Programming Assignment 2

Group Members: Sean Gunawan (1004414), Claudia Chin (1004328)

Handshake Protocol

Issue with Initial Handshake

An issue we identified with the original handshake protocol was that an intruder could replay the client's messages such that the server would not know if the incoming message was from the client or the intruder.

This issue can be fixed by having the server send a nonce to the client, and the client sending a nonce encrypted with its private key and the accompanying public key to decrypt it, so that the server can tell that the client is “live”.

Improved Handshake

1. Client sends a handshake request by sending “CONNECT”.
2. Server sends fixed authentication message encrypted with server private key i.e. send encrypted AUTH_MESSAGE = “ACCESSING CSD SERVER - HANDSHAKE”.
3. Client acknowledges by sending “ACK-C1”.
4. Server sends certificate.
5. Client extracts server public key from certificate and decrypts encrypted fixed message.
6. If the decrypted message matches the fixed message, the client acknowledges by sending “ACK-C2”. Else, close connection.
7. Server sends plaintext nonce i.e. send 1 random integer as string.
8. Client receives nonce, encrypts it with client private key, and sends the encrypted nonce to the server.
9. Server acknowledges by sending “ACK-S1”.
10. Client sends its public key.
11. Server decrypts nonce with the client public key.
12. If the decrypted nonce matches the nonce sent by the server, the server acknowledges by sending “ACK-S2”. Else, close connection.
13. Conclude handshake.

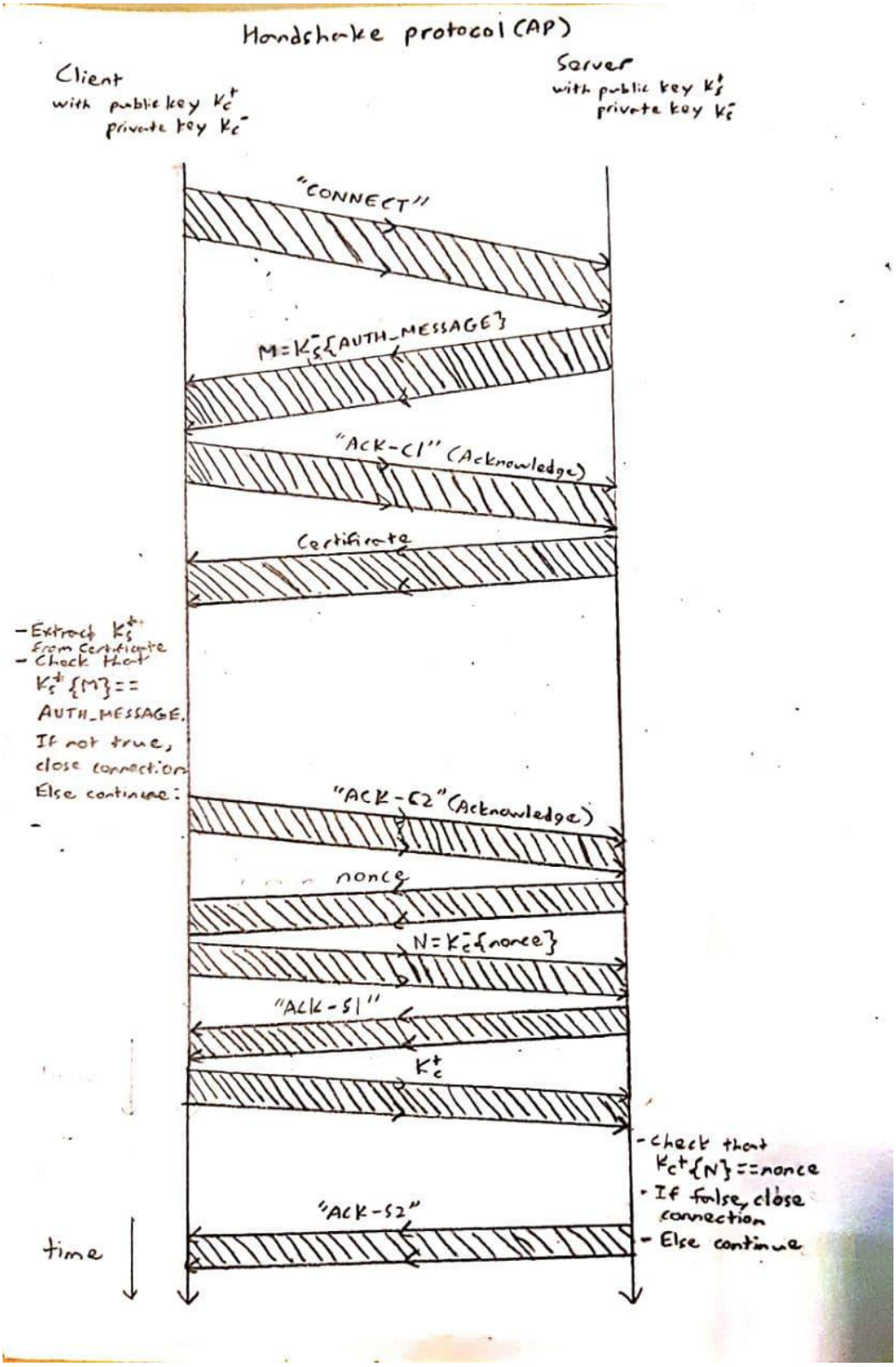

Fig 1: Handshake Protocol (original image here)

