Needham-Schroeder Based PDF Print Server

1. Introduction

This report provides an in-depth analysis of the secure, multi-threaded PDF print server implemented using the Needham-Schroeder protocol. The system consists of three primary components:

- **Key Distribution Center (KDC):** Handles authentication and session key distribution.
- **Print Server (PrnSrv):** Processes PDF conversion requests from clients.
- **Client:** Requests PDF conversion and retrieves the encrypted output.

The implementation is done in **C**, utilizing OpenSSL for cryptographic operations and POSIX threads for multi-threading.

2. System Architecture

2.1 Communication Flow

1. **Client authenticates with KDC:**
   - Sends username and derives a secret key.
   - KDC generates a session key and encrypts it.
2. **Client decrypts session key:**
   - Uses it to securely communicate with Print Server.
3. **Client requests file conversion:**
   - Sends a file to Print Server.
   - Print Server converts the file and encrypts the result.
4. **Client receives and decrypts PDF.**

3. File Descriptions and Code Breakdown

3.1 `config.h`

Purpose: Defines configuration constants for ports, key lengths, and security parameters.

```
#define KDC_PORT 5000
#define PRN_PORT 5001
#define MAX_CLIENTS 10
#define THREAD_POOL_SIZE 5
#define LOG_FILE "server.log"
#define KDC_SALT "fixed_salt_bhargav"
#define KEY_LEN 32
#define NONCE_LEN 12
#define TAG_LEN 16
#define SALT_LEN 16
```

3.2 `cryptoUtils.h` & `cryptoUtils.c`

Purpose: Handles cryptographic functions including key derivation, encryption, and decryption.

Functions:

- `deriveKey()`: Derives an AES-256 key using PBKDF2.
- `encryptData()`: Encrypts data using AES-GCM.
- `decryptData()`: Decrypts AES-GCM encrypted data.
- `generateNonce()`: Generates a random nonce.

Key Implementation (AES-256-GCM Encryption):

```
void encryptData(unsigned char *plaintext, int plaintextLen, unsigned char *key,
                 unsigned char *nonce, unsigned char *ciphertext, unsigned char
*tag) {
    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL, NULL);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN, NONCE_LEN, NULL);
    EVP_EncryptInit_ex(ctx, NULL, NULL, key, nonce);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintextLen);
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, TAG_LEN, tag);
    EVP_CIPHER_CTX_free(ctx);
}
```

3.3 `logger.h` & `logger.c`

Purpose: Logs security events to `server.log`.

```
void logMessage(const char *message) {
    FILE *logFile = fopen(LOG_FILE, "a");
    fprintf(logFile, "[%s] %s\n", ctime(&now), message);
    fclose(logFile);
}
```

3.4 `client.c`

Purpose: Implements client-side communication with KDC and Print Server.

Execution Flow:

1. Connects to **KDC** for authentication.
2. Sends username and receives an encrypted session key.
3. Decrypts session key and connects to **Print Server**.
4. Sends a file for conversion.
5. Receives and decrypts the converted PDF.

3.5 `kdcServer.c`

Purpose: Implements the KDC authentication server.

Execution Flow:

1. Receives authentication request.
2. Derives the user's key using PBKDF2.
3. Generates a session key and encrypts it with the user's key.
4. Sends the encrypted session key and nonce to the client.

3.6 `prnSrv.c`

Purpose: Implements the Print Server.

Execution Flow:

1. Receives file from client.
2. Converts file to PDF using `enscript` or `img2pdf`.
3. Encrypts the PDF using the session key.
4. Sends the encrypted PDF back to the client.

