Source code and detailed setup scripts are available at https://git.trance-0.com/Trance-0/CSE4303H3.

1. Description of experimental setup

   We use the same setup as in the previous assignment.

   Here is the docker compose and environment file used for this assignment.

   We use `sha-1` as the hash function for HMAC, since it is simple and fast compared to other hash functions, and to test the code effectively, we remove the block cipher and stream cipher.

   We moved the helper functions like reading environment variables and calculating HMAC to a separate file `helper.cpp` and `helper.h` to make the code cleaner.

```
services:
  server:
    image: debian:bookworm-slim
    container_name: hw3-server
    working_dir: /app
    env_file:
      - .env
    networks:
      net-hw3:
        ipv4_address: ${SERVER_IP}
    volumes:
      - ./bin/server:/usr/local/bin/server:ro
    entrypoint: ["bash", "-lc", "apt-get update && apt-get install
       -y --no-install-recommends libstdc++6 iproute2 libssl-dev;
       exec stdbuf -oL -eL /usr/local/bin/server"]

  client:
    image: debian:bookworm-slim
    container_name: hw3-client
    working_dir: /app
    env_file:
      - .env
    networks:
      net-hw3:
        ipv4_address: ${CLIENT_IP}
    depends_on:
      - server
    stdin_open: true
    tty: true
    volumes:
      - ./bin/client:/usr/local/bin/client:ro
    entrypoint: ["bash", "-lc", "apt-get update && apt-get install
       -y --no-install-recommends libstdc++6 iproute2 libssl-dev;
       exec stdbuf -oL -eL /usr/local/bin/client"]
```

```
networks:
  net-hw3:
    name: net-hw3
    driver: bridge
    ipam:
      config:
        - subnet: ${NET_SUBNET}
```

Here is the environment file used for this assignment.

```
NET_SUBNET=172.30.0.0/24
SERVER_IP=172.30.0.10
CLIENT_IP=172.30.0.11
SERVER_PORT=3030
HMAC_KEY=
    FBEDF55EB2D01072E2AE0280FEA75F730473A32A57C0902A23CB55AC5DC0214EE6E34D735AA
```

Here is the `helper.cpp` we used for this assignment.

```
// load environment variables from .env file
#include <fstream>
#include <cstdlib>
#include <string>
#include <cstring>

#include <openssl/hmac.h>
#include <openssl/err.h>

void load_env(char const* path)
{
    std::ifstream f(path);
    std::string line;
    while (std::getline(f, line))
    {
        if (line.empty() || line[0] == '#')
            continue;
        auto pos = line.find('=');
        if (pos == std::string::npos)
            continue;

        std::string key = line.substr(0, pos);
        std::string val = line.substr(pos + 1);

#ifdef _WIN32
```

```
        _putenv_s(key.c_str(), val.c_str());
#else
        setenv(key.c_str(), val.c_str(), 1);
#endif
    }
}

// hex to byte helper function
int hex_to_bytes_upper(const char *hex, unsigned char *out, size_t
   out_size)
{
    size_t i = 0;
    while (hex[0] && hex[1])
    {
        if (i >= out_size)
            return -1;
        unsigned char h = hex[0];
        unsigned char l = hex[1];
        int hi = (h <= '9') ? (h - '0') : (h - 'A' + 10);
        int lo = (l <= '9') ? (l - '0') : (l - 'A' + 10);
        // minimal sanity check
        if (hi < 0 || hi > 15 || lo < 0 || lo > 15)
            return -1;
        out[i++] = (unsigned char)((hi << 4) | lo);
        hex += 2;
    }
    return (int)i;
}

char * byte_to_hex(unsigned char *bytes, size_t len)
{
    char *hex = new char[len * 2 + 1];
    for (size_t i = 0; i < len; i++)
    {
        sprintf(hex + i * 2, "%02X", bytes[i]);
    }
    hex[len * 2] = '\0';
    return hex;
}

void cal_hmac(unsigned char *mac, char *message)
{

    /* The secret key for hashing */
    const char *key_str = getenv("HMAC_KEY");
    if (key_str == NULL)
    {
        fprintf(stderr, "HMAC_KEY not set in environment\n");
        return;
    }
    unsigned char key_char[64];
    int key_len = hex_to_bytes_upper(key_str, key_char, sizeof(
       key_char));
    if (key_len != 64)    {
        fprintf(stderr, "Invalid HMAC_KEY format\n");
        return;
    }
    const char *key = (const char *)key_char;
    // printf("HMAC key: %s\n", byte_to_hex((unsigned char *)key,
       key_len));

    /* Change the length accordingly with your chosen hash engine.
    * Be careful of the length of string with the chosen hash
       engine. For
    example, SHA1 needed 20 characters. */
    unsigned int len = 20;
    /* Create and initialize the context */
    HMAC_CTX *ctx = HMAC_CTX_new();
    /* Initialize the HMAC operation. */
    HMAC_Init_ex(ctx, key, key_len, EVP_sha1(), NULL);
    /* Provide the message to HMAC, and start HMAC authentication.
       */
    HMAC_Update(ctx, (unsigned char *)message, strlen(message));
    /* HMAC_Final() writes the hashed values to md, which must have
        enough
    space for the hash function output. */
    HMAC_Final(ctx, mac, &len);
    /* Releases any associated resources and finally frees context
       variable
     */
    HMAC_CTX_free(ctx);
    return;
}
```

