Simulation of TLS (42)

Leskovar Lukas Andreios (KatNr), 5BHIF

March 2022

Contents

1 Introduction

The goal of this project was to simulate communication over Transport Layer Security (TLS) by implementing the Diffie-Hellman Internet Key Exchange (IKE). Any further communication was to be encrypted by a symmetric encryption algorithm.

2 Implementation

2.1 TLS 1.0

2.1.1 Key Generation

2.1.2 Handshake

3 Software Architecture

3.1 Technologies

Purpose	Technology
Build Tool	Meson
Command line interface	CLI11
Configuration files	json
Data serialization	Protobuf
Logging	spdlog
Network Communication	asio
Programming Languages	C++ 17
Encryption	plusaes
Hashing	PicoSHA2
Large Integer Values	BigInt

Table 1: This table lists all the technologies used in this project.

3.2 Classes

3.3 Communication

4 Description of code-blocks

4.1 Asio

Network communication between client and server is established by utilizing asio.

4.1.1 Client Connection

```
TLS_Client::TLS_Client(
  asio::io_context& io_context,
  std::string host,
  std::string port
  )
  : io_context(io_context),
  resolver(io_context),
  socket(io_context)
{
  endpoints = resolver.resolve(host, port);
  asio::connect(socket, endpoints);
  session = std::make_shared<Session>(std::move(socket), 0);
  session->start();
  spdlog::info("Client - Connected to {}:{}", host, port);

  handshake_agent = std::make_shared<TLS_Handshake_Agent>(session);
  session->subscribe(handshake_agent);
}
```

Source Code 1: Creation of socket connection on client side.

4.1.2 Server Connection

```
void TLS_Server::start_accept() {
  spdlog::info("Server - Starting accept");

  acceptor.async_accept(
  [this](const std::error_code& ec, asio::ip::tcp::socket socket) {
    if (!ec) {
      spdlog::info("Server - Accepted connection");
      auto new_session =
        std::make_shared<Session>(std::move(socket), sessions.size());
      new_session->subscribe(shared_from_this());
      new_session->start();
      sessions.push_back(new_session);

      auto new_handshake_agent =
        std::make_shared<TLS_Handshake_Agent>(new_session);
      new_session->subscribe(new_handshake_agent);
      handshake_agents.push_back(new_handshake_agent);

      start_accept();
    } else {
      spdlog::error("Server - Error accepting connection: {}", ec.message());
    }
  });
}
```

Source Code 2: Server asynchronously waiting for client connections.

4.2 Protobuf

Any data to be sent over TCP is serialized using Google Protobuf.

4.2.1 Message Serialization

```
void Pipe::send(google::protobuf::Message& message) {
  u_int64_t message_size{message.ByteSizeLong()};
  asio::write(*socket, asio::buffer(&message_size, sizeof(message_size)));
```

```
  asio::streambuf buffer;
  std::ostream os(&buffer);
  message.SerializeToOstream(&os);
  asio::write(*socket, buffer);

  spdlog::debug("Pipe - Sent message");
}
```

Source Code 3: Server asynchronously waiting for client connections.

4.3 TLS Handshake

Whenever a new message is received the TLS_Handshake_Agent class is responsible for handling and responding to any handshake related message.

4.4 Message Handling

```
void TLS_Handshake_Agent::notify(
  tls::MessageWrapper message,
  unsigned int session_id
) {
  if (currentState == State::UNSECURED
   || currentState == State::ESTABLISHING) {
    handle_message(message);
  }
}
```

Source Code 4: Handshake Agent getting notified of new message.

4.5 External Libraries

4.5.1 CLI11

CLI11 implements a basic Comand Line Interface (CLI) where users are able to specify parameters relevant for the program.

```
CLI::App app{"tls_client"};

std::string host = "localhost";
std::string port = "4433";
spdlog::level::level_enum log_level = spdlog::level::info;
std::map<std::string, spdlog::level::level_enum> log_level_map = {
  {"trace", spdlog::level::trace},
  {"debug", spdlog::level::debug},
  {"info", spdlog::level::info},
  {"warn", spdlog::level::warn},
  {"error", spdlog::level::err},
  {"critical", spdlog::level::critical}
};


app.add_option("-n,--hostname", host, "Hostname");
app.add_option("-p,--port", port, "Port");
app.add_option("-l,--log-level", log_level, "Log level")
  ->transform(CLI::CheckedTransformer(log_level_map, CLI::ignore_case));

CLI11_PARSE(app, argc, argv);
```

Source Code 5: Handshake Agent getting notified of new message.

4.5.2 spdlog

To log important information the logging library spdlog is employed.

4.5.3 JSON

Any further information, e.g. prime number for Diffie-Hellman IKE, is stored in a .json file which is read as follows.

5 Usage

5.1 Command Line Arguments

5.1.1 Configuration

6 Project Structure

```
/
├── LICENSE
├── meson_options.txt
├── meson.build
├── README.md
├── CHANGELOG.org
├── modp_primes.json
├── include
│   ├── tls_client.h
│   └── tls_server.h
├── src
│   ├── client.cpp
│   ├── server.cpp
│   ├── tls_client.cpp
│   └── tls_server.cpp
├── doc
│   ├── doc.tex
│   ├── references.bib
│   └── doc.pdf
├── tls_util
│   ├── include
│   │   ├── BigInt.hpp
│   │   ├── picosha2.h
│   │   ├── plusaes.hpp
│   │   ├── messagebuilder.h
│   │   ├── pipe.h
│   │   ├── session.h
│   │   ├── tls_handshake_agent.h
│   │   └── tls_observer.h
│   ├── src
│   │   ├── Message.proto
│   │   ├── pipe.cpp
│   │   ├── session.cpp
│   │   └── tls_handshake_agent.cpp
│   └── meson.build
└── build
```

References