4. Execution Steps

4.1 Compilation

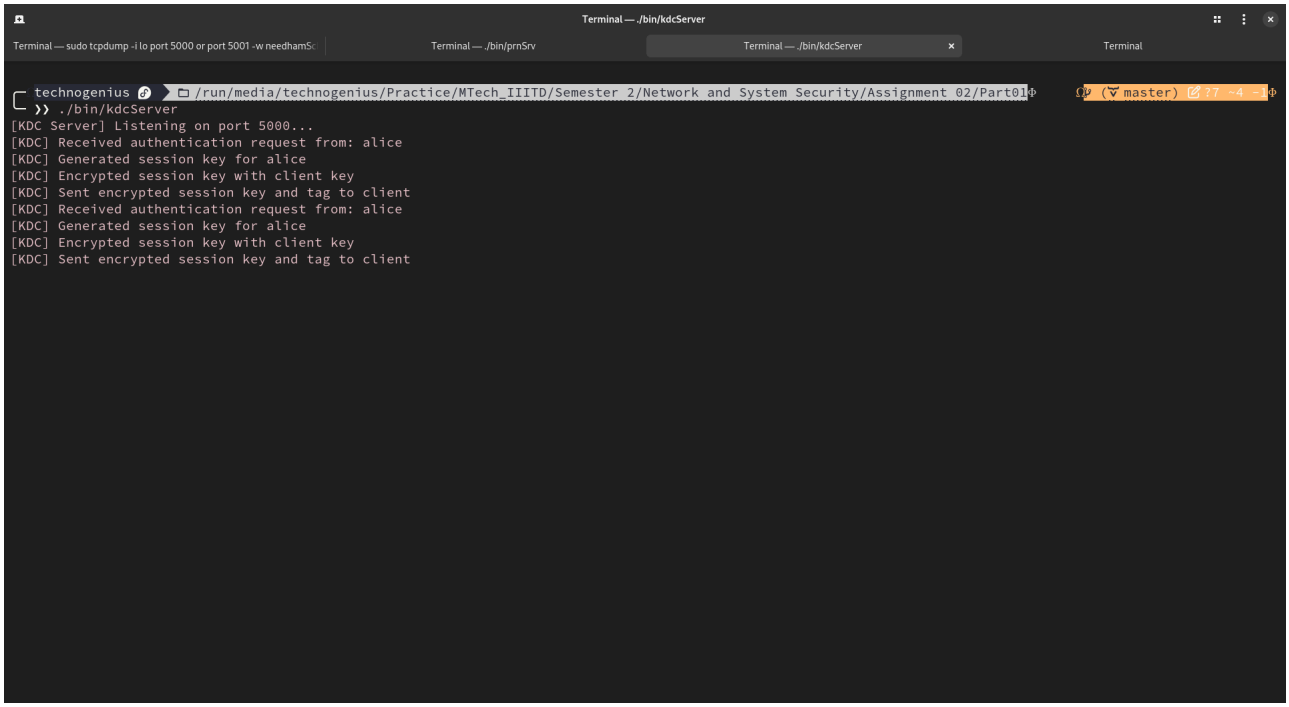
```
gcc -o client client.c cryptoUtils.c logger.c -lssl -lcrypto
gcc -o kdcServer kdcServer.c cryptoUtils.c logger.c -lssl -lcrypto -lpthread
gcc -o prnSrv prnSrv.c cryptoUtils.c logger.c -lssl -lcrypto -lpthread
```

