Midterm Project: TinyTLS

I. On Academic Integrity.

1. Any form of discussion with anyone other than the instructor or the TAs is strictly prohibited.

2. Each given skeleton code distribution is *personalized*. There are hidden cheating detectors in the code. If you copy your friend's code, there are many ways we can figure it out. Avoiding the hidden detectors will be difficult and you probably won't have enough time to complete the exam within 24hrs.

3. The given tasks can be implemented in many different ways. So it is unlikely that your code will be similar to your friends. Also, we will be running plagiarism checkers on the submissions. If we find any suspicious similarities, 1) we can simply give you a grade of '0' if we are certain. 2) we will investigate all students who are involved and have them explain their code if necessary.

II. Submission.

1. Your code. You are to compress the entire project directory (e.g., *.zip or *.tar.gz) and submit to "Miderm Submission‘ in Week 9 page of icampus. This is due at 09:00, Apr 20 (Tuesday).

2. Lab report. You are also required to submit a 1-page report by 09:00, Apr 21 (Wednesday). There is no given format. The report is for you to prove that you understood the tasks and you did your own work.

III. Getting Help.

Discord Channel. The TAs will be available on `https://discord.gg/ASeqGDFRaV` during 09:00-17:00 on Apr 19. The TAs won't help you with the tasks, but this is for possible mistakes and unclear instructions regarding the exam.

IV. Grading

1. Code commenting. If your code doesn't work, but you understood the problem. Put them in the code comments. We will give you up to 20% of the Task grade. Therefore, it is a good strategy to comment ALL code lines.

2. Per-task grade portions. The portions of each task in grading is as follows:

TASK	Grade Portion
TASK1	40
TASK2	20
TASK3	20
TASK4	20

3. Academic dishonesty Again, any kind of academic dishonesty will result in exam grade of '0'. We will run plagiarism checkers, and the TAs will also check each code line by line to catch any kind of cheating. We will do everything we can to catch cheating.

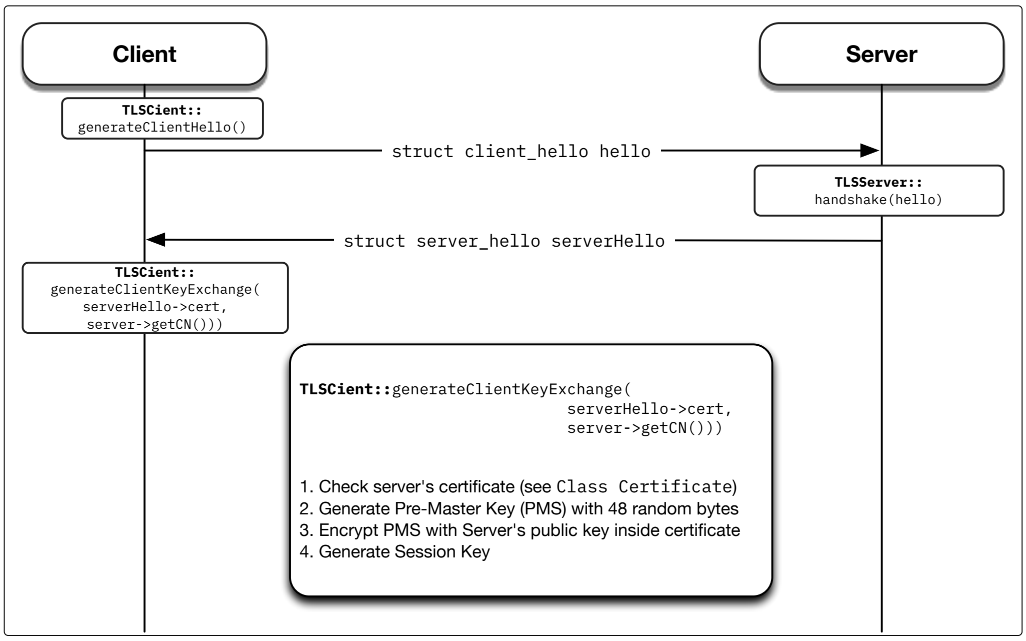
4. Late submission. For each late day, the grade will be counted off by 40%. For instance, if your submission is late by 1 day, the maximum grade your can earn is 60%.

TinyTLS Specifications.

