Practice #2:
TCP Socket Communication

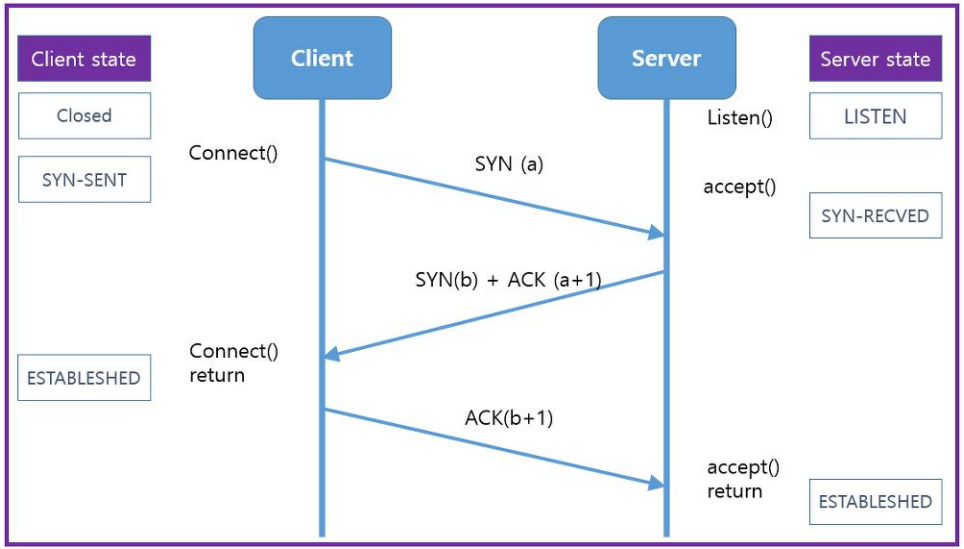
Kisoo Kim (k014520@kaist.ac.kr), Boyan Kostadinov (boyanyk@kaist.ac.kr)
2020/09/24

Goal

- Review **basic concepts and API of network socket**
- Implement **a connection-oriented, client-server protocol** based on given specification
- learn a basic encryption scheme (XOR Cipher)

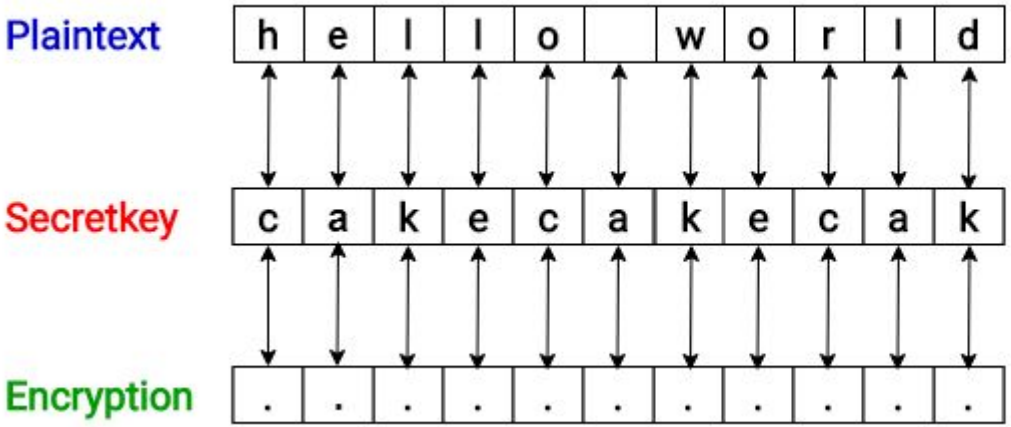
Background: TCP Handshake

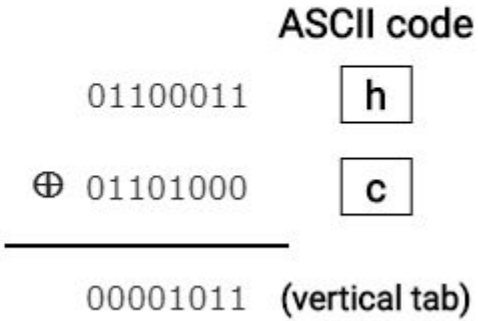
- an automated process of **negotiation between two participants** through the **exchange of information** that establishes the **protocols of a communication link** at the start of the communication [wikipedia]
- Example of three-way handshake



