

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#pragma comment(lib, "ws2_32.lib")
#pragma comment(lib, "libssl.lib")
#pragma comment(lib, "libcrypto.lib")

#define PORT 60000
#define AES_KEY_LENGTH 32 // 256 bits
#define AES_IV_LENGTH 12 // 96 bits for GCM

void initialize_openssl() {
    SSL_load_error_strings();
    OpenSSL_add_ssl_algorithms();
}

void cleanup_openssl() {
    EVP_cleanup();
}

SSL_CTX* create_context() {
    const SSL_METHOD* method;
    SSL_CTX* ctx;

    method = TLS_client_method();
    ctx = SSL_CTX_new(method);
    if (!ctx) {
        perror("Unable to create SSL context");
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    return ctx;
}

void configure_context(SSL_CTX* ctx) {
    if (SSL_CTX_load_verify_locations(ctx, "rootCA.crt", NULL) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
    if (SSL_CTX_set_cipher_list(ctx, "ECDHE-ECDSA-AES256-GCM-SHA384") <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
}
```

```
void aes_gcm_encrypt(const unsigned char* plaintext, int plaintext_len,
    const unsigned char* key, const unsigned char* iv,
    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, AES_IV_LENGTH, NULL);
    EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv);

    int len;
    EVP_EncryptUpdate(ctx, NULL, &len, NULL, plaintext_len);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len);
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag);

    EVP_CIPHER_CTX_free(ctx);
}

void aes_gcm_decrypt(const unsigned char* ciphertext, int ciphertext_len,
    const unsigned char* tag, const unsigned char* key,
    const unsigned char* iv, unsigned char* plaintext) {
    EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
    EVP_DecryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL, NULL);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN, AES_IV_LENGTH, NULL);
    EVP_DecryptInit_ex(ctx, NULL, NULL, key, iv);

    int len;
    EVP_DecryptUpdate(ctx, NULL, &len, NULL, ciphertext_len);
    EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, ciphertext_len);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_TAG, 16, (void*)tag);

    if (EVP_DecryptFinal_ex(ctx, plaintext + len, &len) <= 0) {
        printf("Decryption failed\n");
    }

    EVP_CIPHER_CTX_free(ctx);
}

int main() {
    WSADATA wsaData;
    SSL_CTX* ctx;
    SSL* ssl;
    int server;

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        perror("WSAStartup failed");
        exit(EXIT_FAILURE);
    }

    // Initialize OpenSSL
    initialize_openssl();
    ctx = create_context();
    configure_context(ctx);
}
```

```
// Create socket
server = socket(AF_INET, SOCK_STREAM, 0);
if (server < 0) {
    perror("Unable to create socket");
    exit(EXIT_FAILURE);
}

// Initialize sockaddr_in
struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
if (inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr) <= 0) {
    perror("inet_pton failed");
    closesocket(server);
    SSL_CTX_free(ctx);
    cleanup_openssl();
    WSACleanup();
    exit(EXIT_FAILURE);
}

// Connect to server
if (connect(server, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
    perror("Unable to connect");
    closesocket(server);
    SSL_CTX_free(ctx);
    cleanup_openssl();
    WSACleanup();
    exit(EXIT_FAILURE);
}

// Create SSL structure and set the file descriptor
ssl = SSL_new(ctx);
SSL_set_fd(ssl, server);

// Perform SSL/TLS handshake with server
if (SSL_connect(ssl) <= 0) {
    ERR_print_errors_fp(stderr);
}
else {
    // Send message to server
    const char* msg = "Hello, Eliza!";
    unsigned char key[AES_KEY_LENGTH] = {
        0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
        0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
        0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32, 0x33,
        0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x31
    };
    unsigned char iv[AES_IV_LENGTH] = {
        0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
        0x38, 0x39, 0x30, 0x31
    };
};
```

```
    unsigned char ciphertext[256];
    unsigned char tag[16];
    aes_gcm_encrypt((unsigned char*)msg, strlen(msg), key, iv,
        ciphertext, tag);

    SSL_write(ssl, ciphertext, strlen(msg));
    SSL_write(ssl, tag, 16);

    // Receive encrypted message from server
    unsigned char received_ciphertext[256];
    unsigned char received_tag[16];
    int bytes = SSL_read(ssl, received_ciphertext, sizeof
        (received_ciphertext));
    if (bytes > 0) {
        SSL_read(ssl, received_tag, 16);
        unsigned char plaintext[256];
        aes_gcm_decrypt(received_ciphertext, bytes, received_tag, key,
            iv, plaintext);
        printf("Received: %s\n", plaintext);
    }
    else {
        ERR_print_errors_fp(stderr);
    }
}

// Cleanup
SSL_shutdown(ssl);
SSL_free(ssl);
closesocket(server);
SSL_CTX_free(ctx);
cleanup_openssl();
WSACleanup();

return 0;
}
```