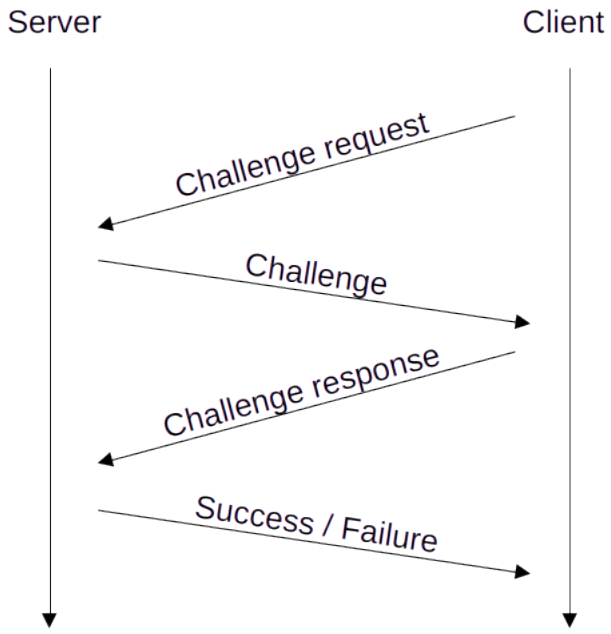
Many Internet client-server programs exchange information back and forth to complete a task. A protocol is used to enable communications. For this assignment you will create a network client that uses a challenge/response protocol.

Client Requirements

The client has one required command line argument, the hostname or IP address of the server. The client will send a challenge request, receive an encrypted challenge, send an encrypted challenge response, and the receive a status response. If the client violates the protocol or sends a malformed message the server will respond with an error.

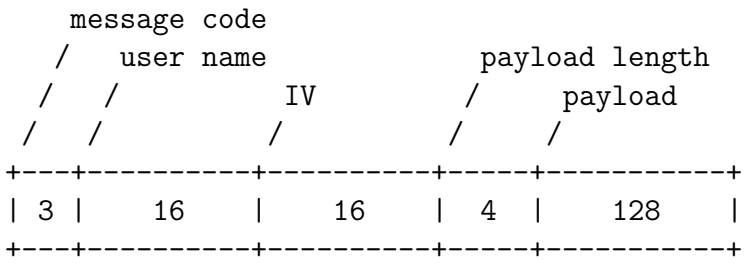


The message codes are:

- 105 - challenge request
- 110 - challenge
- 115 - challenge response
- 205 - success (challenge response is valid)
- 210 - failure (challenge response is invalid)
- 305 - error

The message format is a byte stream composed of 5 fields.

Description	Length	Data Type
Message code	3 bytes	char
User name	16 bytes	char
Initialization vector (IV)	16 bytes	unsigned char
Payload length	4 bytes	uint32_t
Payload (encrypted)	128 bytes	unsigned char



Only the user name field may be null terminated (max length is 16 bytes including the null byte). The full message must always be sent for the challenge and challenge response (all fields are required). The challenge request, success, and failure messages only require the message code and user name fields and the remaining fields shall be ignored by the receiver if sent. The error message only requires the message code field and the remaining fields shall be ignored by the receiver if sent.

The challenge payload[1] must be formatted as follows:

Description	Length	Data Type
Operation	1 byte	char
Left operand	4 bytes	uint32_t
Right operand	4 bytes	uint32_t

The challenge is sent in the payload field and the payload length field indicates how many bytes of the payload field are valid. Numeric values in the challenge message are the payload length and the two operands. All numeric values must be sent in network byte order. The unused portion of the payload field may be padded with arbitrary bytes.

[1]Size on the CS server using clang

Valid operations for the challenge are:

- Addition denoted as +
- Subtraction denoted as −
- Multiplication denoted as ∗
- Integer division denoted as /
- Remainder division denoted as %

The challenge response payload[1] must be formatted as follows:

Description	Length	Data Type
Challenge response	8 bytes	uint64_t

The challenge response is sent in the payload field and the payload length field indicates how many bytes of the payload field are valid. Numeric values in the challenge response message are the payload length and the response value. The unused portion of the payload field may be padded with arbitrary bytes.

The payload field is always sent encrypted. It is encrypted/decrypted using AES with a 128 bit key. A *C* header file and object file to encrypt and decrypt the message payload is available on the class web page (aes_crypto.zip). The header file contains information for how to use the encrypt/decrypt function in the object file. The client **must** use the provided header and object file to encrypt/decrypt the message payload.

The provided encryption function must be called with the actual size of the payload to be encrypted/decrypted (input length parameter). The payload length argument cannot be larger than 96 bytes in order to leave room for expansion during encryption. This, of course, means the input to be encrypted/decrypted cannot be larger than 96 bytes.

The client must use TCP and communicate on port 11,000. Your makefile must link the aes_encrypt.o file and include the linker option for the cryptography library.

The client should produce output for each message sent and received. The output must clearly distinguish between success and failure.

Example output for success:

```
send request
receive challenge
send challenge response
receive ID SUCCESS
```

Example output for failure:

```
send request
receive challenge
send challenge response
receive ID FAILURE
```

The client must be well designed and handle errors gracefully. Well designed means that your code adheres to best practices for efficiency, clarity, and software security. Graceful means your programs must recover from the error and continue if possible, otherwise they must display an informative message, cleanup resources, and terminate.

Your client must be developed in *C* using the BSD socket API on the CS server. The client must compile and run on the CS server with cc (clang).

★ Error handling hint: Follow best practices when exiting a function that needs to release resources on error. See this web page for details `https://wiki.sei.cmu.edu/confluence/display/c/MEM12-C.+Consider+using+a+goto+chain+when+leaving+a+function+on+error+when+using+and+releasing+resources` (search the web for "SEI CERT C goto chain").

Design sketch:

Create one design sketch (markdown file) for the client. Your design sketch must have the following sections:

- Title: (Challenge Client)
- Program requirements: This section describes what the program must do to be considered a success
- Program inputs: This section describes the data that is input to the program. Be specific about the type of data and acceptable values (range).
- Program outputs: This section describes the information produced by the program. Be specific about the type of data and how it will be presented (formatting and interpretation).

- Test plan: This section describes how you will test your solution to determine if it is correct. For each test case show the test input and expected output.
- Design overview: This section describes your solution in a human language (English, flowchart, pseudocode). It should read as a step-by-step description of your solution. Each step must be detailed, concise, and unambiguous. This narrative must relate your design choices to the program requirements.

Due dates:

Wednesday 23rd before 11:59 p.m.
Design, Makefile, and skeleton code (client) commit, tag it "design"

Wednesday 30th before 11:59 p.m.
Completed client commit, tag it "client"

Use an annotated tag *after* your commit to tag the commit.

How to submit:

Create an empty git(1) repository in a folder named "program-7". Commit and tag your files as described above in "Due dates". You may, and should, commit more frequently than the required commit dates. You should build your client program in stages (skeleton, startup, etc.) and commit the working version of each stage as you complete it.

Program evaluation:

Your program must conform to the **UNA CS Code Style Guide** linked on the class web page and documented in style(9). Your program must compile without any compiler warnings or errors. Code will be evaluated for correctness, efficiency, and style.

Helpful resources:

```
Command line arguments: https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/
--
man pages - memcpy(3), memcmp(3), malloc(3), free(3)
man pages - htonl(3), htobe64(3)
man pages - style(9), printf(3), egdb(1)
```

Additional resources:

```
https://beej.us/guide/bgnet/
https://csrc.nist.gov/Projects/ssdf
https://sourceware.org/gdb/current/onlinedocs/gdb.html
```