

SPL-1 Project Report

Secure Chat with Encryption and Compression-Decompression

SE 305: Software Project Lab - 1

Submitted by

Tasnim Mahfuz Nafis

BSSE Roll No. : 1327

BSSE Session: 2020-21

Supervised by

Dr. Zerina Begum

Professor

Institute of Information Technology

University of Dhaka



Institute of Information Technology

University of Dhaka

[21-05-2023]

Table of Contents

1. Introduction	3
1.1. Background Study	3
1.2. Challenges	4
2. Project Overview	4
3. User Manual	9
4. Conclusion	9
References	10

1.Introduction

The Secure Chat with Encryption and Compression-Decompression project is a secure communication tool developed using socket programming in C++. This project incorporates advanced techniques such as Huffman compression and AES encryption to ensure efficient data transmission and data security. The server-client architecture is established through socket programming, enabling clients to connect to the chat server over a network. Socket programming provides a flexible and reliable means of communication between the server and clients, allowing real-time exchange of messages. By implementing the Huffman compression algorithm, the project aims to optimize the data transfer process by reducing the size of the transmitted messages without losing any crucial information. To further enhance the security of the communication, the project employs the Advanced Encryption Standard (AES) algorithm. AES is a widely-used symmetric encryption algorithm known for its strength and efficiency. By encrypting the messages before transmission and decrypting them upon reception, the project ensures that only the intended recipients can access and understand the exchanged information. The combination of these techniques not only optimizes data transmission but also guarantees the confidentiality of the communication, protecting sensitive information from unauthorized access or tampering.

In summary, the Secure Chat Server project offers a secure, efficient, and reliable platform for real-time communication. By leveraging socket programming, Huffman compression, and AES encryption, this project aims to address the increasing need for secure communication solutions, providing users with a secure and seamless chat experience.

1.1 Background Study

To successfully implement the project, the following background information was studied:

1.1.1 Socket Programming in C++: Socket programming is a crucial aspect of network communication, allowing software applications to establish connections, exchange data, and communicate over a network. In C++, the socket programming paradigm is implemented through the use of sockets, which are software endpoints that enable data transfer between different devices or processes. Socket programming in C++ provides a flexible and efficient means of creating client-server architectures, facilitating real-time communication.

1.1.2 Huffman Algorithm: The Huffman algorithm is a widely used data compression technique that efficiently reduces the size of data by assigning variable-length codes to characters based on

their frequency of occurrence. This algorithm allows for the creation of optimal prefix codes, where the most frequently used characters are assigned shorter codes, resulting in significant compression gains. Huffman compression finds extensive application in various domains, such as file compression, data transmission, and storage optimization.

1.1.3 AES Algorithm: The Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely adopted for its robust security and performance. It was established as a replacement for the Data Encryption Standard (DES). AES operates on fixed-size blocks of data and supports key sizes of 128, 192, and 256 bits. The algorithm employs a combination of substitution, permutation, and bitwise operations to provide confidentiality, integrity, and authenticity of the transmitted data. AES encryption finds widespread use in secure communication, data storage, and cryptographic protocols.

1.2 Challenges

To implement the chat facility, I have faced several challenges. They are as follows:

- Working with header files in C++ for the first time.
- Reading and writing of files in C++.
- Establishing socket connection in C++.
- Working with multiple source files.
- Working with long lines of code and keeping track of everything.
- Using Huffman compression and then encrypting it.
- Make the AES algorithm work on text size of any length (initially it was only working with text 32 characters long or less and I had to fix it so that it can work on any size of text.)
- Working with decryption and then decompression in the client server.
- Figuring out how to make the whole project work and explore to find the resources.

1.3 Project Overview

When the server wants to send a message to a client, I have at first compressed the text using Huffman algorithm. This allows for efficient and fast data transmission through the LAN. After the compression process, I have encrypted the text file with selected AES encryption code and sent the file through socket to the desired client. When the text arrived at the client side, it was first decrypted and then the compressed file was decompressed.

