:** The `cat` command reads the contents of the file and stores it in the `KEY` variable. This key is used in conjunction with the passphrase for encryption.

## 4. `read name`

- **Why:** This takes user input (e.g., a name like "Bob") to send to the server.
- **What it does:** The input is stored in the `name` variable.
- **Advanced:** The `read` command is used to capture user input interactively. This is useful for dynamic data entry.

## 5. `openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -in "$msg_file" -out "$enc_file"`

- **Why:** This encrypts the message using AES-256 encryption.
- **What it does:** It takes the user's input (stored in `msg_file`), encrypts it using the passphrase and key, and saves the encrypted data in `enc_file`.
- **Advanced:**
  - `-aes-256-cbc`: Specifies the encryption algorithm (AES-256 in CBC mode).
  - `-base64`: Encodes the output in Base64 format for easy transmission.
  - `-salt`: Adds extra security by using a random salt.
  - `-pbkdf2`: Uses the PBKDF2 key derivation function for stronger key generation.
  - `-pass`: Combines the passphrase and key for encryption.

## 6. `nc -l $CLIENT_PORT > "$resp_file" &`

- **Why:** This starts a netcat listener on the client's port to wait for the server's response.
- **What it does:** It listens for incoming data on `CLIENT_PORT` and saves it to `resp_file`.
- **Advanced:** The `&` at the end runs the command in the background, allowing the script to continue executing while waiting for the server's response.

## 7. `(echo "RESPONSE_PORT:$CLIENT_PORT"; cat "$enc_file") | nc localhost $SERVER_PORT`

- **Why:** This sends the encrypted data to the server.
- **What it does:** It sends the client's port number (for the server to respond) and the encrypted message to the server.

- **Advanced:** The `|` (pipe) operator is used to send the output of one command as input to another. Here, it sends the port number and encrypted data to the server.

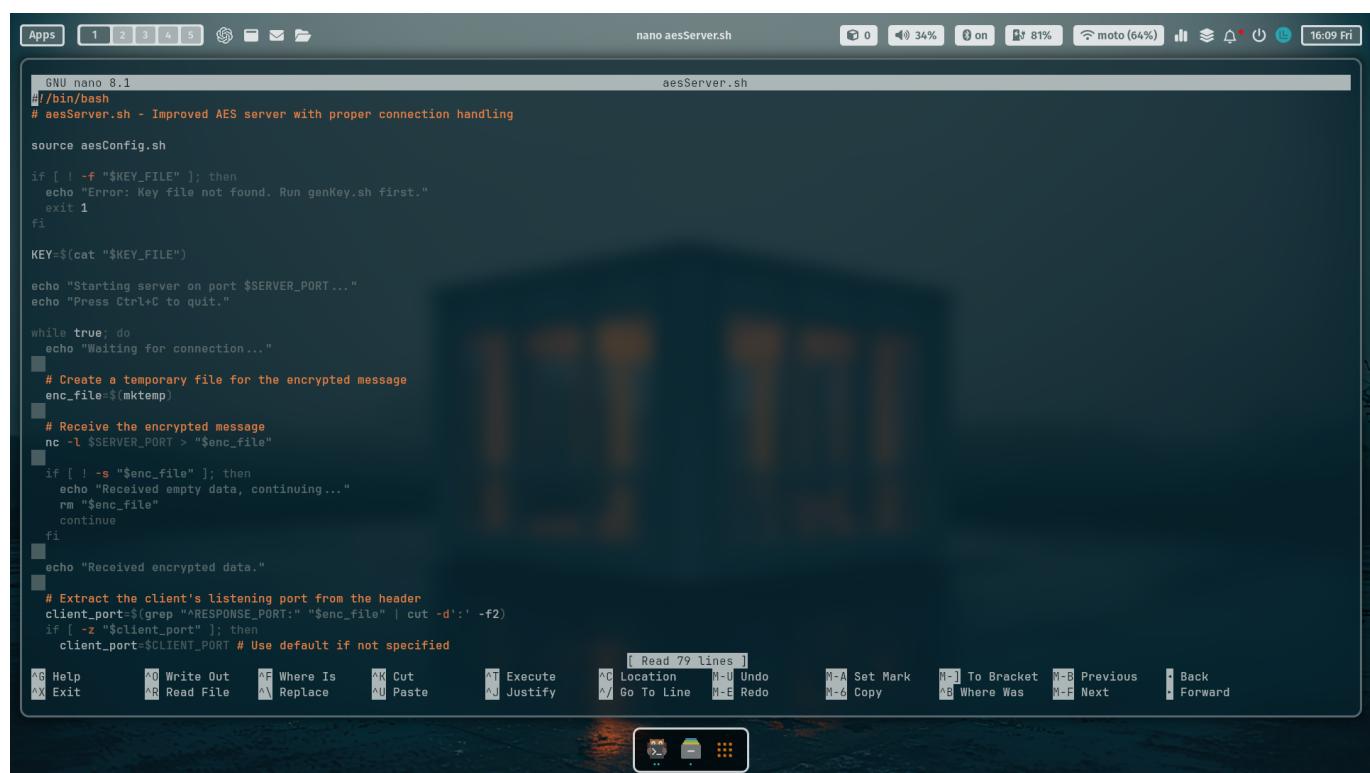
8. `openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -in "${resp_file}.enc" -out "$dec_resp_file"`

- **Why:** This decrypts the server's response.
- **What it does:** It takes the encrypted response from the server, decrypts it, and saves the result in `dec_resp_file`.
- **Advanced:** The `-d` flag is used for decryption. The rest of the options are the same as for encryption.

