1 Specifications

The Server program performs initialization as follows:

1. Generates a random 512-bit safe prime by first generating a 511-bit random prime q and then checking whether $N = 2q + 1$ is prime. If it isn't, generate a new prime q and try again until a valid N is found.
2. Finds a primitive root g of N.
3. Creates a socket and waits for the Client to connect to it.
    - **Addition: Please use IPV4 address 127.0.4.18 and port 31802 for your submission to the autograder.**

The Client program performs registration as follows:

1. The Client prompts the user for a username I and a password p.
    - **Note: To be sent over the socket, the Client should encode the username into bytes, and calculate the number of bytes in the username, $|I|$. The value $|I|$ should be encoded as a 4-byte number, big-endian formatted.**[1]
2. Generates a random 16-byte salt s and computes the value v as described in step 2 of the registration phase.
3. Connects to the Server
    - **Clarification: Upon connection, the Server will immediate send N and g to the Client. These should be sent as 64-byte numbers, formatted in big-endian.**
4. **Clarifications: After receiving server values, Client sends the tuple ('r', $|I|, I, s, v$) to the Server via a socket connection. 'r' is a one-byte UTF-8 character, while v is encoded as a 64-byte number in big-endian format.**
5. The Server stores I, s, and v for future use. It prints out a message indicating the process has successfully completed.
6. The Client closes the socket. It prints out a message indicating the process has successfully completed.

The Client program then initiates the protocol [2] as outlined above. Specifically:

[1] The standard encoding for sending numbers across networks is big-endian.

[2] Typically registration would be done only once, while secret sharing would happen on every execution of the client program. In the interests of making the assignment simpler, the Client program should not quit after registration, instead creating a second connection to negotiate a shared secret.

1. **The Client connects to the Server, which immediately sends the Client N and g as 64-byte numbers, big-endian formatted.**
2. **The Client sends the tuple ('p', $|I|, I, A$) to the Server via a socket connection. A is encoded as a 64-byte number in big-endian format.**
3. The Server replies with the tuple (s, B).
4. After finishing computation, the Client sends $M1$.
5. If the Server's computation indicates the negotiation was successful, it sends the Client $M2$, otherwise it closes the socket. It prints out a message indicating if a shared secret was successfully negotiated.
6. The Client closes the socket, if it has not been closed already. It prints out a message indicating if a shared secret was successfully negotiated.

1.1 Important Notes

Be sure to translate all numbers into 64-byte big-endian format, unless they denote the length of a string. The salt and cryptographic hashes can be sent as raw byte arrays, without any translation, while the username must have its length prepended to it as a 4-byte big-endian number. Do not add commas, brackets, or any characters beyond the contents of the variables in the tuple.

When printing outgoing network traffic to the screen, use the following format:

```
Client:  Sending len(username) <0000000b>
```

Switch "`Client`" with "`Server`" and "`len(username)`" with the variable being sent, as appropriate, and represent binary data in hexadecimal. When printing variables, use this format:

```
Server:  u = 866656320605032431945512885183598895395801498 67
```

Display integers as integers, binary data in raw hexadecimal as above,[3] and delimit strings with single quotes, " ' ". Bytearrays must only be printed in hexadecimal. Do not implement line wrapping.

If the registration process went successfully, the Client and Server print out this message:

```
Server:  Registration successful.
```

with `Server` changed as appropriate. To conclude the shared secret process, the Client and Server print out this message:

```
Client:  Negotiation successful.
```

with `successful` changed to `unsuccessful` as appropriate.

[3] Include the angle brackets, for consistency's sake.