Background: XOR Cipher

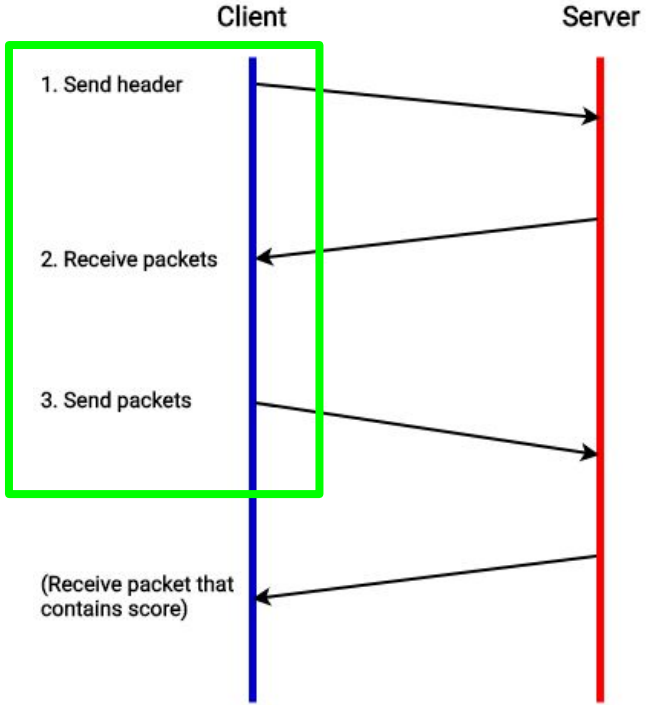
- https://en.wikipedia.org/wiki/XOR_cipher
  - **We use multi-char XOR**
- Encryption and Decryption algorithms are the same





Task 1: Basic string decryption service

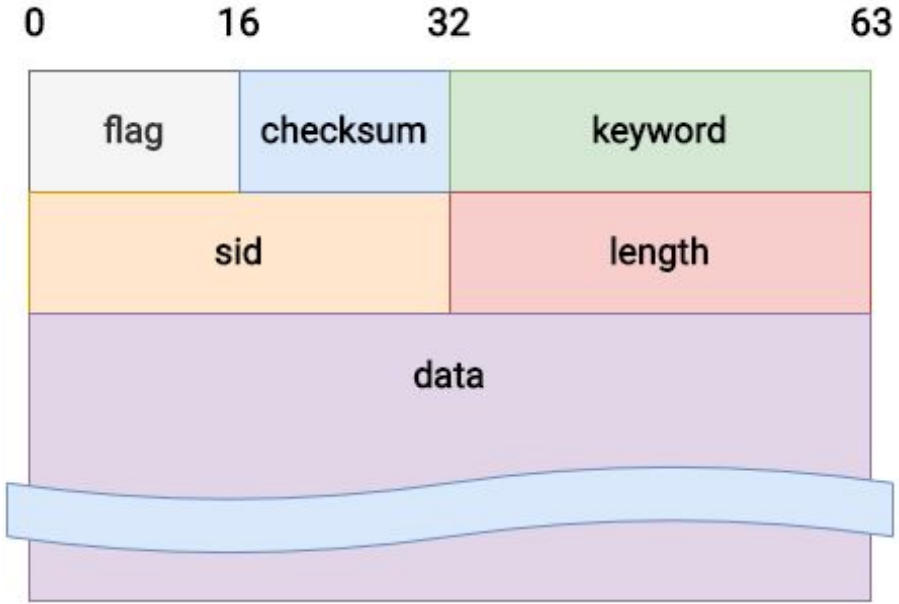
- **Implement “a client”** which communicates with the server
  - based on the **specification**
  - in *Python 3.6+*
- All protocols are implemented over **TCP sockets**
- Server to test against and submit is provided
  - Server scores *three main tasks*
  - If you send initial header with the key “**sbmt**”, server stores your score on database
  - Server sends ***packets containing error message*** if your implementation has a problem