1.3.1 Socket Programming

Socket programming in C++ allows for the implementation of network communication between different devices or processes. Here is a technical overview socket programming implementation in C++:

1. Including the necessary libraries:
 - Start by including the required libraries for socket programming in C++, such as `<sys/socket.h>`, `<netinet/in.h>`, and `<arpa/inet.h>`.
 - These libraries provide the necessary functions and data structures for socket creation, binding, listening, accepting connections, sending, and receiving data.
2. Creating a socket:
 - Use the `socket()` function to create a socket.
 - Specify the address family (IPv4 or IPv6) and the socket type (TCP or UDP).
 - The function returns a file descriptor representing the socket.
3. Binding the socket to an address and port:
 - Use the `bind()` function to associate the socket with a specific address and port.
 - Specify the socket file descriptor, address family, IP address (using `sockaddr_in` or `sockaddr_in6` structure), and port number.
 - This step is necessary for the server to listen on a specific network interface and port.
4. Listening for incoming connections (for server):
 - Use the `listen()` function to make the socket listen for incoming connections.
 - Specify the socket file descriptor and the maximum number of pending connections that can be queued.
 - This step is crucial for the server to accept client connections.
5. Accepting client connections (for server):
 - Use the `accept()` function to accept incoming client connections.
 - Specify the socket file descriptor, along with an optional client address structure (`sockaddr_in` or `sockaddr_in6`) to store client details.
 - The function returns a new file descriptor representing the accepted connection, which can be used for further communication with the client.
6. Connecting to a server (for client):
 - Use the `connect()` function to establish a connection to the server.
 - Specify the socket file descriptor, server address structure (`sockaddr_in` or `sockaddr_in6`), including the server's IP address and port number.

- This step establishes a connection with the server, enabling bidirectional data transfer.

7. Sending and receiving data:

- Use the `send()` and `recv()` functions to send and receive data on the sockets, respectively.
- Specify the socket file descriptor, a buffer to hold the data, the data size, and optional flags for additional control.
- Data can be sent and received in chunks or using loops until the entire message is transmitted.

8. Closing the socket:

- Use the `close()` function to release the socket resources.
- Specify the socket file descriptor to be closed.
- Closing the socket terminates the connection and frees up system resources.

These steps provide a high-level technical overview of socket programming implementation in C++. By utilizing the appropriate functions and data structures from the socket libraries, developers can create robust network communication systems and build client-server architectures for various applications.

```

4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <unistd.h>
8  #include <errno.h>
9  #include <string.h>
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <netinet/in.h>
13 #include <netdb.h>
14 #include <arpa/inet.h>
15 #include <sys/wait.h>
16 #include <signal.h>
17
18 #define PORT "3490" // the port users will be connecting to
19
20 #define BACKLOG 10 // how many pending connections queue will hold
21
22 void sigchld_handler(int s)
23 {
24     // waitpid() might overwrite errno, so we save and restore it:
25     int saved_errno = errno;
26
27     while(waitpid(-1, NULL, WNOHANG) > 0);
28
29     errno = saved_errno;
30 }

```

This picture above is an example of the codebase for socket programming in c++.

1.3.2 Huffman Encryption:

The Huffman algorithm is commonly used for data compression and involves the following steps in its implementation:

1. Character Frequency Analysis:

- Starting by analyzing the frequency of characters in the data that needs to be compressed.
- Counting the occurrences of each character and storing them in a frequency table or data structure.

2. Building a Huffman Tree:

- Creating priority queue or min-heap based on the character frequencies.
- Each node of the priority queue or min-heap represents a tree with a character and its frequency.
- Combining the two nodes with the lowest frequencies into a new node, with the combined frequency being the sum of the child nodes.
- Repeating this process until only one node remains in the priority queue or min-heap, which becomes the root of the Huffman tree.

3. Generating Huffman Codes:

- Traverse the Huffman tree from the root to each leaf node.
- Assign a '0' bit for a left child and a '1' bit for a right child while traversing the tree.
- As you reach a leaf node, record the Huffman code (a sequence of '0' and '1' bits) associated with that character.
- Store the generated Huffman codes in a lookup table or data structure for encoding and decoding purposes.

4. Encoding the Data:

- With the Huffman codes generated, encode the original data by replacing each character with its corresponding Huffman code.
- Concatenate the Huffman codes of all characters to form the encoded data.
- The encoded data is a compressed representation of the original data, occupying fewer bits.

5. Decoding the Data:

- To decode the encoded data, start from the root of the Huffman tree.
- Traverse the Huffman tree based on the '0' and '1' bits in the encoded data.

- When a leaf node is reached, output the corresponding character and continue decoding from the root.