TinyTLS is a simple TLS-like protocol created for education purposes. There is no actual network connections, but the communication is made through message passing between C++ objects (e.g., function calls and argument passing).

Handshake and Key Exchange of TinyTLS. The below diagram depicts the simple handshake and key exchange procedure of Tiny TLS. It uses a simple key exchange algorithm that is an approximation of the RSA key exchange algorithm.

1. The client prepares a Hello message through a function called `generateClientHello()` and sends it.
2. `TLSServer::handshake(struct client_hello)` is the server-side function that handles the client's hello message.The server then sends a `struct server_hello` structure.
3. `TLSClient::generateClientKeyExchange()` has to.. 1. validate server certificate 2. Generate Pre-master Key (PMS) 3. Encrypt PMS with server's public key included in the certificate. 4. Then generate a session key using the PMS.


Client
Server
TLSCient::
generateClientHello()
struct client_hello hello
TLSServer::
handshake(hello)
struct server_hello serverHello
TLSCient::
generateClientKeyExchange(
serverHello->cert,
server->getCN()))
TLSCient::generateClientKeyExchange(
serverHello->cert,
server->getCN()))
1. Check server's certificate (see Class Certificate)
2. Generate Pre-Master Key (PMS) with 48 random bytes
3. Encrypt PMS with Server's public key inside certificate
4. Generate Session Key

Project Code Organization.

```
├── Dockerfile
├── docker_run.sh
├── entry-point.sh
├── include
│   ├── certificate.h
│   ├── mitm.h
│   ├── replay.h
│   ├── tinyTLSClient.h
│   ├── tinyTLS.h
│   └── tinyTLSServer.h
├── integrity.cc     <-- TASK 2 : Implement Message Integrity
├── lib
│   ├── certificate.cc
│   ├── Makefile
│   ├── tinyTLS.cc
│   └── tinyTLSServer.cc
├── main.cc
├── Makefile
├── mitm.cc          <-- TASK 3 : Launch MITM Attack on TinyTLS
├── replay.cc        <-- TASK 4 : Implement Replay Prevention on TinyTLS
└── tinyTLSClient.cc <-- TASK 1 : TinyTLS Client
```

How to develop *without* docker. The only dependency of the TinyTLS is `OpenSSL 1.1.1f`. You can install the package by `sudo apt-get install libssl-dev`, if you use Ubuntu or similar Linux distribution.

How to develop *inside* docker. In a docker-enabled environment (e.g., your linux VM), run `docker_run.sh` to enter the container.

How to build project. Simply type `make` in project root directory

Task Specifications.

Here we provide a specification for your tasks. The common requirement for all tasks is that *you can only use the given functions and the openssl library*. `https://www.openssl.org/docs/`

TASK1. TinyTLS Client:

Your implementation of `generateClientHello()` must perform the following tasks:

Subtask 1 Validate server's certificate (hint: examine `Class Certificate`)

Subtask 2 Generate Pre-Master Secret (PMS). The PMS must be exactly 48 random bytes.

Subtask 3 Encrypt PMS with the server's public key. Note that the server's public key is a RSA public key

Subtask 4 Then you should generate a session key. In TinyTLS, a session key is a `SHA256` hash of the PMS.

TASK2. Implement Message Authentication in TinyTLS. Even though, we get a secure channel after a successful handshake. This doesn't mean that the attacker cannot *modify* the messages. TinyTLS uses `AES-CBC` which does not have built-in message authentication. Implement a message integrity in TinyTLS in `${ProjectRoot}/integrity.cc`. Your message integrity method must provide similar level of security as TLS (i.e., see how TLS achieves message integrity).

TASK3. Launch a Man-In-The-Middle Attack on Tiny TLS The CA key has been leaked as you will see in `mitm() in ${ProjectRoot}/mitm.cc`. Your job is to launch a MITM attack to succeed in making a (illegitimate) handshake as a middle man. You should get a `PASS!` message when you succeed (see `${ProjectRoot}/main.cc`).

TASK4. Prevent a Replay Attack on Tiny TLS Thank you for implementing message integrity in TASK2. However, there remains another vulnerability in

TinyTLS. How can you make TinyTLS secure from replay attacks? For this one, you should also get a `PASS!` message when you succeed (see `${ProjectRoot}/replay.cc` and `${ProjectRoot}/main.cc`).