9. `rm "$msg_file" "$enc_file" "$resp_file" "${resp_file}.enc" "$dec_resp_file"`

- **Why:** This cleans up temporary files created during the process.
- **What it does:** It deletes all temporary files to free up space.
- **Advanced:** The `rm` command is used to remove files. This is important for security, as it ensures that sensitive data (like encrypted messages) is not left on the system.

## b. aesServer.sh



```

GNU nano 8.1
#!/bin/bash
# aesServer.sh - Improved AES server with proper connection handling

source aesConfig.sh

if [ ! -f "$KEY_FILE" ]; then
    echo "Error: Key file not found. Run genKey.sh first."
    exit 1
fi

KEY=$(cat "$KEY_FILE")

echo "Starting server on port $SERVER_PORT..."
echo "Press Ctrl+C to quit."

while true; do
    echo "Waiting for connection..."
    # Create a temporary file for the encrypted message
    enc_file=$(mktemp)

    # Receive the encrypted message
    nc -l $SERVER_PORT > "$enc_file"

    if [ ! -s "$enc_file" ]; then
        echo "Received empty data, continuing..."
        rm "$enc_file"
        continue
    fi

    echo "Received encrypted data."
    # Extract the client's listening port from the header
    client_port=$(grep "RESPONSE_PORT:" "$enc_file" | cut -d':' -f2)
    if [ -z "$client_port" ]; then
        client_port=$CLIENT_PORT # Use default if not specified
    fi
done

```

```

GNU nano 8.1
aesServer.sh

if [ $? -ne 0 ] || [ ! -s "$dec_file" ]; then
    echo "Error: Failed to decrypt the message."
    rm "$enc_file" "${enc_file}.enc" "$dec_file"
    continue
fi

# Get the decrypted message
name=$(cat "$dec_file")
echo "Decrypted message: '$name'"

# Create response
response="Hello $name!"
echo "Response: '$response'"

# Encrypt the response using passphrase + key
resp_enc_file=$(mktemp)
echo -n "$response" | openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -out "$resp_enc_file" 2>/dev/null

if [ $? -ne 0 ]; then
    echo "Error: Failed to encrypt the response."
    rm "$enc_file" "${enc_file}.enc" "$dec_file"
    continue
fi

echo "Sending encrypted response to client port $client_port..."

# Send the response to the client's listening port
cat "$resp_enc_file" | nc localhost "$client_port"

# Clean up temporary files
rm "$enc_file" "${enc_file}.enc" "$dec_file" "$resp_enc_file"

echo "Response sent."
done

```

The screenshot shows a terminal window with the title 'aesServer.sh'. The window contains the 'aesServer.sh' script in the nano text editor. The script is a shell script that reads an encrypted file, decrypts it, creates a response, encrypts the response, and then sends it back to a client. It uses OpenSSL for encryption and netcat for listening. The terminal has a dark theme with various icons at the top and bottom.

This script is the server-side program that listens for encrypted messages, decrypts them, and sends back an encrypted response.

### 1. `source aesConfig.sh`

- **Why:** This loads the configuration file (`aesConfig.sh`) to get variables like `SERVER_PORT`, `KEY_FILE`, etc.
- **What it does:** It ensures the server knows which port to listen on and where to find the encryption key.
- **Advanced:** The `source` command is used to include the configuration file in the current script, making the variables available for use.

### 2. `KEY=$(cat "$KEY_FILE")`

- **Why:** This reads the AES key from the key file.
- **What it does:** The key is stored in the `KEY` variable for later use in decryption.
- **Advanced:** The `cat` command reads the contents of the file and stores it in the `KEY` variable. This key is used in conjunction with the passphrase for decryption.

### 3. `nc -l $SERVER_PORT > "$enc_file"`

- **Why:** This listens for incoming connections from the client.
- **What it does:** It receives encrypted data from the client and saves it to `enc_file`.
- **Advanced:** The `nc` (netcat) command is used to listen for incoming connections on the specified port.

### 4. `client_port=$(grep '^RESPONSE_PORT:' "$enc_file" | cut -d':' -f2)`

- **Why:** This extracts the client's port number from the received data.
- **What it does:** It tells the server where to send the response.

- **Advanced:** The `grep` command searches for the line containing `RESPONSE_PORT:`, and `cut` extracts the port number.

5. `openssl enc -aes-256-cbc -d -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -in "${enc_file}.enc" -out "$dec_file"`

- **Why:** This decrypts the client's message.
- **What it does:** It takes the encrypted data, decrypts it, and saves the result in `dec_file`.
- **Advanced:** The `-d` flag is used for decryption. The rest of the options are the same as for encryption.

6. `response="Hello $name!"`

- **Why:** This creates the server's response message.
- **What it does:** It prepends "Hello" to the decrypted name (e.g., "Hello Bob!").
- **Advanced:** This is a simple string concatenation, but it could be extended to include more complex logic.

7. `openssl enc -aes-256-cbc -base64 -salt -pbkdf2 -pass "pass:${PASSPHRASE}${KEY}" -out "$resp_enc_file"`

- **Why:** This encrypts the server's response.
- **What it does:** It encrypts the response message and saves it in `resp_enc_file`.
- **Advanced:** The same encryption options are used as in the client script.