1.3.3 AES Algorithm

The implementation of AES is as follows:

1. Key Expansion:
 - Begin by expanding the original encryption key into a set of round keys.
 - The key expansion process involves applying various transformations to generate a set of round keys, which will be used in each round of encryption and decryption.
2. Initial Round:
 - Start with the original plaintext data or input block.
 - Perform an XOR operation between the input block and the first round key.
3. Rounds:
 - AES operates in multiple rounds, with the number of rounds determined by the key size.
 - Each round consists of four main steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey.
4. a. SubBytes:
 - In this step, each byte of the input block is substituted with a corresponding byte from the AES S-box.
 - The S-box is a precomputed lookup table that replaces each byte based on a non-linear transformation.
5. b. ShiftRows:
 - In this step, the bytes in each row of the input block are cyclically shifted.
 - The first row remains unchanged, the second row shifts one byte to the left, the third row shifts two bytes to the left, and the fourth row shifts three bytes to the left.
6. c. MixColumns:
 - In this step, each column of the input block is mixed using a matrix multiplication operation.
 - This step provides diffusion and enhances the cryptographic strength of AES.
7. d. AddRoundKey:
 - In this step, an XOR operation is performed between the processed block and the round key for the current round.
 - The round key is derived from the key expansion process.

8. Final Round:

- The final round of AES excludes the MixColumns step.
- It includes the SubBytes, ShiftRows, and AddRoundKey steps.

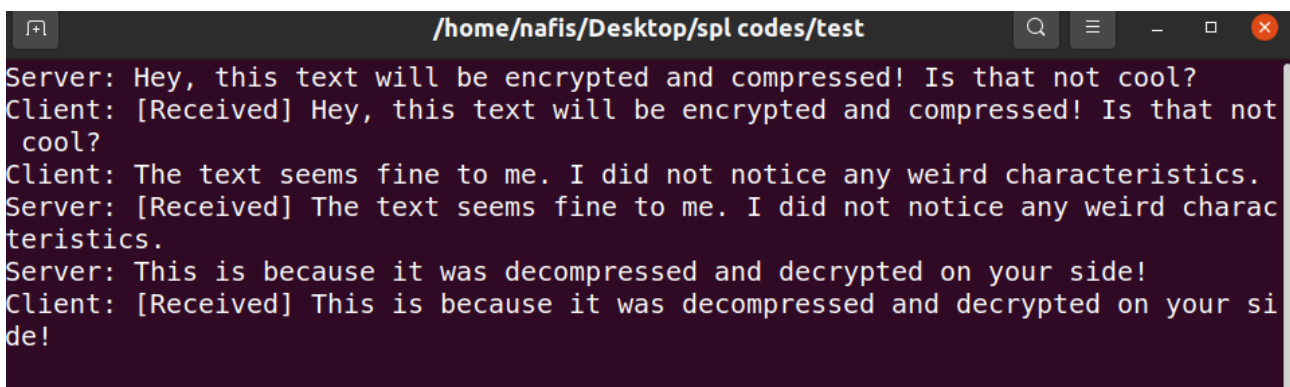
9. Output:

- After the final round, the resulting processed block is the ciphertext.
- The ciphertext is the encrypted version of the original plaintext data.

To decrypt the ciphertext, the same process is followed, but in reverse order. The decryption process involves using the round keys in reverse order and applying the inverse transformations for each step.

1.4 User Manual

After running the executable file, connection can be established between the server and client. Text will be sent from the server side and the compression and encryption will be done automatically, no menu needs to be selected for this. When the text reaches the client side, it will again automatically be decompressed and decrypted before the main text is shown in the client side. When texts are replied, client and server will switch sides.



```

/home/nafis/Desktop/spl codes/test
Server: Hey, this text will be encrypted and compressed! Is that not cool?
Client: [Received] Hey, this text will be encrypted and compressed! Is that not cool?
Client: The text seems fine to me. I did not notice any weird characteristics.
Server: [Received] The text seems fine to me. I did not notice any weird characteristics.
Server: This is because it was decompressed and decrypted on your side!
Client: [Received] This is because it was decompressed and decrypted on your side!

```

1.5 Conclusion

The project taught me to work with socket programming in C++. Before working on the project, I had no knowledge about network programming. I also learned to incorporate security and better transmission facility in my codes. I have tried to apply some OOP principles such as increasing modularity and decreasing the coupling. I learnt file handling and working with header files in C++. Besides all these, I have had the opportunity to be supervised by a professor in the technology field which gives me the confidence to take on bigger projects in future.

References

- [1] <https://beej.us/guide/bgnet>, Beej's Guide to Network Programming, last accessed on 14-05-2023
- [2] Information Security: Principles and Practice by Mark Stamp 2 nd Edition Wiley 2011