4.2 Running the Servers

Start KDC Server

```
./kdcServer
```

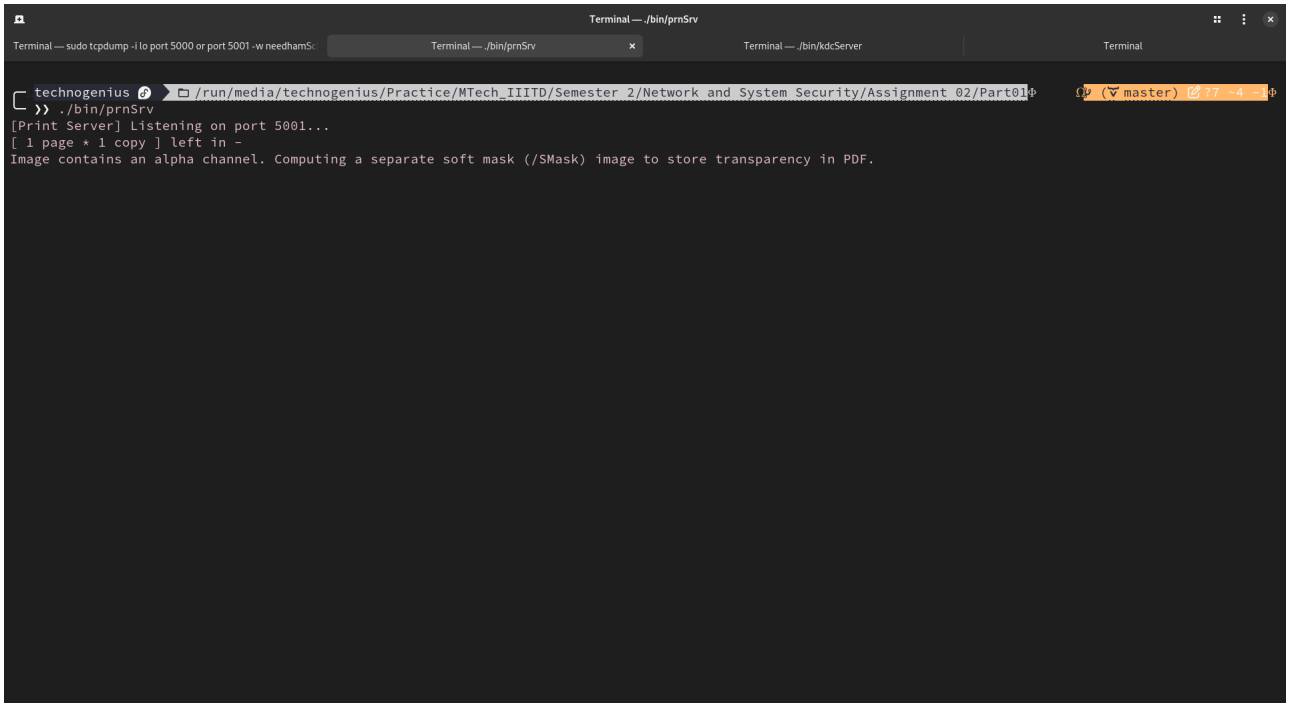
1. **KDC authentication process:** Verify successful authentication.



Start Print Server

```
./prnSrv
```

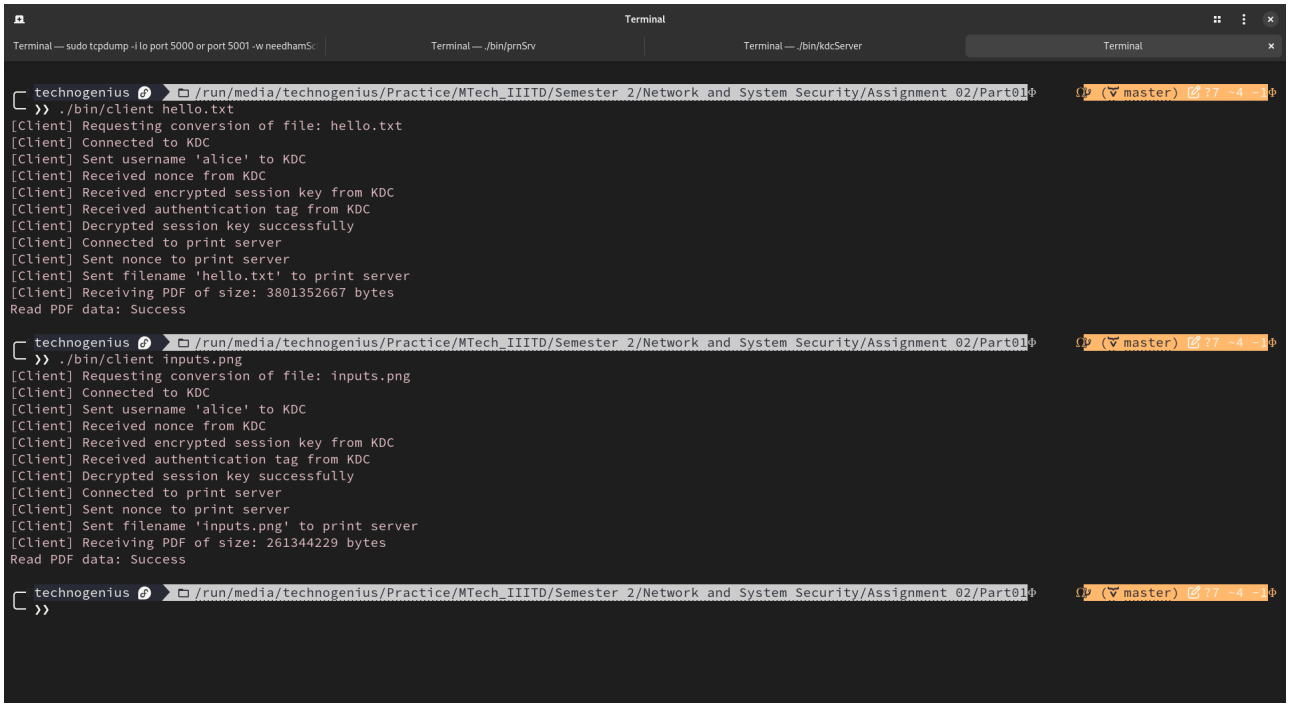
2. **Print Server processing:** Display PDF conversion.



