Secure File Transfer – Client Side (Milestone Five)

CLIENT.CPP Documentation

1 Purpose of the Client Program

The client program is responsible for securely transmitting a file to the server. It reads a file from disk, encrypts the file using AES-256-CBC, computes an integrity hash, and sends the encrypted data and metadata to the server over a TCP connection.

This program ensures:

- Confidentiality (through AES encryption)
- Integrity (through SHA-256 hashing)
- Secure transmission of file contents

2 Included Libraries and Their Purpose

- `#include <iostream>`
  Used for console output and logging client status.
- `#include <openssl/evp.h>`
  `#include <openssl/rand.h>`
  `#include <openssl/aes.h>`
  Used for cryptographic operations:
  - AES-256-CBC encryption
  - SHA-256 hashing
  - OpenSSL EVP interface
- `#include <vector>`
  `#include <string>`
  Used for dynamic memory storage and file data handling.

- `#include <sys/socket.h>`
  `#include <netinet/in.h>`
  `#include <arpa/inet.h>`
  `#include <unistd.h>`
  Used for low-level TCP socket programming.

3 Cryptographic Parameters

```
unsigned char key[32];
unsigned char iv[16];
```

- `key[32]` defines a 256-bit AES encryption key.
- `iv[16]` defines a 128-bit initialization vector required for AES-CBC mode.

Note: These values are hard-coded for demonstration and educational purposes.

4 Encryption Function Declaration

```
vector<unsigned char> encryptAES(...);
```

This function encrypts plaintext file data using AES-256-CBC through the OpenSSL EVP interface.

5 Socket Creation and Connection

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
```

The client creates a TCP socket using IPv4 and connects to the server using:

- `AF_INET`: IPv4 addressing
- `htons(6767)`: Server listening port
- `127.0.0.1`: Localhost address

6 Reading the File

```
FILE* fp = fopen(filename, "rb");
```

The client:

- Opens the file in binary mode
- Determines file size using `fseek` and `ftell`
- Reads the entire file into memory

7 Sending the Filename

```
int flen = strlen(filename);
send(sock, &flen, sizeof(flen), 0);
send(sock, filename, flen, 0);
```

The filename is transmitted first so the server knows how to name the decrypted output file.

8 Encrypting the File

```
vector<unsigned char> cText = encryptAES(pText, key, iv);
```

The plaintext file data is encrypted using AES-256-CBC before transmission.
The encryption process uses:

- `EVP_EncryptInit_ex`
- `EVP_EncryptUpdate`
- `EVP_EncryptFinal_ex`

Padding is handled automatically by the OpenSSL EVP interface.

9 Sending Encrypted File Size

```
long encryptSize = cText.size();
send(sock, &encryptSize, sizeof(encryptSize), 0);
```

The encrypted file size is sent so the server knows how many bytes to receive.

10 Sending Encrypted Data

```
send(sock, cText.data(), encryptSize, 0);
```

The encrypted file bytes are transmitted over the TCP connection.

11 Integrity Hash Generation and Transmission

```
unsigned char digest[32];
EVP_Digest(cText.data(), cText.size(), digest, NULL, EVP_sha256(), NULL);
send(sock, digest, 32, 0);
```

The client computes a SHA-256 hash of the encrypted file and sends it to the server. This allows the server to verify that the encrypted data was not altered during transmission.

12 Connection Cleanup

```
close(sock);
```

Closes the socket after the file transfer is complete.

13 Security Properties Achieved

Confidentiality

The file is encrypted using AES-256 before transmission. Plaintext is never sent over the network.

Integrity

SHA-256 hashing ensures the encrypted file is not modified or corrupted in transit.

Correctness

The server can successfully decrypt the received ciphertext and reconstruct the original file.

14 Limitations

- Encryption keys and IVs are hard-coded.
- No authentication or secure key exchange is implemented.
- Intended for academic and instructional use only.

15 Conclusion

The client implementation fulfills all Milestone Five requirements by securely encrypting and transmitting file data. When combined with the server program, it forms a complete secure file transfer system that demonstrates confidentiality, integrity, and correct protocol design.