

50.005 Programming Assignment 2

Authors: Vincent Setiawan, Jing Yu Koh

This application implements a secure file upload application from a client to an Internet file server with the following requirements:

1. Before the upload, the identity of the file server is authenticated using public key cryptography.
2. During the upload, the data is encrypted. We implement two encryption strategies: the first using public key cryptography (RSA), and the second that uses a session key (AES). We compare and benchmark the two strategies.

This program is written in Java and was tested in Java 8.

Problem with Original Protocol

The issue with the original protocol is that it does not prevent a playback attack. In order to tackle this, we introduced a nonce into our system. The nonce is generated by the client and encrypted before being sent to the server.

Design Challenge

Additionally, our client opens multiple sockets for sending and receiving data. The client will open a number of threads specified by the server (e.g., 7). Each thread takes up a new port number, and the server accepts the corresponding threads. The file is then transferred in parallel over all the threads available.

Similarly, the server receives the corresponding part of the file specified by a port, and combines it in the same way that it was split to ensure a lossless file output.

Prerequisites

Java is required to build the program.

Compiling the Program

The program is compiled using the `javac` command:

```
javac ClientSecure.java ServerSecure.java
```

This produces the necessary class files for running the program in the next step.

Running the Program

We test our program on two separate devices connected to the same network, the server and the client.

Server

On the server, run

```
java ServerSecure
```

This will prompt the server to enter what encryption mode it expects (RSA vs AES).

Client

Similarly, on the client, run

```
java ClientSecure
```

The user will be prompted to select the encryption mode as well. **Note:** It is expected that the client will choose the same encryption mode as the server. If not, the output of the server is not guaranteed to be valid.

Next, the user is asked for the file to be transferred. Enter the name of the file in the current working directory, such as `rr.txt` (the Mona Lisa photo). Afterwards, the file will be transferred to the server and saved to the `recv/` folder.

Specifications

The specification of the protocols are as follows:

AP

CP-1

CP-2

Example of Successful Output

Upon a successful execution, the client will close its connection to the server and report the running time. The server will patiently await the next connection.