4.3 Running the Client

```
./client sample.txt
```

3. **Client request handling:** Show file being sent and encrypted.



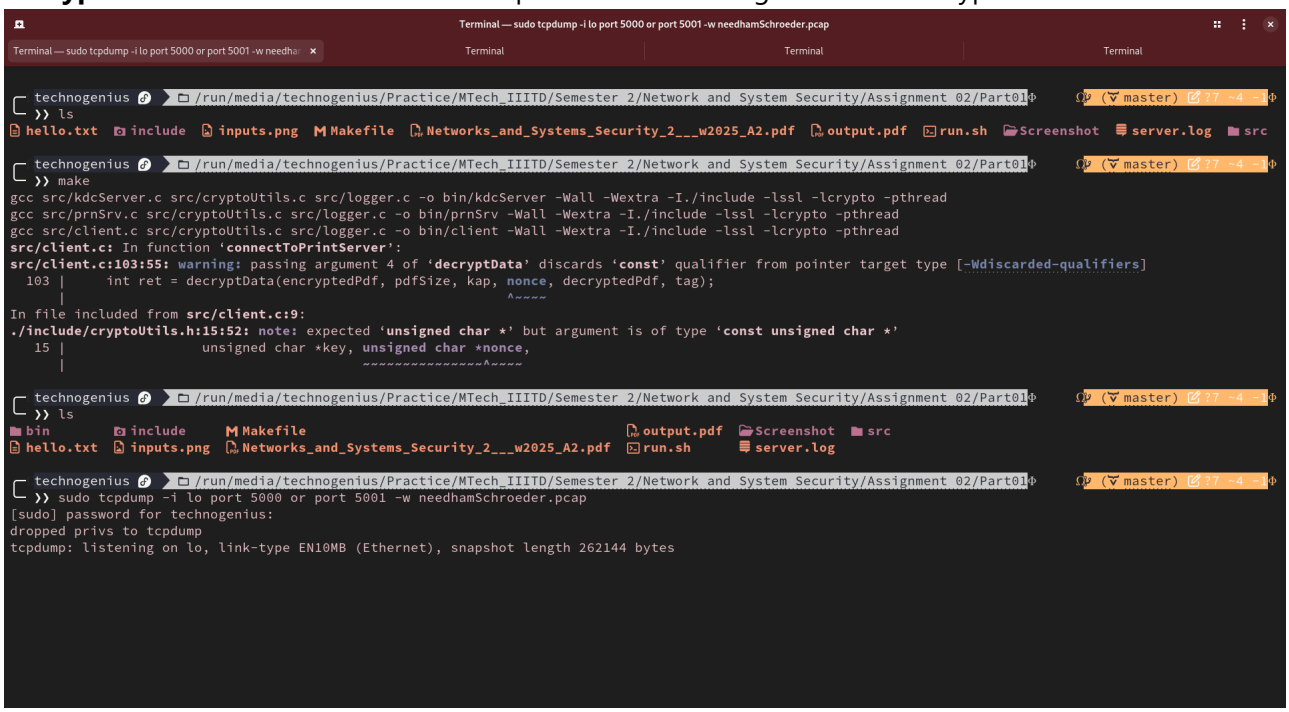
5. Security Verification

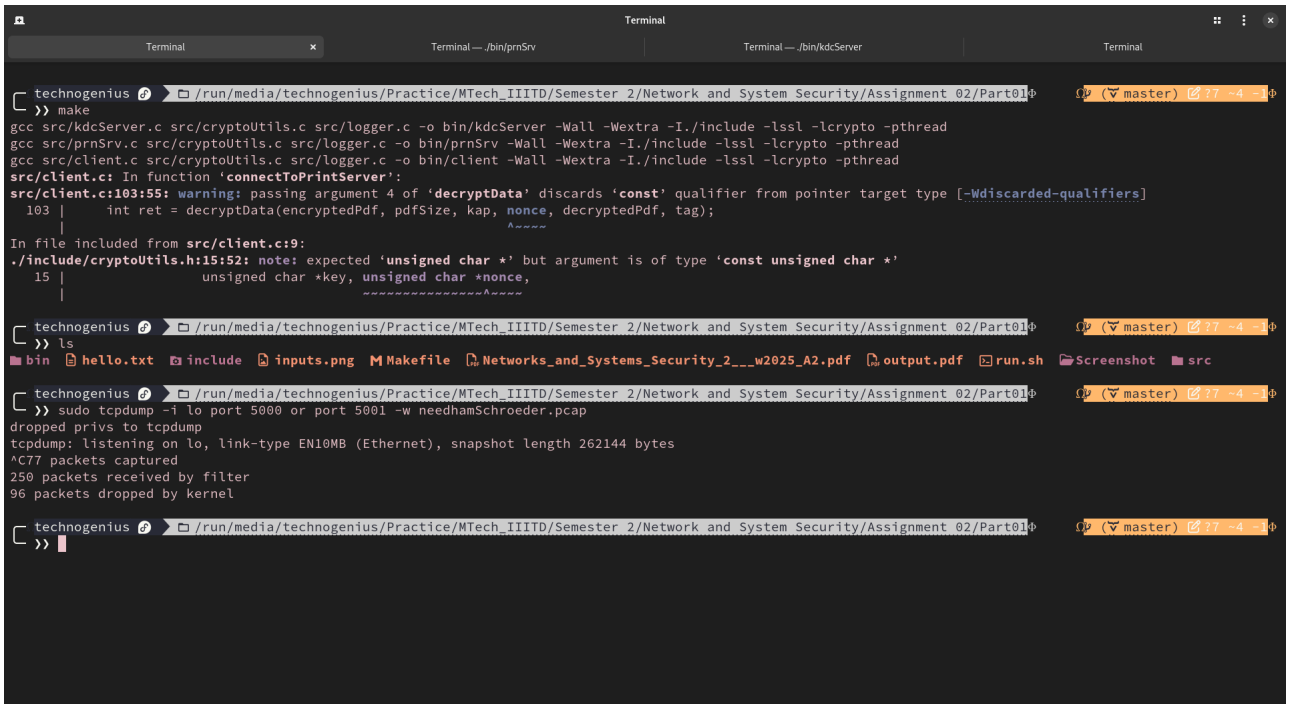
5.1 Capturing Encrypted Traffic

Run the following command to capture and analyze network traffic:

```
sudo tcpdump -i lo -w needhamSchroeder.pcap port 5000 or port 5001
```

4. **Encrypted network traffic:** Wireshark capture demonstrating AES-GCM encryption.