Negotiation Protocol

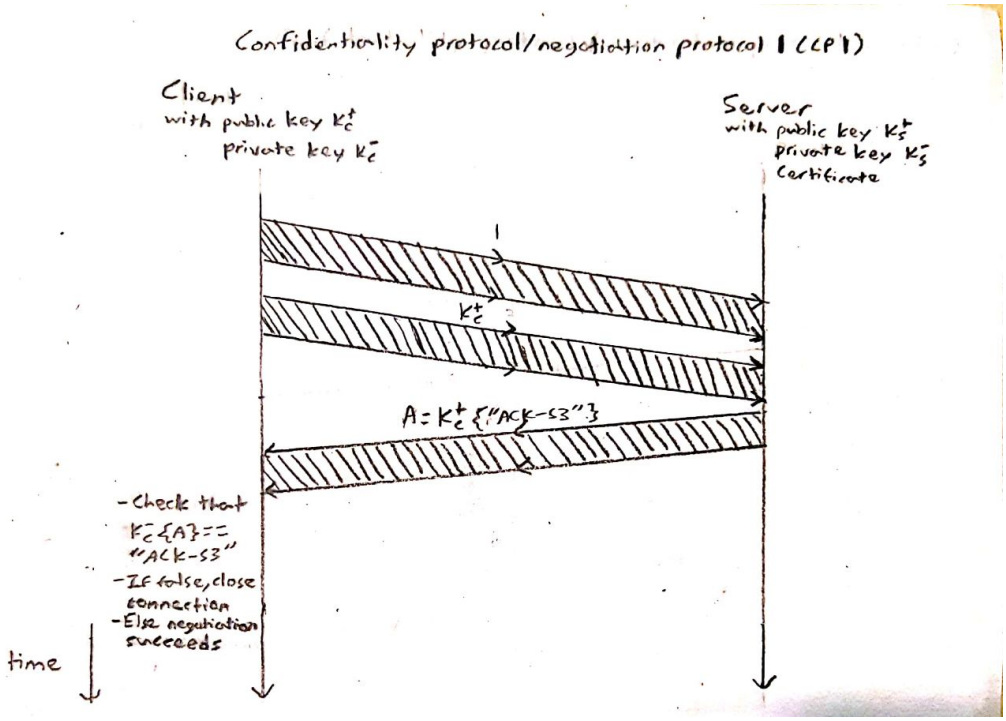

Fig 2: CP1 (original image here)

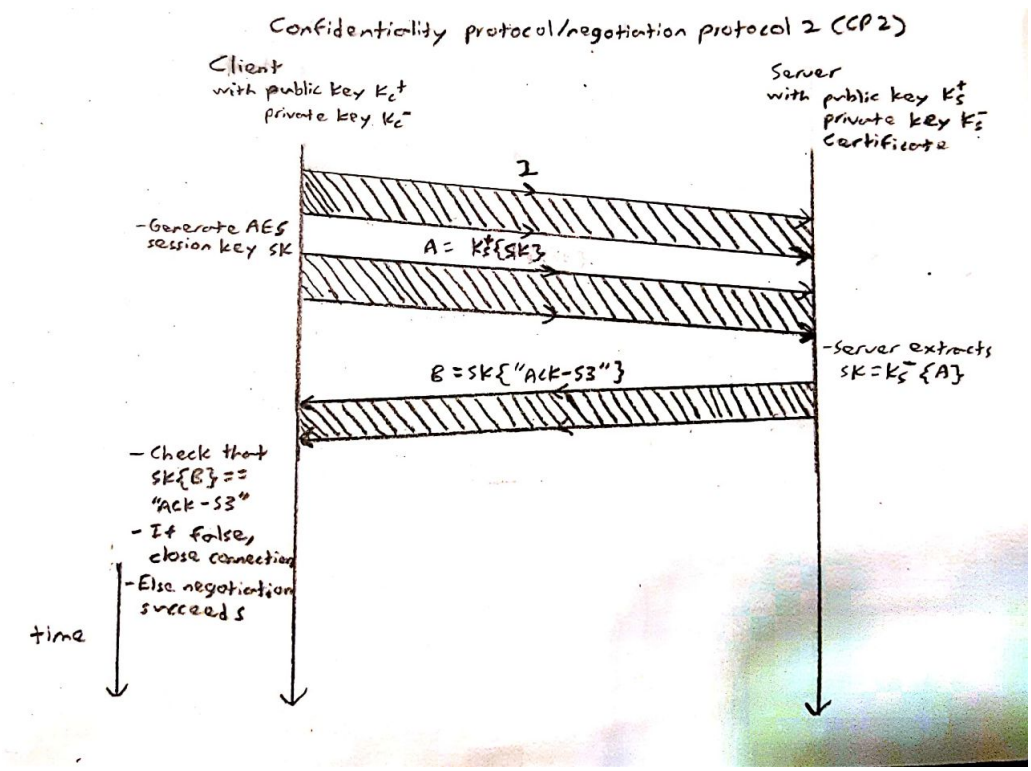

Fig 3: CP2 (original image here)

