server enhanced

#include <iostream>

#include <chrono>

#include <thread>

#include <random>

#include <openssl/aes.h>

#include <openssl/rand.h>

#include <boost/asio.hpp>

#include <Eigen/Dense>

#include <pykalman/pykalman.h>

using namespace std;

using namespace boost::asio;

// Constants for server configuration

const int SERVER\_PORT = 5005; // Server listening port

const int CLIENT\_PORT = 5006; // Client receiving port

const int DATA\_DIMENSION = 2; // Number of dimensions (position, velocity)

const int AES\_KEY\_SIZE = 32; // AES-256 key size

// Function to collect random telemetry data

Eigen::Vector2d collectTelemetryData() {

static std::default\_random\_engine generator;

std::uniform\_real\_distribution<double> positionDist(0.0, 1000.0);

std::uniform\_real\_distribution<double> velocityDist(50.0, 200.0);

double position = positionDist(generator);

double velocity = velocityDist(generator);

return Eigen::Vector2d(position, velocity);

}

// Function to encrypt data using AES-256

void encryptData(const Eigen::Vector2d& data, unsigned char\* encryptedData, const unsigned char\* key) {

AES\_KEY encryptKey;

AES\_set\_encrypt\_key(key, AES\_KEY\_SIZE \* 8, &encryptKey); // Key size in bits

unsigned char iv[AES\_BLOCK\_SIZE];

RAND\_bytes(iv, sizeof(iv)); // Generate a random IV

AES\_cbc\_encrypt(data.data(), encryptedData, sizeof(Eigen::Vector2d), &encryptKey, iv, AES\_ENCRYPT);

}

// Function to initialize the Kalman Filter

KalmanFilter initializeKalmanFilter() {

KalmanFilter kf;

kf.initialize(DATA\_DIMENSION); // Initialize Kalman Filter for 2D state

return kf;

}

// Main server function

int main() {

try {

// Set up the server socket

io\_service ioService;

ip::udp::socket socket(ioService, ip::udp::endpoint(ip::udp::v4(), SERVER\_PORT));

// Generate a random encryption key (32 bytes for AES-256)

unsigned char key[AES\_KEY\_SIZE];

RAND\_bytes(key, sizeof(key));

cout << "Server Encryption Key (hex): ";

for (int i = 0; i < sizeof(key); ++i) {

printf("%02x", key[i]);

}

cout << endl;

// Initialize the Kalman Filter

KalmanFilter kf = initializeKalmanFilter();

while (true) {

// Collect telemetry data and apply Kalman Filter

Eigen::Vector2d rawData = collectTelemetryData();

Eigen::Vector2d smoothedData = kf.update(rawData); // Apply Kalman Filter

// Encrypt the smoothed data

unsigned char encryptedData[sizeof(Eigen::Vector2d)];

encryptData(smoothedData, encryptedData, key);

// Send encrypted data to the client

socket.send\_to(buffer(encryptedData, sizeof(encryptedData)),

ip::udp::endpoint(ip::address::from\_string("127.0.0.1"), CLIENT\_PORT));

cout << "Sent encrypted data: (" << smoothedData[0] << ", " << smoothedData[1] << ")" << endl;

// Simulate data collection delay

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

} catch (const std::exception& e) {

cerr << "Error: " << e.what() << endl;

}

return 0;

}

clientenhanced

#include <iostream>

#include <boost/asio.hpp>

#include <openssl/aes.h>

#include <SFML/Graphics.hpp>

#include <Eigen/Dense>

using namespace std;

using namespace boost::asio;

// Constants for client configuration

const int CLIENT\_PORT = 5006; // Client receiving port

const int DATA\_DIMENSION = 2; // Number of dimensions (position, velocity)

const int AES\_KEY\_SIZE = 32; // AES-256 key size

// Function to decrypt data using AES-256

Eigen::Vector2d decryptData(const unsigned char\* encryptedData, const unsigned char\* key) {

AES\_KEY decryptKey;

AES\_set\_decrypt\_key(key, AES\_KEY\_SIZE \* 8, &decryptKey); // Key size in bits

unsigned char iv[AES\_BLOCK\_SIZE] = {0}; // Using a zeroed IV for demo purposes

Eigen::Vector2d decryptedData;

AES\_cbc\_encrypt(encryptedData, decryptedData.data(), sizeof(Eigen::Vector2d), &decryptKey, iv, AES\_DECRYPT);

return decryptedData;

}

// Main client function

int main() {

try {

// Set up the client socket

io\_service ioService;

ip::udp::socket socket(ioService, ip::udp::endpoint(ip::udp::v4(), CLIENT\_PORT));

// Use a placeholder key for demonstration purposes (should match the server's key)

unsigned char key[AES\_KEY\_SIZE] = {0};

// Set up SFML for visualization

sf::RenderWindow window(sf::VideoMode(800, 600), "Missile Trajectory Visualization");

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed)

window.close();

}

// Receive encrypted data from the server

unsigned char encryptedData[sizeof(Eigen::Vector2d)];

ip::udp::endpoint senderEndpoint;

size\_t receivedBytes = socket.receive\_from(buffer(encryptedData), senderEndpoint);

// Validate received data size

if (receivedBytes != sizeof(Eigen::Vector2d)) {

cerr << "Received data size mismatch: expected " << sizeof(Eigen::Vector2d)

<< ", got " << receivedBytes << endl;

continue; // Skip this iteration if data is malformed

}

// Decrypt the received data

Eigen::Vector2d decryptedData = decryptData(encryptedData, key);

// Clear the window

window.clear(sf::Color::Black);

// Visualize the trajectory

sf::CircleShape point(5); // Circle representing the missile position

point.setFillColor(sf::Color::Red);

point.setPosition(decryptedData[0], decryptedData[1]); // Position in the window

window.draw(point);

window.display(); // Render the window

}

} catch (const std::exception& e) {

cerr << "Error: " << e.what() << endl;

}

return 0;

}