8. `cat "$resp_enc_file" | nc localhost "$client_port"`

- **Why:** This sends the encrypted response back to the client.
- **What it does:** It sends the encrypted data to the client's listening port.
- **Advanced:** The `|` (pipe) operator is used to send the output of one command as input to another. Here, it sends the encrypted data to the client.

---

### c. `captureTraffic.sh`

```

GNU nano 8.1                               captureTraffic.sh
#!/bin/bash
# captureTraffic.sh - Captures network traffic using tcpdump

source aesConfig.sh

case "$1" in
    start)
        echo "Starting packet capture on ports $SERVER_PORT and $CLIENT_PORT..."

        # Check if running as root (required for tcpdump)
        if [ "$(id -u)" -ne 0 ]; then
            echo "Error: This script must be run as root to capture packets."
            echo "Try: sudo $0 start"
            exit 1
        fi

        # Start tcpdump in background to capture traffic on specified ports
        tcpdump -i lo -w "$PCAP_FILE" "port $SERVER_PORT or port $CLIENT_PORT" &

        # Save PID for stopping later
        echo $! > tcpdump.pid

        echo "Capture started. Run 'sudo $0 stop' to end capture."
        echo "Traffic is being saved to $PCAP_FILE"
        ;;

    stop)
        if [ ! -f tcpdump.pid ]; then
            echo "Error: No capture process found. Start capture first."
            exit 1
        fi

        # Check if running as root
        if [ "$(id -u)" -ne 0 ]; then
            echo "Error: This script must be run as root to stop the capture."
            echo "Try: sudo $0 stop"
        fi
    ;;
esac

```

```

stop)
    if [ ! -f tcpdump.pid ]; then
        echo "Error: No capture process found. Start capture first."
        exit 1
    fi

    # Check if running as root
    if [ "$(id -u)" -ne 0 ]; then
        echo "Error: This script must be run as root to stop the capture."
        echo "Try: sudo $0 stop"
        exit 1
    fi

    # Kill the tcpdump process
    echo "Stopping packet capture..."
    kill $(cat tcpdump.pid)
    rm tcpdump.pid

    echo "Capture stopped. Captured packets saved to $PCAP_FILE"

    # Show a summary of the captured packets
    echo "Capture summary:"
    tcpdump -r "$PCAP_FILE" -n | head -10
    echo "..."
    echo "Use 'wireshark $PCAP_FILE' to view the full capture."
    ;;

*)
    echo "Usage: $0 [start|stop]"
    echo ""
    echo "start - Begin capturing traffic on ports $SERVER_PORT and $CLIENT_PORT"
    echo "stop - Stop capturing and save the results"
    ;;

esac

```

This script captures network traffic between the client and server using `tcpdump`.

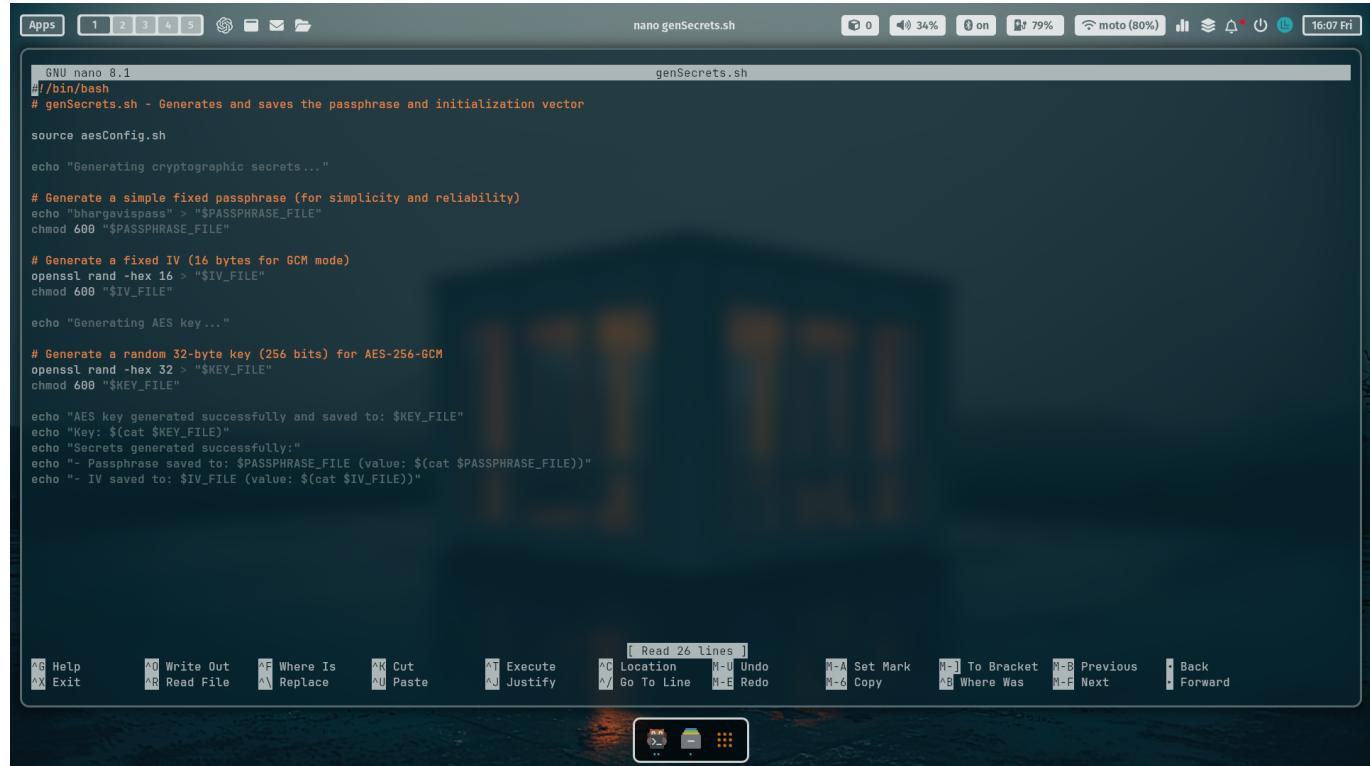
## 1. `tcpdump -i lo -w "$PCAP_FILE" "port $SERVER_PORT or port $CLIENT_PORT" &`

