CSE351 Computer Networks

Fall, 2025, Programming Assignment #1
Due: September 21, 2025 (Sun), 23:59 KST

Instructor: Taesik Gong (taesik.gong@unist.ac.kr), TA: Changmin Lee (lchm1106@unist.ac.kr)

==========

Submission Instructions. You shall submit this assignment as a single ZIP file named PA1_StudentNumber_Name.zip (e.g., PA1_20251234_HongGildong.zip) on the Blackboard. The ZIP must include client.c, server.c, a Makefile (running $ make all must build the executables client and server), and report.pdf. You may include additional .c/.h files if needed, but they must be built by the Makefile and briefly explained in the report. No skeleton files are provided—you must implement everything from scratch.

Collaboration Policy. You are welcome to discuss the homework with your classmates to understand the concepts, and you may also use code snippets from the provided guide links. However, if your report or code is found to be unreasonably identical to someone else's work, it will be considered plagiarism and will result in no points for all involved.

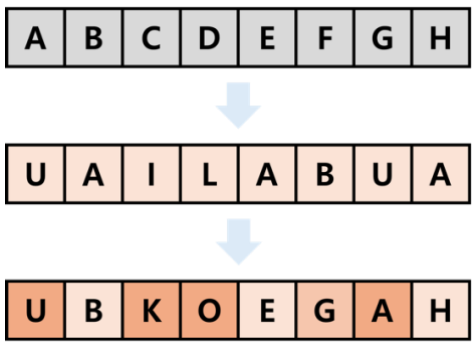
Project Overview. In this project, you will build a small client–server application that supports text encryption. The goal is to get hands-on experience with socket APIs; by the end, you should be comfortable implementing connection-oriented (TCP) programs that exchange data over sockets.

What to Do. Implement both a client and a server that communicate over TCP. The client reads a string from standard input and sends it to the server. The server applies the Vigenère cipher to the string (encrypt or decrypt, as requested), returns the processed result, and continues running to handle additional requests. The client prints the server's response and then exits.

==========

[Description 1] Vigenère cipher

Vigenère cipher is a classical polyalphabetic substitution cipher that runs on strings. The algorithm is straightforward; it uses a repeating key so that each letter in the string is shifted by the amount determined by the corresponding key letter (mapping a→0, b→1, …, z→25). For example, with the key "uailab", the six-letter string "abcdef" becomes "ubkoeg" (letters are treated case-insensitively, non-letters unchanged). (For more information on Vigenère cipher, refer to https://cryptii.com/pipes/vigenere-cipher.)

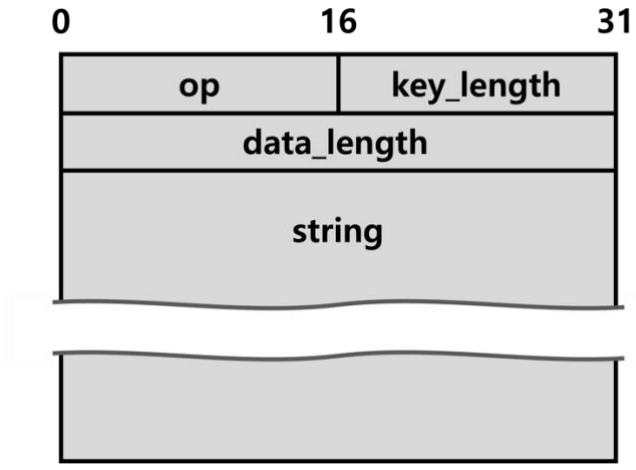


In this project, the given input may include both uppercase and lowercase alphabets. It may also include non-alphabetical letters, such as numbers or special characters. Your program should convert all the uppercase letters to lowercase, and perform Vigenère cipher only on alphabets, ignoring other special characters or numbers.

[Description 2] Application layer protocol specification

When your client and server communicate, we need a protocol so that we can send extra data in addition to the

plaintext/ciphertext string. Here, the extra data includes things like, “Do we encrypt or decrypt the string?” and “What keyword should we use for the Vigenère cipher?”



The figure above shows our message structure. The numbers above each box indicate **bits** (not bytes).

(a) ***‘op’ field:*** This field takes the first 16 bits. It denotes the operation type of the request. If this field is 0, it means encryption. If this field is 1, it means decryption.

(b) ***‘key_length’ field:*** This field takes the next 16 bits. It denotes the length (in bytes) of the keyword. Also, this field should be in network order. (For network/host order, refer to https://en.wikipedia.org/wiki/Endianness and https://stackoverflow.com/questions/15859649/why-is-data-sent-across-the-network-converted-to-network-byte-order)

(c) ***‘data_length’ field:*** This field takes the next 32 bits. The value of this field denotes the length (in bytes) of the payload(data). Also, this field should be in network order.