2. Server code

   (a) Screenshot or well-formatted copy of code.

   Here is the server code used for this assignment.

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

// open ssl
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>

#include <errno.h>  // for perror()
#include <unistd.h> // for close()

#include <string>
#include <cstring>
#include <cstdlib> // for atoi()
#include <vector>

// load environment variables from .env file, and additional
   helper functions
#include "helper.h"

using std::string;
using std::vector;

int main(void)
{
    printf("Server starting...\n");
    load_env(".env");

    // Declare variables
    const char *server_ip = getenv("SERVER_IP");
    const int server_port = atoi(getenv("SERVER_PORT"));
    char client_message[2048];
    char server_message[2048];
    const char *custom_message_success = "Server: Hello from
       server, message authenticated!\n";
    const char *custom_message_failed = "Server: Hello from
       server, message failed authentication!\n";

    // debug
```

```
printf("Server starting at IP: %s, Port: %d\n", server_ip,
   server_port);

// Create socket
const int server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket == -1)
{
    perror("Failed to create socket");
    return 1;
}

// Bind to the set port and IP
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(server_port);
inet_pton(AF_INET, server_ip, &server_addr.sin_addr);

if (bind(server_socket, (struct sockaddr *)&server_addr,
   sizeof(server_addr)) == -1)
{
    perror("Failed to bind socket");
    close(server_socket);
    return 1;
}
printf("Done with binding with IP: %s, Port: %d\n",
   server_ip, server_port);

// Listen for clients:
const char *client_ip = getenv("CLIENT_IP");
if (listen(server_socket, 1) == -1)
{
    perror("Failed to listen on socket");
    close(server_socket);
    return 1;
}
printf("Listening for incoming connections...\n");

// Accept an incoming connection
struct sockaddr_in client_addr;
socklen_t client_addr_len = sizeof(client_addr);
int client_socket = accept(server_socket, (struct sockaddr
   *)&client_addr, &client_addr_len);
if (client_socket == -1)
{
    perror("Failed to accept connection");
```

```
        close(server_socket);
        return 1;
    }
    printf("Client connected at IP: %s \n", client_ip);

    // clean exising buffer
    memset(client_message, 0, sizeof(client_message));

    // store message history, for HMAC calculation
    // msg on odd is msg, and on even is corresponding HMAC
    vector<string> message_history;