- **Why:** This starts capturing network traffic on the loopback interface (`lo`) for the specified ports.
- **What it does:** It saves the captured traffic to `PCAP_FILE` for later analysis.
- **Advanced:** The `-i lo` flag specifies the loopback interface, and `-w` writes the output to a file. The `&` runs the command in the background.

## 2. `kill $(cat tcpdump.pid)`

- **Why:** This stops the `tcpdump` process.
  - **What it does:** It ends the packet capture and saves the results.
  - **Advanced:** The `kill` command terminates the process with the PID stored in `tcpdump.pid`.
- 

#### d. genSecrets.sh



```
GNU nano 8.1                               genSecrets.sh
#!/bin/bash
# genSecrets.sh - Generates and saves the passphrase and initialization vector

source aesConfig.sh

echo "Generating cryptographic secrets..."

# Generate a simple fixed passphrase (for simplicity and reliability)
echo "bhargavispass" > "$PASSPHRASE_FILE"
chmod 600 "$PASSPHRASE_FILE"

# Generate a fixed IV (16 bytes for GCM mode)
openssl rand -hex 16 > "$IV_FILE"
chmod 600 "$IV_FILE"

echo "Generating AES key..."

# Generate a random 32-byte key (256 bits) for AES-256-GCM
openssl rand -hex 32 > "$KEY_FILE"
chmod 600 "$KEY_FILE"

echo "AES key generated successfully and saved to: $KEY_FILE"
echo "Key: $(cat $KEY_FILE)"
echo "Secrets generated successfully:"
echo "- Passphrase saved to: $PASSPHRASE_FILE (value: $(cat $PASSPHRASE_FILE))"
echo "- IV saved to: $IV_FILE (value: $(cat $IV_FILE))"

^G Help      ^O Write Out    ^F Where Is     ^K Cut          ^T Execute      [ Read 26 lines ]
^X Exit      ^R Read File    ^A Replace      ^P Paste        ^C Location     M-U Undo
^M Set Mark   ^/ Go To Line   ^J Justify     ^U Paste        M-D Copy       M-J To Bracket
^B Where Was  M-E Redo      M-B Previous   M-C Copy       M-F Next       M-Back
M-F Forward
```

This script generates the passphrase, initialization vector (IV), and AES key.

##### 1. `echo "bhargavispass" > "$PASSPHRASE_FILE"`

- **Why:** This creates a fixed passphrase for encryption.
- **What it does:** It saves the passphrase to `PASSPHRASE_FILE`.
- **Advanced:** A fixed passphrase is used for simplicity, but in a real-world scenario, a more secure method (e.g., random generation) should be used.

