Source code and detailed setup scripts are available at https://git.trance-0.com/Trance-0/CSE4303H4.

1. Description of experimental setup

   We used the same docker setup as the previous assignment.

```
services:
  server:
    image: debian:bookworm-slim
    container_name: hw4-server
    working_dir: /app
    env_file:
      - .env
    networks:
      net-hw4:
        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: hw4-client
    working_dir: /app
    env_file:
      - .env
    networks:
      net-hw4:
        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-hw4:
    name: net-hw4
    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
INITIAL_VECTOR=0783F40FBD09EAE9AA3E5F295414074E
SECRET_KEY=
    F57D76F79AEE1CDDB399865653E1871E2E23619C1ADA333AEDBA387C0E5472A2
ADD=whatever metadata you want authenticated
```

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

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

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;
}
```

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 gcm_decrypt(unsigned char *ciphertext, int ciphertext_len,
                unsigned char *aad, int aad_len,
                unsigned char *tag,
                unsigned char *key,
                unsigned char *iv, int iv_len,
                unsigned char *plaintext)
{
    EVP_CIPHER_CTX *ctx;
    int len, plaintext_len, ret;
    /* Create and initialize the context */
    ctx = EVP_CIPHER_CTX_new();
    if (!ctx)
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
```

```
    }

    /* Initialize the decryption operation. */
    if (!EVP_DecryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, key,
        iv))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /* Set IV length. Not necessary if this is 12 bytes (96
        bits) */
    if (!EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN,
        iv_len, NULL))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /* Initialize key and IV */
    if (!EVP_DecryptInit_ex(ctx, NULL, NULL, key, iv))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /*
     * Provide any AAD data. This can be called zero or more
         times as
     * required
     */
    if (!EVP_DecryptUpdate(ctx, NULL, &len, aad, aad_len))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }

    /*
    * Provide the message to be decrypted, and obtain the
        plaintext
output.
    * EVP_DecryptUpdate can be called multiple times if
        necessary
    */
```

```
    if (!EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext,
        ciphertext_len))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    plaintext_len = len;
    /* Set expected tag value. Works in OpenSSL 1.0.1d and
        later */
    if (!EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_TAG, 16, tag
        ))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /*
     * Finalize the decryption. A positive return value
        indicates success,
     * and anything else is a failure - the plaintext is not
        trustworthy.
     */
    ret = EVP_DecryptFinal_ex(ctx, plaintext + len, &len);
    /* Clean up */
    EVP_CIPHER_CTX_free(ctx);
    if (ret > 0)
    {
        /* Success */
        plaintext_len += len;
        return plaintext_len;
    }
    else
    {
        /* Verify failed */
        return 0;
    }
}

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);

// encryption parameters
const char *inital_vector = std::getenv("INITIAL_VECTOR");
const char *secret_key = std::getenv("SECRET_KEY");
const char *add = std::getenv("ADD");
unsigned char key_bytes[32], iv_bytes[16];

int key_len = hex_to_bytes_upper(secret_key, key_bytes,
    sizeof(key_bytes));
int iv_len = hex_to_bytes_upper(inital_vector, iv_bytes,
    sizeof(iv_bytes));

if (key_len != 32 || iv_len != 16)
{
    fprintf(stderr, "Invalid key/IV size for AES-256-GCM\n"
        );
    return 1;
}

// 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 TAG calculation
// msg on odd is msg, and on even is corresponding TAG
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 TAG calculation
message_history.push_back(string((char *)client_message
    , n));

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

// check TAG
const string tag_str = message_history.back();
unsigned char tag[16];
memcpy(tag, tag_str.c_str(), 16);
const string ciphertext_str = message_history[
    message_history.size() - 2];
char ciphertext_cstr[2048];
strcpy(ciphertext_cstr, ciphertext_str.c_str());

unsigned char plaintext[2048];
memset(plaintext, 0, sizeof(plaintext));
int plaintext_len = gcm_decrypt((unsigned char *)
    ciphertext_cstr,
            strlen(ciphertext_cstr),
            (unsigned char *)add,
            strlen(add),
            tag,
            key_bytes,
            iv_bytes,
            16,
            plaintext);
```

```
            if (plaintext_len == 0)
            {
                fprintf(stderr, "TAG mismatch or invalid message\n"
                    );
                memcpy(server_message, custom_message_failed,
                    strlen(custom_message_failed));
                ;
            }
            else
            {
                fprintf(stderr, "Decryption success\n, decrypted
                    message: %s\n", plaintext);
                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 use `aes-256-gcm` to decrypt the message, with period of 2, we load cipher text and TAG from the message history received from the client.

Then we use `gcm_decrypt` to decrypt the message, and check the TAG on the process of decryption.

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

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