(d) ***‘string’ field:*** The size of this field is variable and is determined by the values above. It is the concatenation of two parts placed back-to-back:

string = key_string (key_length bytes) || data_string (data_length bytes)

ex) [‘u’ ‘a’ ‘i’ ‘l’ ‘a’ ‘b’][‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘ ’ ‘w’ ‘o’ ‘r’ ‘l’ ‘d’]

key_string (6B) data_string (11B)

Values on ‘op’, ‘key_length’, and ‘data_length’ fields are treated as unsigned values. There’s one more restriction in the protocol, that the maximum length of the message (which includes header, key, data) is limited to 10 MB (= 10,000,000 bytes). See https://www.quora.com/What-is-the-difference-between-a-megabyte-and-a-mebibyte).

[Problem 1] Client implementation (30 points)

(a) Your client program should be able to take an IP address, a port number of a server and operation type and shift through the command line parameters. Example usage of the client program should look like this:

```
$ ./client -h 172.18.0.2 -p 1234 -o 0 -k uailab
```

In this case, the server’s IP address is `172.18.0.2`, the server’s port number is `1234`, and we are encrypting with `k = uailab`. You must follow this command line parameters format and binary name (which is ‘client’).

(b) The client takes the string by standard input (stdin). The client then creates the message and sends it to the server. Each data should be wrapped in the correct message format that follows the protocol above. Once the client receives a reply from the server, it should unwrap the message and only print the

resulting string to the standard output (stdout). Note that if the input is too long (more than 10 MB), then the client should split the input and wrap it in different messages. Otherwise, the server will reject the request.

(c) Your client program should be able to handle any kind of string, including binary data.

(d) Your client program should only terminate when EOF (end-of-file) is received in stdin.

[Problem 2] Server implementation (50 points)

You should also implement a server program that can communicate with the client program you have built above.

(a) Your server should receive the required information through the command line parameters. An example usage of the server program should look like this:

```
$ ./server -p 1234
```

Unlike the client, when you run your server, you should be able to set the server's port number manually. In fact, it is very natural that by setting the server's port number with the desired value we can advertise the port number to the clients out there. You must follow this command line parameters format and binary name (which is 'server').

(b) The server program listens to incoming connection requests and then accepts them. Once the server program receives a string from the client, the server performs the Vigenère cipher on this string and sends the resulting message back to the client.

(c) Your server program must be able to handle multiple connections in parallel (up to 50). You have several options for this requirement, like fork() or select()/epoll().

(d) Your server should reject connections from clients that violate the protocol.

[Appendix A] Extra specification

(a) ***Language & Libraries:*** Write your program in C. No other languages are allowed. Only use C standard libraries and Linux system calls. No other 3rd party libraries are allowed.

(b) ***Byte Order:*** All multi-byte integers on the wire must be in network byte order (big-endian). If unfamiliar with endianness, see: https://en.wikipedia.org/wiki/Endianness

(c) ***Build (Makefile)*** *(15 points):* Provide a Makefile. Automated scripts will run $ make all; this must produce two executables: client and server.

(d) ***Report:*** Submit a brief implementation report (≤ 3 pages, PDF). Cite any external sources used.

[Appendix B] Guide & Tips

(a) Go to this guide (https://beej.us/guide/bgnet/html/multi/index.html) and read what kind of APIs are out there. There are many example codes as well. You are allowed to use code segments and socket API usages in the guide above, but make sure you cite it in the report.

(b) Please be aware that a message in our protocol is not served by a single network packet. As you implement an application-layer protocol on top of TCP (like HTTP), your message can be split into multiple packets no matter how small your message is.

(c) Your client program should print nothing but encryption/decryption results. Please avoid printing '\n' at the end of enc/dec results to make it look better on the Terminal. Use stderr if you need to print something else for debugging purposes.

[Appendix C] Environmental setting

We provide you a few sample input/output files for your program. However, when grading, we will use a different set of inputs (which are typically longer and more complicated).

(a) Sample encrypt/decrypt files use `k = uailab`

In addition, we provide a Dockerfile. All tests are expected to be executed inside a container built from this Dockerfile. Please follow the accompanying docker_setting_guide.pdf for setup and run instructions. Grading will be performed in the same containerized environment to ensure consistency and reproducibility.

Note that any problem that is caused by using different environments is the student's responsibility. We will not accept any regrade request such as “It runs on my computer”.

[Appendix D] Grading criteria

The grading will be done as below.

(a) (30%) Code: Client implementation

(b) (50%) Code: Server implementation

(c) (15%) Code: Makefile

(d) (5%) Submission format (follow ***Submission instructions*** section)

(e) (0%) Report: The report will not be explicitly graded to the numerical value, but we will use it when there are any issues with grading.

(f) Any violations (wrong Makefile script, wrong file name, report is not PDF, etc.) will result in a penalty.

(g) If your code has formatting problems (such as adding unnecessary \n or other whitespaces to make output pretty), we will deduct 20% from your grade that you will get when the formatting problem is solved. As mentioned above, don’t add anything but just simply print out the encryption/decryption result.

(h) We will run your submission 10 times and will give you the highest score.

(i) We will set a 30 seconds limit per test case first, and re-run those with a 3 minutes limit that failed due to the timeout on the first trial. (There will be no penalty for failing a 30 seconds test but passing a 3 minutes test.) If your code fails to run in 3 minutes, you will get no credits for that test.