##### 2. `openssl rand -hex 16 > "$IV_FILE"`

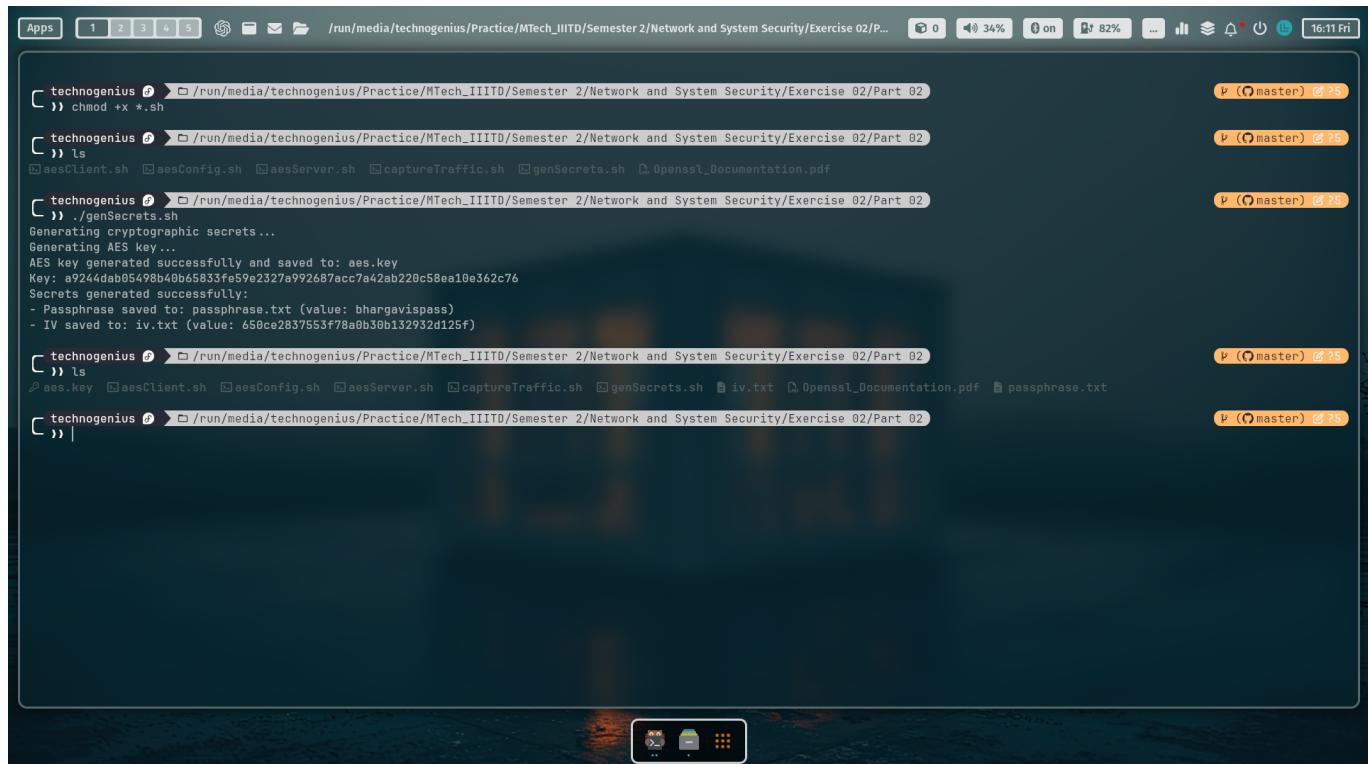
- **Why:** This generates a random 16-byte IV for AES encryption.
- **What it does:** It saves the IV to `IV_FILE`.
- **Advanced:** The `openssl rand` command generates cryptographically secure random data.

##### 3. `openssl rand -hex 32 > "$KEY_FILE"`

- **Why:** This generates a random 32-byte key for AES-256 encryption.
  - **What it does:** It saves the key to `KEY_FILE`.
  - **Advanced:** The key is 32 bytes (256 bits), which is the required size for AES-256 encryption.
- 

## 2. Screenshots and Output Images

## Screenshot 1: Running `genSecrets.sh`



```
technogenius@technogenius-OptiPlex-5090: ~ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
chmod +x *.sh
ls
aesClient.sh aesConfig.sh aesServer.sh captureTraffic.sh genSecrets.sh OpenSSL_Documentation.pdf
./genSecrets.sh
Generating cryptographic secrets...
Generating AES key ...
AES key generated successfully and saved to: aes.key
Key: a9244da05498b40b65833fe9e2327a992687acc7a42ab220c58ea10e362c76
Secrets generated successfully:
- Passphrase saved to: passphrase.txt (value: bhargavispass)
- IV saved to: iv.txt (value: 650ce2837553f78a0b30b132932d125f)

ls
aes.key aesClient.sh aesConfig.sh aesServer.sh captureTraffic.sh genSecrets.sh iv.txt OpenSSL_Documentation.pdf passphrase.txt
|
```

- **Description:** This screenshot shows the output of running `genSecrets.sh`. It generates the passphrase, IV, and AES key, and saves them to their respective files.

## Screenshot 2: Running `aesServer.sh`

```
technogenius ~ % ./aesServer.sh
Starting server on port 12345 ...
Press Ctrl+C to quit.
Waiting for connection ...
Received encrypted data.
Decrypted message: 'bob'
Response: 'Hello bob!'
Sending encrypted response to client port 12346 ...
Response sent.
Waiting for connection ...
Received encrypted data.
Decrypted message: 'alice'
Response: 'Hello alice!'
Sending encrypted response to client port 12346 ...
Response sent.
Waiting for connection ...
```

- **Description:** This screenshot shows the server starting and waiting for incoming connections on port 12345.

### Screenshot 3: Running aesClient.sh

```
[technogenius ~] ➜ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02 ✘ (master) 📁 ?5
  » ./aesClient.sh
Enter your name: bob
Encrypting: 'bob'
Sending encrypted data to server...
Received encrypted response.
Server response: Hello bob!

[technogenius ~] ➜ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02 ✘ (master) 📁 ?5
  » ./aesClient.sh
Enter your name: alice
Encrypting: 'alice'
Sending encrypted data to server...
Received encrypted response.
Server response: Hello alice!

[technogenius ~] ➜ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02 ✘ (master) 📁 ?5
  » |
```

- **Description:** This screenshot shows the client sending an encrypted message to the server and receiving an encrypted response.

```

technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─› ./aesClient.sh
Enter your name: bob
Encrypting: 'bob'
Sending encrypted data to server...
Received encrypted response.
Server response: Hello bob!

technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─› ./aesClient.sh
Enter your name: alice
Encrypting: 'alice'
Sending encrypted data to server...
Received encrypted response.
Server response: Hello alice!