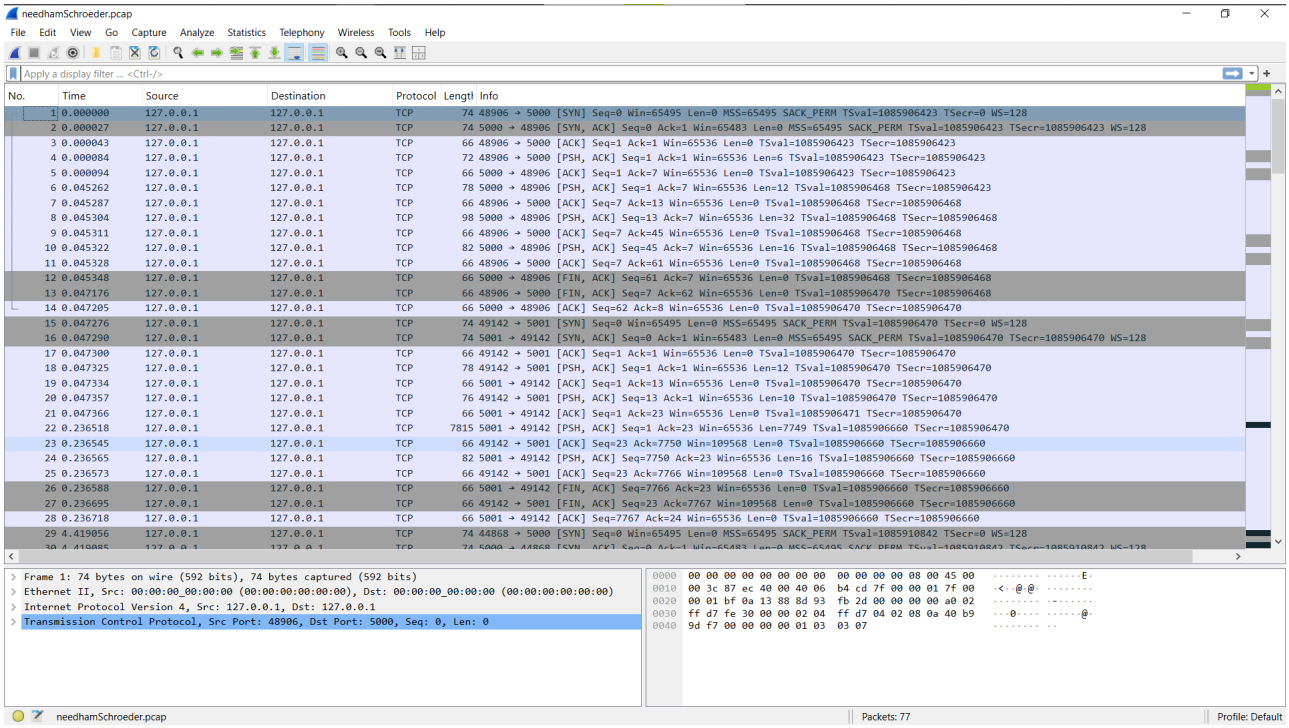


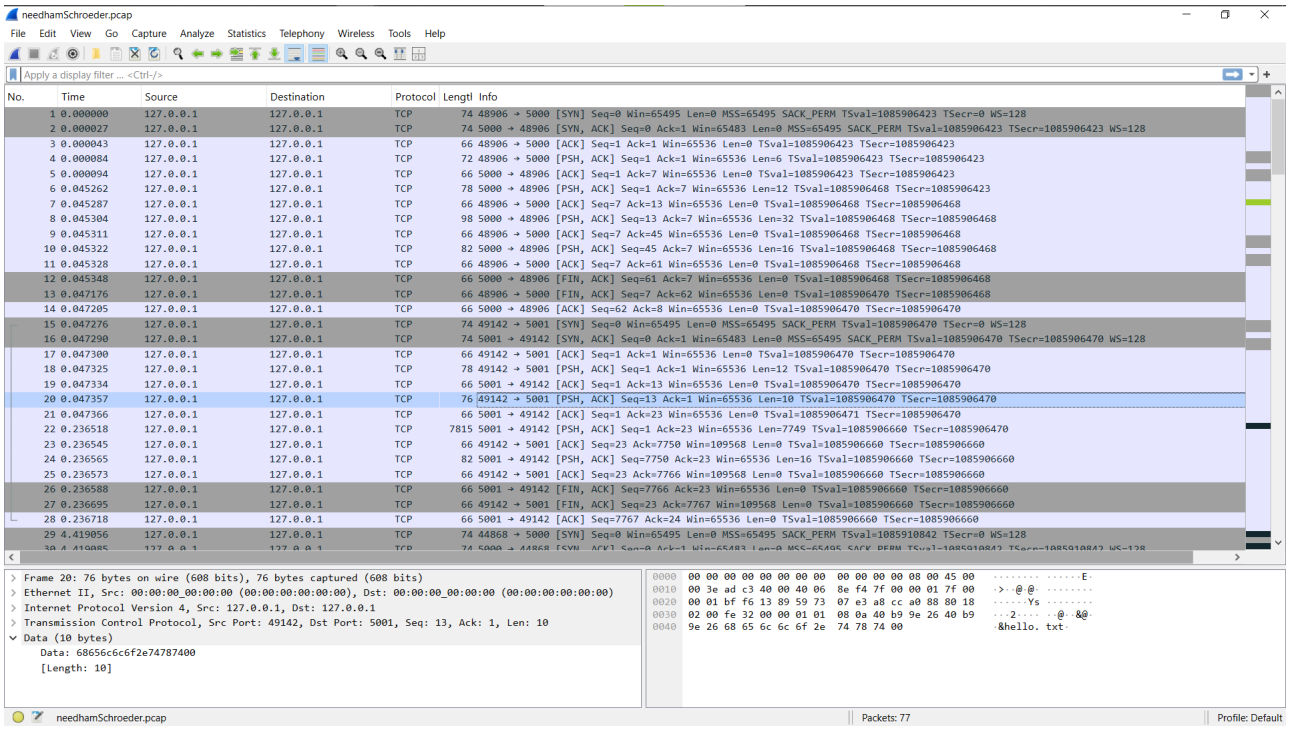
5.2 Analyzing the Capture

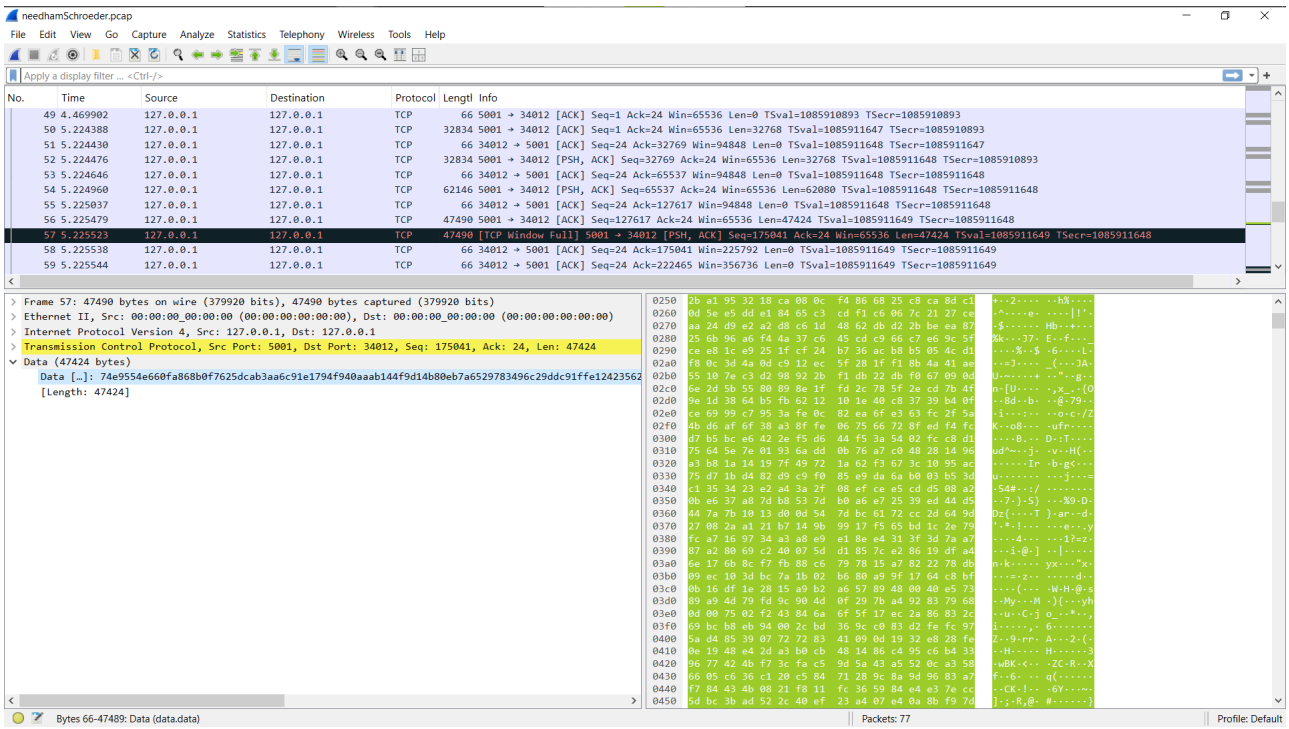
Open the capture file in Wireshark:

```
tshark -r traffic.pcap -x
```

5. **Wireshark**







Expected Outcome: Data should be encrypted, showing no plaintext communication.

6. Security Considerations

- **Replay Attack Prevention:** Nonces ensure each session is unique.
- **Authentication Security:** Uses PBKDF2 for strong key derivation.
- **Data Integrity:** AES-GCM ensures confidentiality and integrity.
- **Denial-of-Service (DoS) Protection:** Multi-threading ensures performance...