1. Client indicates which CP to use and sends the cp choice (integer 1 or 2) to the server.
2. If CP1, client sends its public key. Else, If CP2, client encrypts AES session key with server's public key, and sends it to server.
3. Server acknowledges the client by sending "ACK-S3" encrypted with the received key (the client public key for CP1 or AES session key for CP2).
4. If the decrypted message matches "ACK-S3", negotiation succeeds, and both sides can start the main operation. Otherwise close connection.

Results

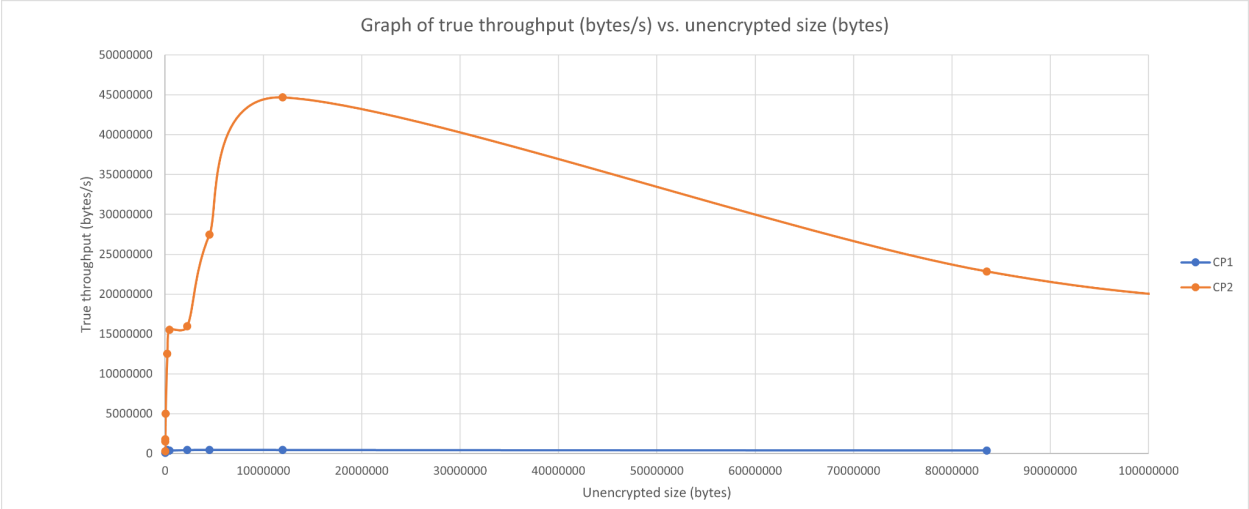

Fig 4: Graph of throughput in bytes/s vs. unencrypted file size in bytes (original image here)

Note that the programs were tested against all given test files, as well as 2 additional larger files for both confidentiality protocols (CP2 was subject to more tests with even larger file sizes as unlike the case with CP1, the tests with these files were still of acceptable duration.)

Appendix: Main Operation

Flow

1. Client sends command in the form of “command <arg0>”
2. Server responds with “[response code]\n<additional data>” where response code is:
    a. 0 if successful - server may provide additional data eg. for ls, this will be where ls output is displayed
    b. 1 otherwise - server usually provides reason
    c. 2 - bad request
3. If file transfer is involved, client/server goes on with file writing/reading immediately after response

Commands

Upload (put)	- To send “put [remote filename]” - On success, response is simply “0” - Will be rejected if filename already exists, or if there is no write permission for file
Disconnect (exit)	- To send “exit” - No response will be returned - just close connection immediately
Shutdown (shutdown)	- To send “shutdown” - Server will return “0\nServer shutting down…” (or similarly-formatted response)
List directory (ls)	- To send “ls” - On success, return “0\n[ls output]” - Will be rejected if server fails for some reason eg. no read permission for server’s current directory
Print working directory (pwd)	- To send “pwd” - On success, return “0\n[working directory]”
Change working directory (cwd)	- To send “cwd [target directory]” - On success, return “0” - Will be rejected if directory does not exist/can’t be accessed
Download (get)	- To send “get [remote filename]” - Will be rejected if file doesn’t exist or file can’t be read eg. due to server not having read permissions

Message Passing Contract

- Sender starts with sending 0.
- Sender writes byte array length n as integer followed by n bytes for each packet (of encrypted data).
- Sender ends by sending 0.

File Passing Contract

- Similar to Message Passing Contract but server starts and ends with sending 1.
- Additionally file readability/writability must be ensured first prior to initiating file passing - file passing should not be interrupted under normal circumstances, but if it is, file handles will be closed automatically

-- END --