    // Receive client's message
    while (1)
    {
        // clean exising buffer
        memset(client_message, 0, sizeof(client_message));

        ssize_t n = recv(client_socket, client_message, sizeof(
            client_message), 0);
        if (n == 0)
            break;
        if (n < 0)
        {
            perror("recv");
            break;
        }

        if (strcmp((char *)client_message, "\\exit\n") == 0)
            break;

        printf("Msg from client: %s \n", byte_to_hex((unsigned
            char *)client_message, n));
        // store message history for HMAC calculation
        message_history.push_back(string((char *)client_message
            , n));

        // only respond after receiving both msg and HMAC
        if (message_history.size() % 2 == 1)
            continue;

        // check HMAC
        const string expected_hmac_str = message_history.back()
            ;
        unsigned char expected_hmac[20];
        memcpy(expected_hmac, expected_hmac_str.c_str(), 20);
        const string plaintext_str = message_history[
            message_history.size() - 2];
        char plaintext_cstr[2048];
        strcpy(plaintext_cstr, plaintext_str.c_str());
        printf("Plaintext: %s, string length: %d\n",
            byte_to_hex((unsigned char *)plaintext_cstr, strlen(
            plaintext_cstr)), strlen(plaintext_cstr));

        unsigned char calculated_hmac[20];
        memset(calculated_hmac, 0, sizeof(calculated_hmac));
        cal_hmac(calculated_hmac, plaintext_cstr);

        if (memcmp(expected_hmac, calculated_hmac, 20) != 0)
        {
            fprintf(stderr, "HMAC mismatch\n, expected: %s, 
                calculated: %s\n", byte_to_hex(expected_hmac,
                20), byte_to_hex(calculated_hmac, 20));
            memcpy(server_message, custom_message_failed,
                strlen(custom_message_failed))
            ;
        }
        else
        {
            memcpy(server_message, custom_message_success,
                strlen(custom_message_success))
            ;
        }
        // Respond to client
        // prepare server message

        size_t reply_len = strlen(server_message);
        if (send(client_socket, server_message, reply_len, 0)
            == -1)
        {
            perror("Send failed");
            break;
        }

        printf("Response sent to client: %s\n", server_message)
            ;
    }
    // Close the socket
    close(client_socket);
    close(server_socket);
```

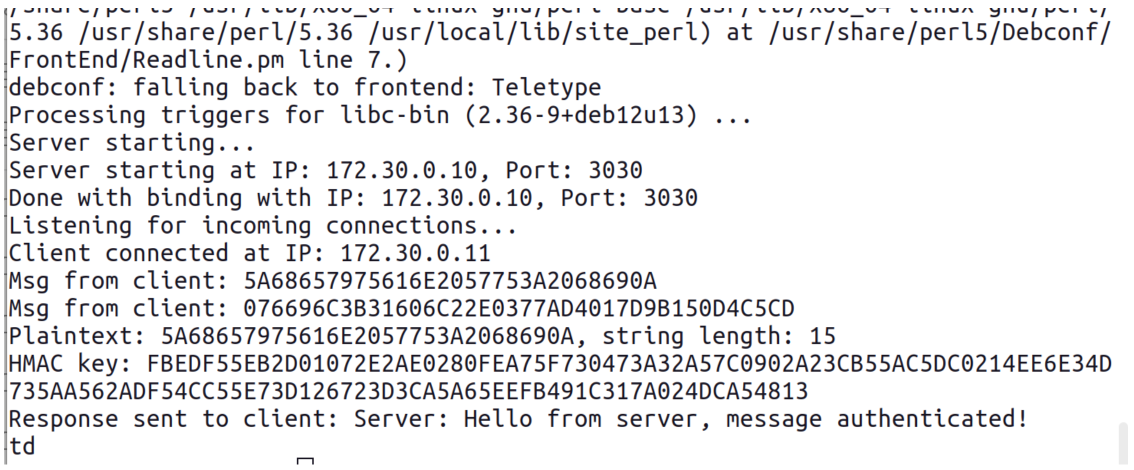
```
        return 0;
    }
```

(b) Quick overview of what the code does in your own words.

We let the server listen for incoming connection, and once the client sends both message and HMAC, the server will calculate the HMAC of the message and compare it with the received HMAC, if they match, the server will respond with a success message, otherwise, it will respond with a failure message.

(c) Screenshots showing your server program during/after execution.

Here is the screenshot from the server side receiving the message from the client.