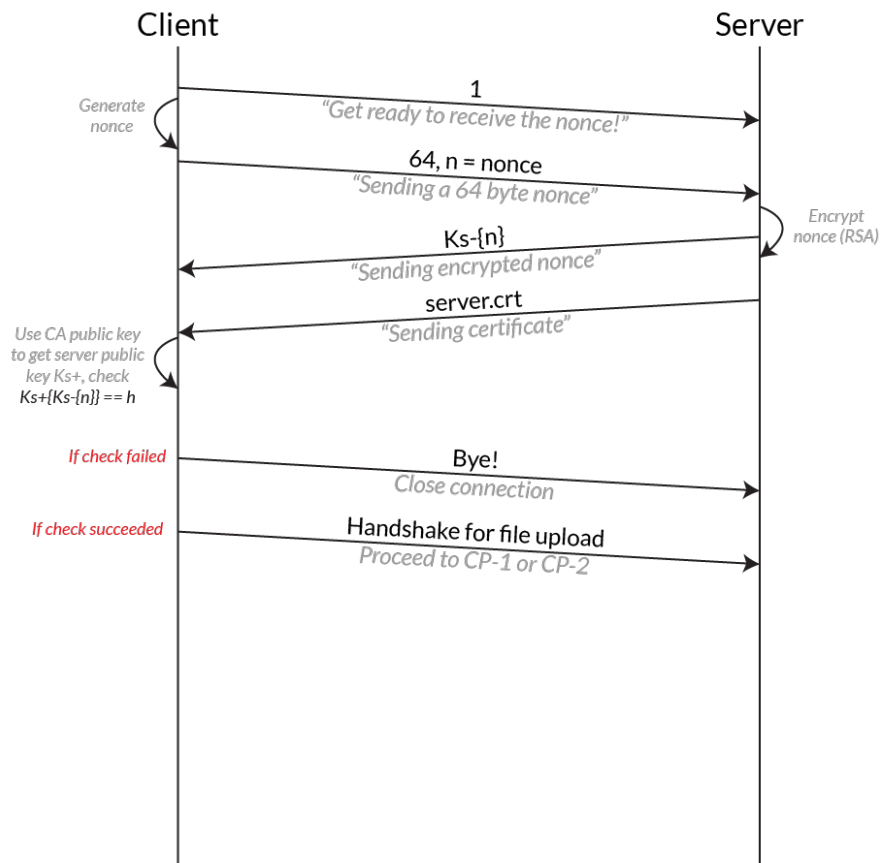


Figure 1: AP Protocol Specification

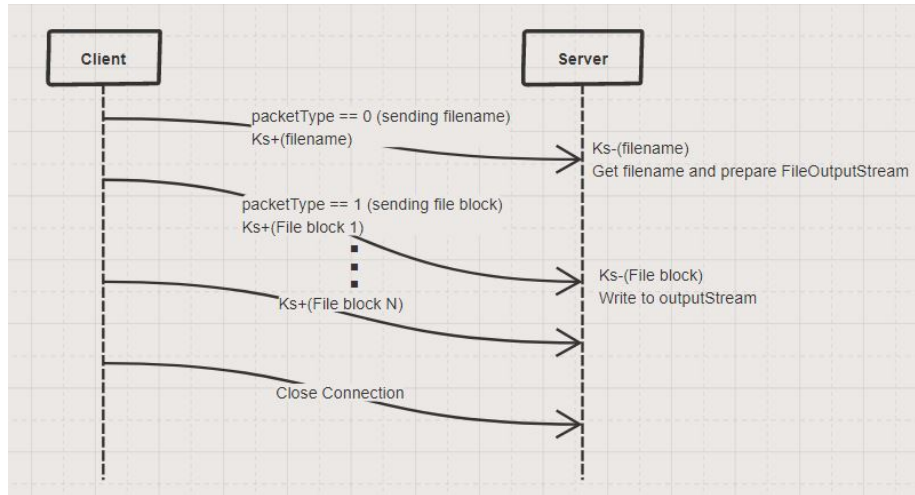


Figure 2: CP-1 Protocol Specification

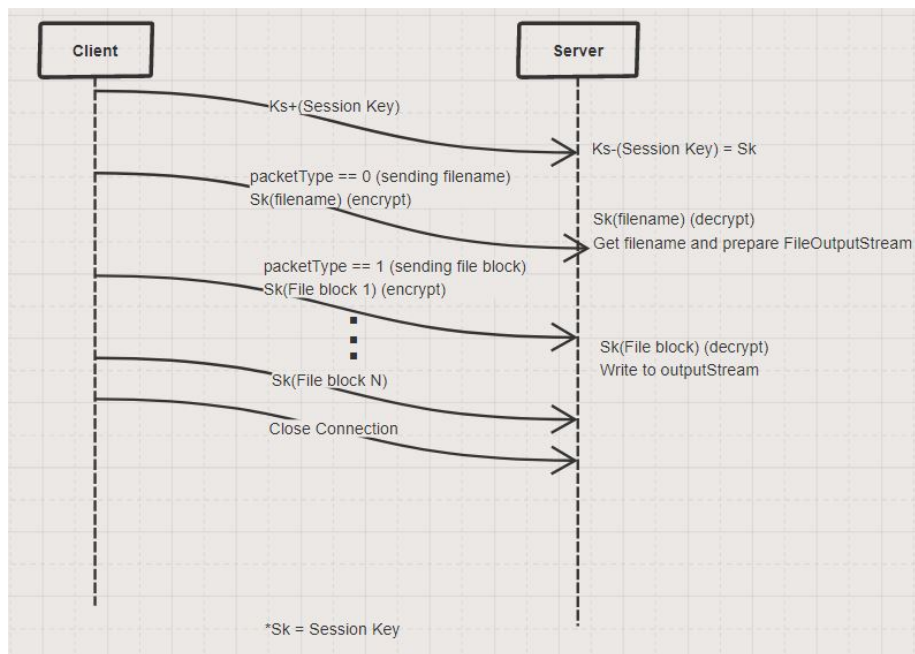


Figure 3: CP-2 Protocol Specification

Results

We transferred files of varying sizes. All experiments were done using a **single thread**. We observed the following trend:

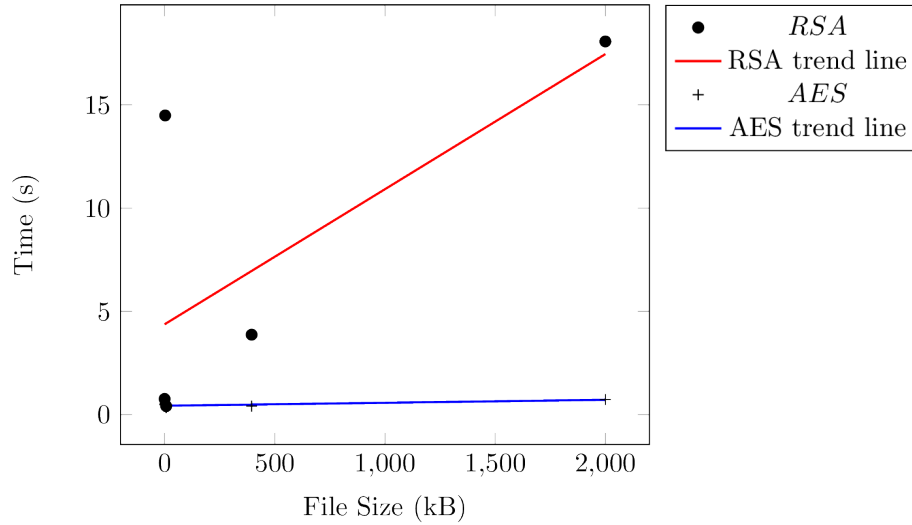


Figure 4: Plot

It is clear that the AES encryption standard is much quicker than using RSA for file transfer.

Conclusion

We use public key authentication to perform a handshake between the client and server for file transfer. Additionally, we allow the server and client to specify the encryption policy used (public key encryption, or symmetric key encryption with AES). We benchmark the speed of both policies and conclude that symmetric key encryption is significantly faster.

Additionally, our file transfer system allows the user to transfer files over multiple sockets in parallel. For large files, this speeds up file transfer significantly. However, for smaller files, the overhead in creating new threads may outweigh the performance gains from transferring files in parallel.