Task 1: Protocol Specification

- Protocol indicates the length of string by using a separate field
- Byte order follows **network order**

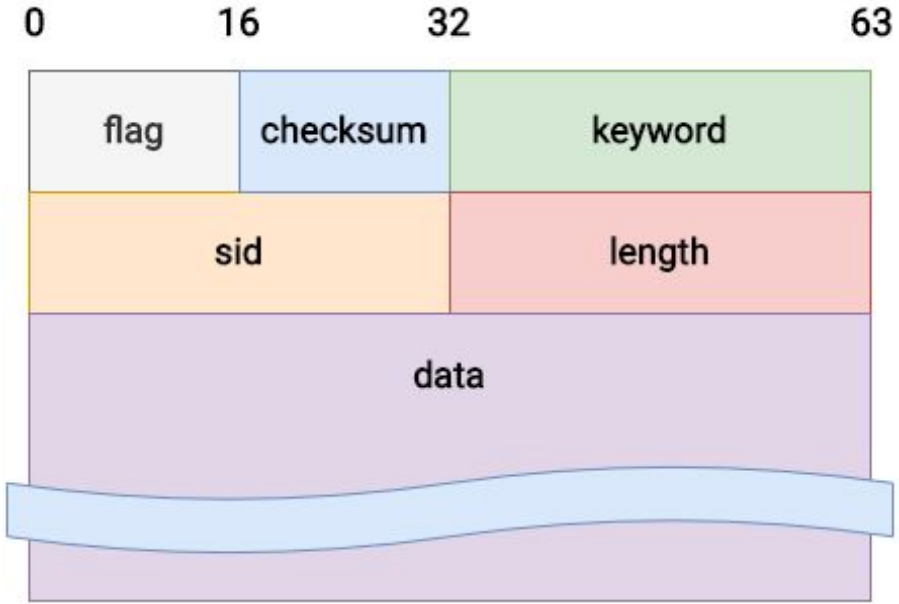
- **flag** field (16 bits)
  - whether the packet is **last one or not**
  - 0 (next one will come), 1(last packet)
    - if you send only one packet, then 1
- **checksum** field (16 bits)
  - used for **error-checking of protocol fields**
  - calculated in the same way as **TCP checksum**
- **keyword** field (32 bits, 4 characters)
  - all characters are **lowercase alphabet**



Task 1: Protocol Specification

- Protocol indicates the length of string by using a separate field
- Byte order follows **network order**

- **sid** field (32 bits)
  - your **8 digit** student id
  - (also it is used for score)
- **length** field (32 bits, in bytes)
  - total length of a message
  - flag, checksum, keyword, sid, length, data
  - Maximum length **10KB** (10,000 Bytes)
- **data** field
  - String to be transmitted



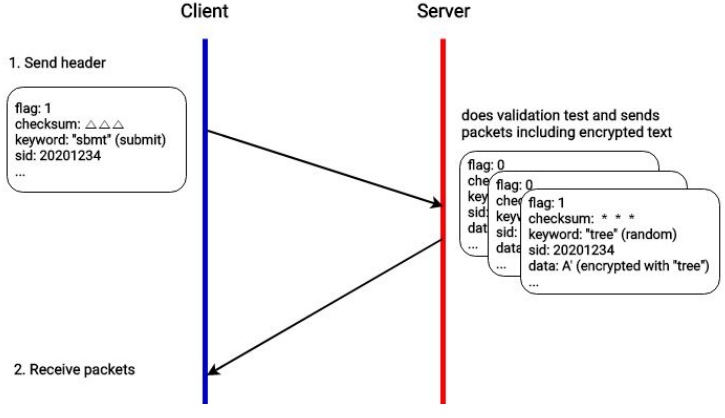
Task 1: What client should do

1. **Send header (10 pt.)**

   a. Initial keyword could be **any 4 letter word (lowercase) for testing** (otherwise, ‘sbmt’)
   b. Server validates header and gives 10 pt if it is made and sent correctly