```
5.36 /usr/share/perl/5.36 /usr/local/lib/site_perl) at /usr/share/perl5/Debconf/
FrontEnd/Readline.pm line 7.)
debconf: falling back to frontend: Teletype
Processing triggers for libc-bin (2.36-9+deb12u13) ...
Server starting...
Server starting at IP: 172.30.0.10, Port: 3030
Done with binding with IP: 172.30.0.10, Port: 3030
Listening for incoming connections...
Client connected at IP: 172.30.0.11
Msg from client: 5A68657975616E2057753A2068690A
Msg from client: 076696C3B31606C22E0377AD4017D9B150D4C5CD
Plaintext: 5A68657975616E2057753A2068690A, string length: 15
HMAC key: FBEDF55EB2D01072E2AE0280FEA75F730473A32A57C0902A23CB55AC5DC0214EE6E34D
735AA562ADF54CC55E73D126723D3CA5A65EEFB491C317A024DCA54813
Response sent to client: Server: Hello from server, message authenticated!
td
```

3. Client code

   (a) Screenshot or well-formatted copy of code

   Here is the client code used for this assignment.

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

// open ssl
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>

#include <errno.h>  // error handling
#include <unistd.h> // for close()

#include <string>
#include <cstring>
#include <cstdlib> // for atoi()
#include <vector>

#include "helper.h"

int main(void)
{
    load_env(".env");
    // Declare variables
    const char *server_ip = std::getenv("SERVER_IP");
    const int server_port = std::atoi(std::getenv("SERVER_PORT"
        ));
    printf("Connecting to server %s:%d\n", server_ip,
        server_port);

    char client_message[1024];
    char server_message[1024];
    const char *custom_message = "Zheyuan Wu: ";

    // Create socket:
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1)
    {
        perror("Failed to create socket");
        return 1;
    }
    else
    {
        printf("Socket created successfully\n");
    }

    // Send connection request to server, be sure to set por
       tand IP the same as server-side
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(server_port);
    inet_pton(AF_INET, server_ip, &server_addr.sin_addr);

    if (connect(client_socket, (struct sockaddr *)&server_addr,
        sizeof(server_addr)) == -1)
    {
        perror("Failed to connect to server");
        close(client_socket);
        return 1;
    }
    else
    {
        printf("Connected to server successfully\n");
    }

    // Get input from the user:
    printf("Enter message sent to the server (type \\quit to
       exit): ");
    // clean the buffer
    memset(client_message, 0, sizeof(client_message));
    if (fgets(client_message, sizeof(client_message), stdin) ==
         NULL)
    {
        // EOF or error reading from stdin, exit the loop
        perror("Error reading from stdin");
        return 1;
    }

    while (strcmp(client_message, "\\quit\n") != 0)
    {

        // Send the message to server:
        // add my name in the front
        char buffer[2048];
        std::snprintf(buffer, sizeof(buffer), "%s%s",
            custom_message, client_message);
        printf("Message sent to server: %s, length: %d \n",
            byte_to_hex((unsigned char *)buffer, strlen(buffer))
            , (int)strlen(buffer));

        ssize_t sent = send(client_socket, buffer, strlen(
            buffer), 0);
        if (sent <= 0)
        {
             perror("No message sent to server");
             break;
        }
        // send hmac of the message
        unsigned char hmac[20];
        memset(hmac, 0, sizeof(hmac));
        cal_hmac(hmac, buffer);
        printf("HMAC sent to server: %s\n", byte_to_hex(hmac,
            20));

        ssize_t sent_hmac = send(client_socket, hmac, strlen((
            char *)hmac), 0);
        if (sent_hmac <= 0)          {
             perror("No HMAC sent to server");
             break;
        }

        // Receive the server's response:
        // add terminator for string
        ssize_t recvd = recv(client_socket, server_message,
            sizeof(server_message) - 1, 0);
        if (recvd <= 0)
        {
             perror("No message received from server");
             break;
        }
        server_message[recvd] = '\0';

        printf("Server's response: %s\n", server_message);

        printf("Enter message sent to the server (type \\quit
            to exit): ");

        // clean the buffer
        memset(client_message, 0, sizeof(client_message));
        memset(server_message, 0, sizeof(server_message));
```

```
        if (fgets(client_message, sizeof(client_message), stdin
            ) == NULL)
        {
            // EOF or error reading from stdin, exit the loop
            perror("Error reading from stdin");
            break;
        }
    }

    // Close the socket
    close(client_socket);

    return 0;
}
```