```
Setting up libelf1:amd64 (0.188-2.1) ...
Setting up libbpf1:amd64 (1:1.1.2-0+deb12u1) ...
Setting up libgssapi-krb5-2:amd64 (1.20.1-2+deb12u4) ...
Setting up libtirpc3:amd64 (1.3.3+ds-1) ...
Setting up iproute2 (6.1.0-3) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
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) ...
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: 8474F7FAEF0FA9C67ACBD29C368506F788D0
Msg from client: 01D390B8991BB873EDF434CC9B2607DB
Response sent to client: Server: Hello from server, message authenticated!
@
Decryption success
, decrypted message: Zheyuan Wu: tired
```

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"

// GCM encryption
int gcm_encrypt(unsigned char *plaintext, int plaintext_len,
                unsigned char *aad, int aad_len,
                unsigned char *key,
                unsigned char *iv, int iv_len,
                unsigned char *ciphertext,
                unsigned char *tag)
{
    EVP_CIPHER_CTX *ctx;
    int len, ciphertext_len;
    /* Create and initialize the context */
    ctx = EVP_CIPHER_CTX_new();
    if (!ctx)
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /* Initialize the encryption operation. */
    if (!EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, key,
       iv))
```

```
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /*
     * Set IV length if default 12 bytes (96 bits) is not
        appropriate
     */
    if (!EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN,
       iv_len, NULL))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    };
    /* Initialize key and IV */
    if (!EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }

    /*
     * Provide any AAD data. This can be called zero or more
        times as
     * required
     */
    if (!EVP_EncryptUpdate(ctx, NULL, &len, aad, aad_len))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }

    /*
    * Provide the message to be encrypted, and obtain the
       encrypted
output.
    * EVP_EncryptUpdate can be called multiple times if
       necessary
    */
    if (!EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext,
       plaintext_len))
```

```
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    ciphertext_len = len;
    /*
     * Finalize the encryption. Normally ciphertext bytes may
        be written at
     * this stage, but this does not occur in GCM mode
     */
    if (!EVP_EncryptFinal_ex(ctx, ciphertext + len, &len))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    ciphertext_len += len;
    /* Get the tag */
    if (!EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag
        ))
    {
        ERR_print_errors_fp(stderr);
        EVP_CIPHER_CTX_free(ctx);
        return 0;
    }
    /* Clean up */
    EVP_CIPHER_CTX_free(ctx);
    return ciphertext_len;
}

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: ";
```

```
// encryption parameters
const char *inital_vector = std::getenv("INITIAL_VECTOR");
const char *secret_key = std::getenv("SECRET_KEY");
const char *add = std::getenv("ADD");
unsigned char key_bytes[32], iv_bytes[16];

int key_len = hex_to_bytes_upper(secret_key, key_bytes,
    sizeof(key_bytes));
int iv_len = hex_to_bytes_upper(inital_vector, iv_bytes,
    sizeof(iv_bytes));

if (key_len != 32 || iv_len != 16)
{
    fprintf(stderr, "Invalid key/IV size for AES-256-GCM\n"
        );
    return 1;
}

// 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",
        buffer, (int)strlen(buffer));

    unsigned char tag[16] = {0};
    unsigned char cipher_text[2048] = {0};
    int cipher_text_len = gcm_encrypt((unsigned char *)
        buffer,
                (int)strlen(buffer),
                (unsigned char *)add,
                (int)strlen(add),
                key_bytes,
                iv_bytes,
                iv_len,
                cipher_text,
                tag);

    printf("Ciphertext sent to server: %s\n", byte_to_hex(
        cipher_text, cipher_text_len));
    ssize_t sent = send(client_socket, cipher_text,
```

```
    cipher_text_len, 0);
if (sent <= 0)
{
    perror("No message sent to server");
    break;
}
sleep(1);
// send tag of the message
printf("TAG sent to server: %s\n", byte_to_hex(tag, 16)
    );

ssize_t sent_tag = send(client_socket, tag, 16, 0);
if (sent_tag <= 0)
{
    perror("No TAG 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.

We use `aes-256-gcm` for encryption, we read the add, initial vector, and key from environment variable. First we encrypt the add, then we encrypt the message, and finally we send the ciphertext to the server.

(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.

```
Setting up libgssapi-krb5-2:amd64 (1.20.1-2+deb12u4) ...
Setting up libtirpc3:amd64 (1.3.3+ds-1) ...
Setting up iproute2 (6.1.0-3) ...
debconf: unable to initialize frontend: Dialog
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): tired
Message sent to server: Zheyuan Wu: tired
, length: 18
Ciphertext sent to server: 8474F7FAEF0FA9C67ACBD29C368506F788D0
TAG sent to server: 01D390B8991BB873EDF434CC9B2607DB
Server's response: Server: Hello from server, message authenticated!
@
Enter message sent to the server (type \quit to exit): \quit
Done.
```