2. **Receive packets (10 pt.)**

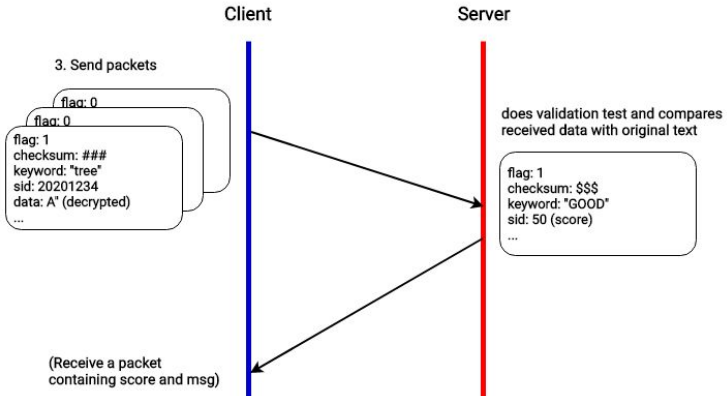
   a. *Server assigns you a **random word** as a key and sends **packets** containing encrypted data*
   b. If client receives all packets, then server gives 10 pt



Task 1: What client should do

3. Send packets (15 pt. + 15 pt.)

a. Client **decrypts data with given key (XOR Cipher)** and sends packets
b. Server gives 15 pt if the **length** of received data and original data are same
c. If given data is **correctly decrypted** (15 pt), server sends a packet
d. otherwise, server sends packet with error message
    i. If you use ‘sbmt’ key and your implementation is wrong, then **partial score will be stored on database** (if you use <u>your student id properly</u>, otherwise registration fail)



Task 1: Client - Submission Requirements

- Format
  - **You MUST follow this argument format**
  - `python client.py --host=143.248.56.39 --port=4000 --studentID=20200000`
    - Server is running (**IP: 143.248.56.39, Port: 4000**)
  - If you use struct library, format string is '!HH4sII'
- Keep the maximum length of packet **10KB (10,000 Bytes)**
- Use of external libraries is not allowed

`ONLY THE BEST SCORE IS SAVED!`

Task 1: Client - Example Screenshots

- Command line with **key, score, and message** of result packet (or error packet)

```
❯ python3 client.py --host=143.248.56.39 --port=4000 --studentID=20149322
KEY: GOOD
Score: 50
Message: DECRYPTED DATA MATCH & YOUR POINTS ARE REGISTERED ON DATABASE
```

Task 1: Recommended Links

- python **struct** library (Interpret bytes as packed binary data)
  - https://docs.python.org/3.6/library/struct.html
- python **socket** library
  - https://docs.python.org/3.6/library/socket.html

Task 2: Basic Server Using Sockets

- Implement server that listens to a socket port and can handle multiple concurrent clients
- Server generates a UUID for each client
  - Use the python `uuid.uuid3` method from the built-in uuid library
- A client to test against will be provided in the form of a binary file
  - There is a binary for Win-x64, Mac-x64 and Linux-x64
  - Make sure you download the correct one for your platform!
- Client will send multiple simultaneous requests to server
  - There is no order to the requests - they have to be handled regardless of the sequence the requests come in
- If the server processes all clients' requests successfully, you get full points
- You need to be on the campus network or on the KAIST VPN

Task 2: Request types

`“FIBONACCI” / “FACTORIAL” / “WORD_COUNT / “COMPLETE”`

- FIBONACCI
  - Client sends a message with body “FIBONACCI”
  - After a short pause, it sends a number *n*
  - Your server has to calculate the *n-th* Fibonacci number and return the result to the client
- FACTORIAL
  - Client sends a message with body “Factorial”
  - After a short pause, it sends a number *n*
  - Your server has to calculate the Factorial of *n* and return the result to the client
- WORD_COUNT
  - Client sends a message with body “WORD_COUNT”
  - After a short pause, it sends a text file to your server
  - Your server has to receive the complete text file and count how many words there are inside, returning the count result to the client (hint: count whitespaces)
- COMPLETE - Client is done and you can close the socket