```

```

technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─› ./aesServer.sh
Starting server on port 12345...
Press Ctrl+C to quit.
Waiting for connection...
Received encrypted data.
Decrypted message: 'bob'
Response: 'Hello bob!'
Sending encrypted response to client port 12346...
Response sent.
Waiting for connection...
Received encrypted data.
Decrypted message: 'alice'
Response: 'Hello alice!'
Sending encrypted response to client port 12346...
Response sent.
Waiting for connection...

```

**Screenshot 4: Capturing Traffic with captureTraffic.sh**

```

technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─› sudo ./captureTraffic.sh
[sudo] password for technogenius:
Usage: ./captureTraffic.sh [start|stop]

start - Begin capturing traffic on ports 12345 and 12346
stop - Stop capturing and save the results

technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─› sudo ./captureTraffic.sh start
Starting packet capture on ports 12345 and 12346...
Capture started. Run 'sudo ./captureTraffic.sh stop' to end capture.
Traffic is being saved to trafficCapture.pcap

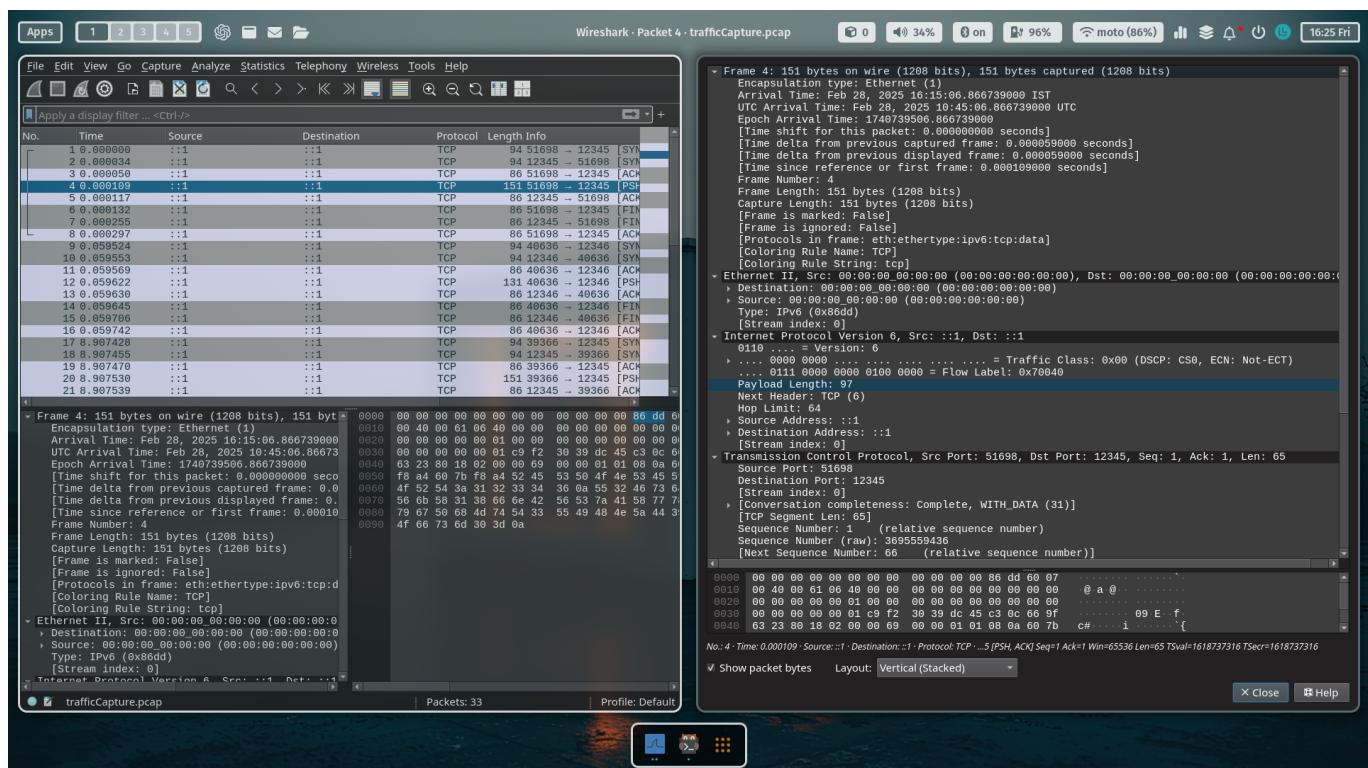
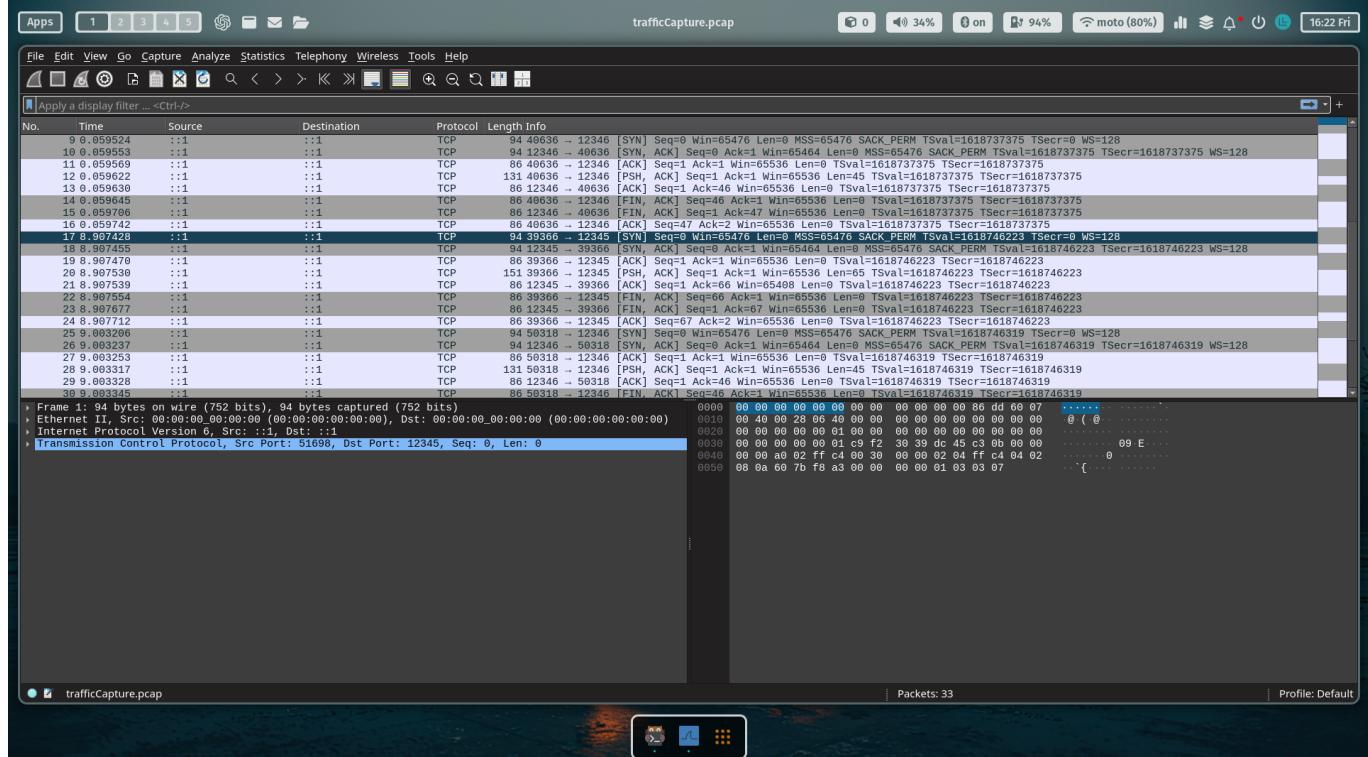
technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─› sudo ./captureTraffic.sh stop
Stopping packet capture...
Capture stopped. Captured packets saved to trafficCapture.pcap
Capture summary:
tcpdump: truncated dump file; tried to read 4 file header bytes, only got 0
...
Use 'wireshark trafficCapture.pcap' to view the full capture.

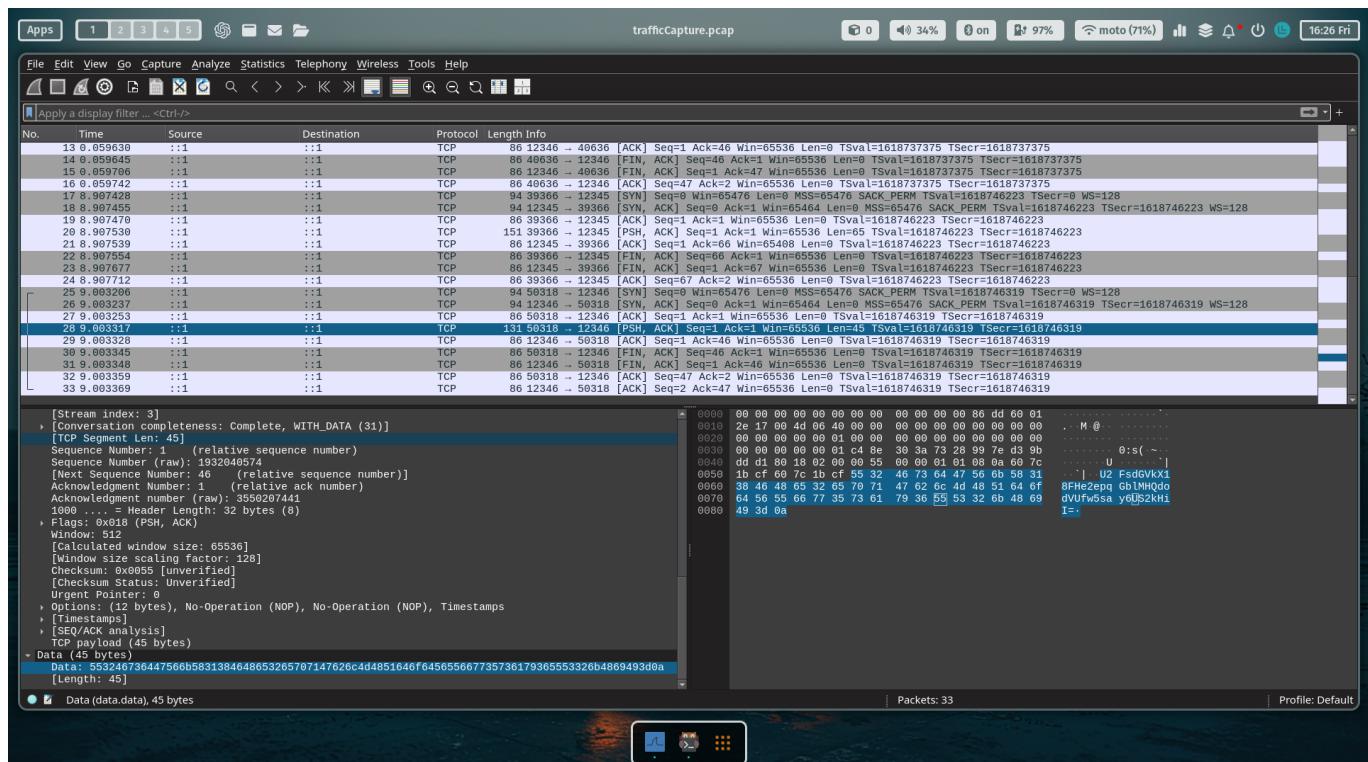
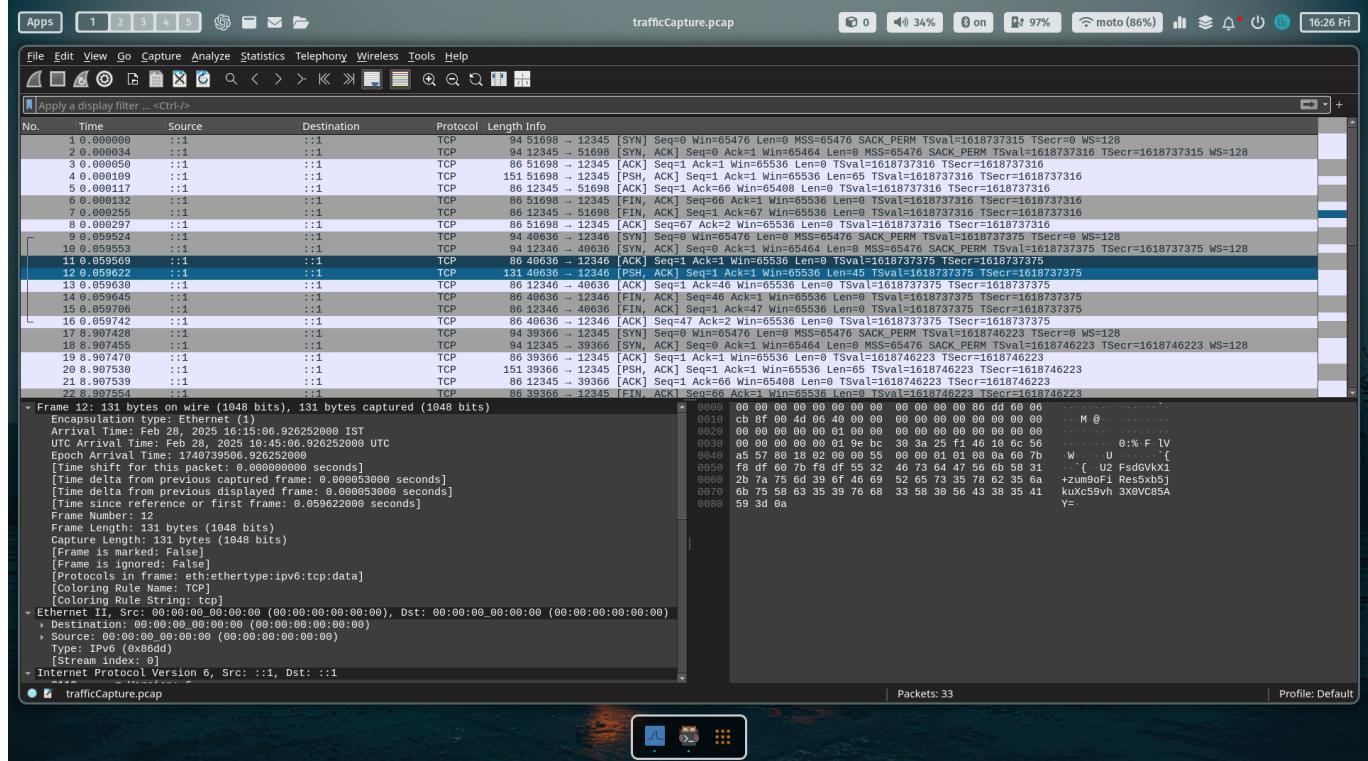
technogenius @ /run/media/technogenius/Practice/MTech_IIITD/Semester 2/Network and System Security/Exercise 02/Part 02
└─›

```

- **Description:** This screenshot shows the `tcpdump` process capturing traffic on ports `12345` and `12346`.

**Screenshot 5: Wireshark Encrypted Traffic**





- Description:** This screenshot shows the encrypted traffic in Wireshark. The data appears as random bytes, confirming that the communication is encrypted.

### 3. Encrypted Traffic in Wireshark

- Screenshot:** Wireshark showing encrypted traffic between the client and server on ports 12345 and 12346.
- Explanation:** The traffic is encrypted, so the data appears as random bytes. This confirms that the communication is secure.

## Conclusion

This exercise demonstrates how to use OpenSSL for AES encryption, netcat for communication, and `tcpdump` for traffic capture. The client and server exchange encrypted messages, and the traffic is confirmed to be encrypted using Wireshark. This setup is a basic example of secure communication using symmetric encryption, but it can be extended with more advanced features like key rotation, secure key exchange, and more.