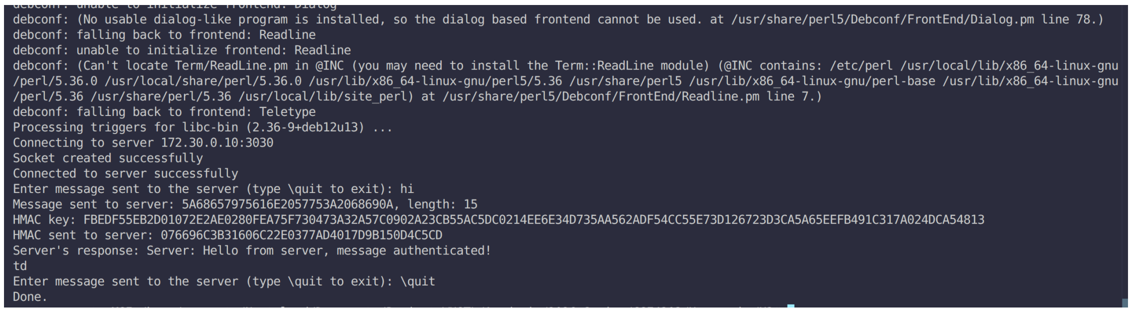
(b) Quick overview of what the code does in your own words.

The same as basic protocol in the assignment 0, we send two message for each user input, one is the plaintext message, and the other is the HMAC of the message.

Then we wait for server response, if the HMAC is correct, the server will respond with a success message, otherwise, it will respond with a failure message.

(c) Screenshots showing your client program during/after execution

Here is the screenshot from the client side sending message to the server and server response with plain text.



```
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 78.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC contains: /etc/perl /usr/local/lib/x86_64-linux-gnu
/perl/5.36.0 /usr/local/share/perl/5.36.0 /usr/lib/x86_64-linux-gnu/perl5/5.36 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl-base /usr/lib/x86_64-linux-gnu
/perl/5.36 /usr/share/perl/5.36 /usr/local/lib/site_perl) at /usr/share/perl5/Debconf/FrontEnd/Readline.pm line 7.)
debconf: falling back to frontend: Teletype
Processing triggers for libc-bin (2.36-9+deb12u13) ...
Connecting to server 172.30.0.10:3030
Socket created successfully
Connected to server successfully
Enter message sent to the server (type \quit to exit): hi
Message sent to server: 5A68657975616E2057753A2068690A, length: 15
HMAC key: FBEDF55EB2D01072E2AE0280FEA75F730473A32A57C0902A23CB55AC5DC0214EE6E34D735AA562ADF54CC55E73D126723D3CA5A65EEFB491C317A024DCA54813
HMAC sent to server: 076696C3B31606C22E0377AD4017D9B150D4C5CD
Server's response: Server: Hello from server, message authenticated!
td
Enter message sent to the server (type \quit to exit): \quit
Done.
```