Task 2: Sample Responses

- Let’s consider a client with UUID of `123e4567-e89b-12d3-a456-426614174000`
- Each response must be of the form `answer_uuid`. For example:
- Fibonacci:
  - If request is Fib of 7, answer is 8:
  - Response message: `8_123e4567-e89b-12d3-a456-426614174000`
- Factorial
  - If request is Fact of 6, answer is 720
  - Response message: `720_123e4567-e89b-12d3-a456-426614174000`
- Word Count
  - If sent text has 700 words
  - Response message: `700_123e4567-e89b-12d3-a456-426614174000`

Task 2: Server

- Implement server that listens to a socket port and can handle multiple concurrent clients
  - Server returns UUID to each client - they have to be different
  - Server has to handle following requests in an arbitrary order:
    - “FIBONACCI”, “FACTORIAL”, “WORD_COUNT”
- When ready to submit, use the ***--submit==True*** argument! Otherwise you will NOT get any points for implementation even if you submit on KLMS
- Running client - all arguments (port/studentID/submit) are required

```
./client --port=1234 --studentID=20200000 --submit=False
```

```
./client --port=1234 --studentID=20200000 --submit=True
```

ONLY THE BEST SCORE IS SAVED!

Task 2: Example procedure

- You run your server with: `python server.py --port=1234`
- Then, you run client with:

```
./client --port=1234 --studentID=20200000 --submit=False
```

- Client creates 4 Threads and connections to your server
- Each thread generates sequence of 6 requests - 2 of each type, e.g:

```
{"FIBONACCI", "WORD_COUNT", "FACTORIAL", "FACTORIAL", "FIBONACCI", "WORD_COUNT"}
```

- Your server has to handle all incoming requests to receive full points
- Note that you will have to turn off the server once all the client requests have been complete - you can handle this in your code or manually interrupt the process

Task 2: Example screenshots

```
Client name is: Client3
Client name is: Client1
Client name is: Client4
Client name is: Client2
None
None
None
None
FACTORIAL
FIBONACCI
FILE
FILE
A B C D E F G H I J
[127.0.0.1:37808] File request from client, words in file are 10
A B C D E F G H I J
[127.0.0.1:37806] File request from client, words in file are 10
FIBONACCI
FIBONACCI
[127.0.0.1:37802] Factorial request from client, fact of 1
[127.0.0.1:37804] Fibonacci request from client, fib of 1
FACTORIAL
FILE
A B C D E F G H I J
[127.0.0.1:37802] File request from client, words in file are 10
FILE
[127.0.0.1:37808] Fibonacci request from client, fib of 4
[127.0.0.1:37806] Fibonacci request from client, fib of 4
FACTORIAL
FIBONACCI
[127.0.0.1:37804] Factorial request from client, fact of 3
FACTORIAL
A B C D E F G H I J
[127.0.0.1:37802] File request from client, words in file are 10
FACTORIAL
[127.0.0.1:37806] Factorial request from client, fact of 3
[127.0.0.1:37808] Fibonacci request from client, fib of 5
```

Sample Server Output

```
[Client1] Socket connected to 127.0.1.1:3333
[Client3] Socket connected to 127.0.1.1:3333
[Client2] Socket connected to 127.0.1.1:3333
[Client4] Socket connected to 127.0.1.1:3333
[Client3] UUID: 5d6dc07b61f93ad9b00e829ee52e6b92
[Client1] UUID: f97e575c28d536b49170f3cbaef757af
[Client4] UUID: 64b4bd84dd76328f93d50f0c2a1c8684
[Client2] UUID: c5fdeac9dbd9322ca157a09ac7052824
[Client4] WORD_COUNT Okay
[Client2] WORD_COUNT Okay
[Client1] FIBONACCI Okay
[Client3] FACTORIAL Okay
[Client3] WORD_COUNT Okay
[Client2] FIBONACCI Okay
[Client4] FIBONACCI Okay
[Client1] FACTORIAL Okay
[Client3] WORD_COUNT Okay
[Client2] FACTORIAL Okay
[Client4] FIBONACCI Okay
[Client1] FACTORIAL Okay
[Client3] FACTORIAL Okay
[Client1] WORD_COUNT Okay
[Client2] FIBONACCI Okay
[Client4] FACTORIAL Okay
[Client1] WORD_COUNT Okay
[Client3] FIBONACCI Okay
[Client4] WORD_COUNT Okay
[Client2] FACTORIAL Okay
[Client1] FIBONACCI Okay
[Client3] FIBONACCI Okay
[Client4] FACTORIAL Okay
[Client2] WORD_COUNT Okay
Fib 8, Fact 8, Word 8
------------
Total points 40/40
------------
```

Client Output

Task 2: Server - Submission Requirements

- No external libraries!
- Python 3.6+
- When ready to submit, use the ***--submit==TRUE*** argument! Otherwise you will NOT get any points for implementation
- Server has to be named server.py and take a port as an argument. See example:

```
python server.py --port=1234
```

- Report: Describe how you handle the different requests and include execution screenshots
- Scoring - 40 points in total for code; 4 points for report. Details on last slide
  - Single client successful handling - 20 points
  - Multiple clients successful handling - 20 points
  - Report - 4 points

We will evaluate your code on Python 3.6+,

Deliverables

Due: 10/7, 11:59 pm

Submit a zip file to KLMS containing:

- client.py
- server.py
- report.pdf

For both tasks you need to also use the submit function in server/client.

plagiarism, late submission: **0 point**

- Task 1:

```
python client.py --host=143.248.56.39 --port=4000 --studentID=20200000
(with 'sbmt' initial key: Slide 5)
```

- Task 2 - You need to be on the campus network or on the KAIST VPN

```
./client --port=1234 --studentID=20200000 --submit=True
```

- **If you only submit to KLMS you don't get points for your implementation**

Scoring (100 pt.)

- client.py (50 pt.)
  - Sending Headers (10 pt.), Receiving Packets (10 pt.)
  - Sending Packets (15 pt.), XOR Decryption (15 pt.)
- server.py (40 pt.)
  - Handling 1 Fibonacci (5 pt.) / 4 Fibonacci (10 pt.)
  - Handling 1 Factorial (5 pt.) / 4 Factorial (10 pt.)
  - Handling 1 Word_Count (10 pt.) / 4 Word_Count (20 pt.)
  - Should be tested with the provided client file
- Report in .pdf (10 pt.) - maximum 3 pages.
  - Screenshots of the results from executing server.py and client.py .
  - Explain your code implementation.

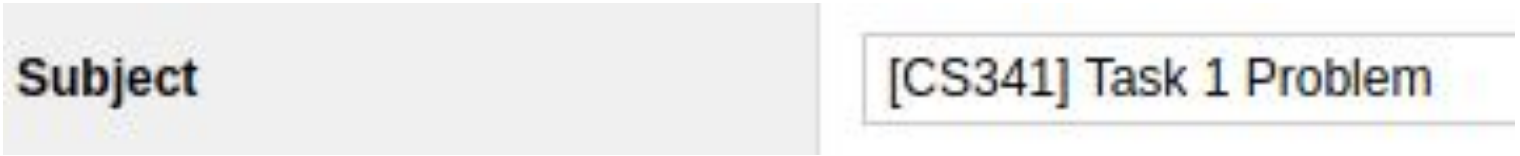
Questions

If you have any questions regarding the tasks, ask on KLMS Board first

If you're unsure about the question, you can also contact us directly:

- Task 1 - Contact Kisoo Kim (k014520@kaist.ac.kr)
- Task 2 - Contact Boyan Kostadinov (boyanyk@kaist.ac.kr)

Put [CS341] in the subject, for example if you have a problem with Task 1:



Email may otherwise not be received!