

**ELEC-H417 - Communication networks : protocols and architectures**

---


# Communication networks project Tor Network

---

De Clercq Tom  
El Qaisy Driss  
Latte Samuel  
Abbas Ismaïl

Dricot Jean-Michel

2022-2023



## Contents

|          |                                    |          |
|----------|------------------------------------|----------|
| <b>1</b> | <b>Introduction</b>                | <b>1</b> |
| 1.1      | Introduction . . . . .             | 1        |
| <b>2</b> | <b>Architecture</b>                | <b>2</b> |
| 2.1      | Tor network architecture . . . . . | 2        |
| 2.1.1    | Server . . . . .                   | 3        |
| 2.1.2    | Relays . . . . .                   | 3        |
| 2.1.3    | Client . . . . .                   | 3        |
| 2.1.4    | Custom Tor Network . . . . .       | 4        |
| 2.2      | Challenge Response . . . . .       | 5        |
| <b>3</b> | <b>Innovation and creativity</b>   | <b>6</b> |
| 3.1      | Multiple terminals . . . . .       | 6        |
| 3.2      | Challenge Response : key . . . . . | 6        |
| 3.3      | Http links opener . . . . .        | 7        |
| 3.4      | Error message management . . . . . | 7        |
| <b>4</b> | <b>Challenges</b>                  | <b>8</b> |
| 4.1      | Libraries . . . . .                | 8        |
| 4.2      | Socket management . . . . .        | 8        |

## 1.1 Introduction

Protecting your data while surfing the internet has become an important issue in a world where everything seems to be more and more connected. For this purpose, as part of the ELEC-H417 course, it has been expected to create a TOR network in order to simulate a more secure access between a client and a server. In general, a Tor network is a group of volunteer-operated servers that allows people to improve their privacy and security on the Internet. It is done by routing internet traffic through a network of servers, each of which removes a layer of encryption.

### The Onion Router (TOR) Network

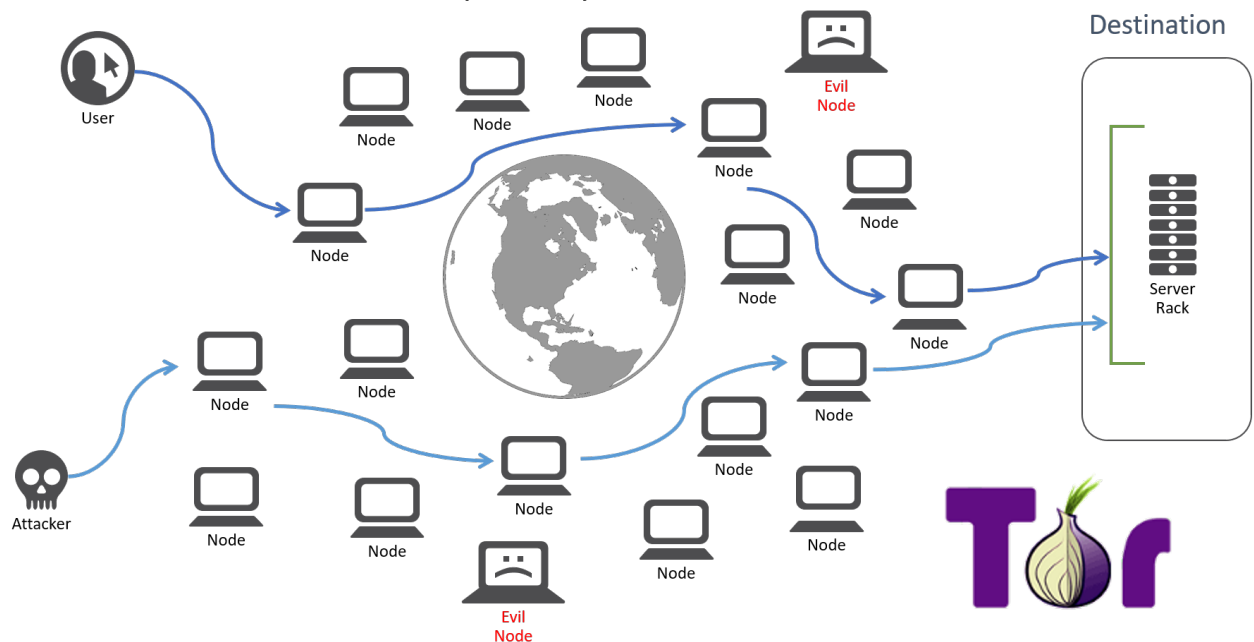


Figure 1.1: Tor Network (source = <https://www.f5.com/labs/articles/threat-intelligence/snooping-on-tor-from-your-load-balancer>)

## 2.1 Tor network architecture

To make the Tor Network easier to implement and debug, the architecture was split into several parts. The 4 main parts are:

- The Server
- The Relays
- The Client
- The Tor Network

Each of these parts will have its own python code. The goal is that the clients, the server and the relays form a network. This will be done through the Tor network python code that will link them all together.

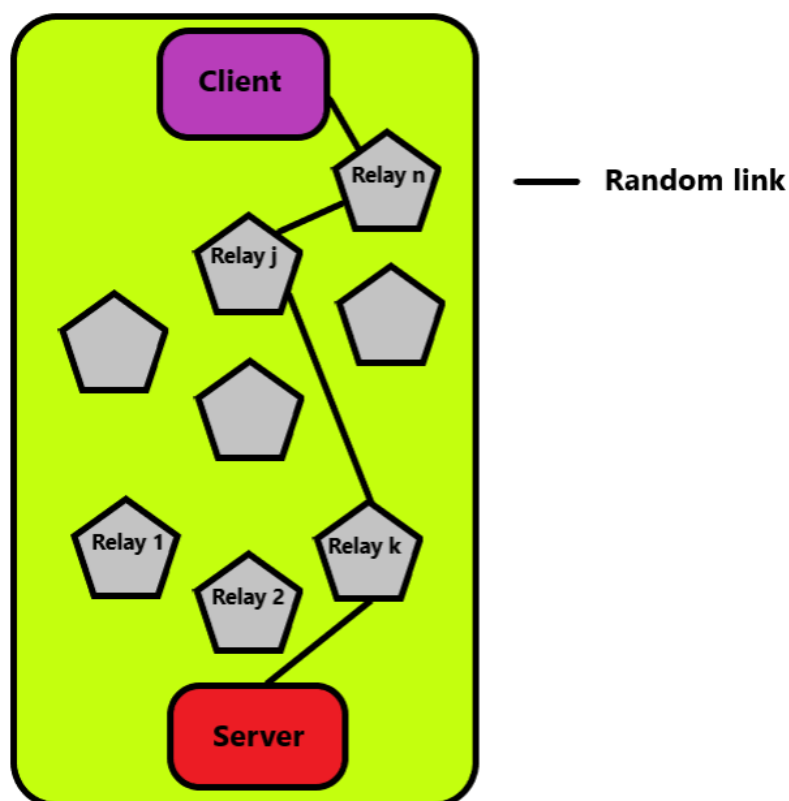
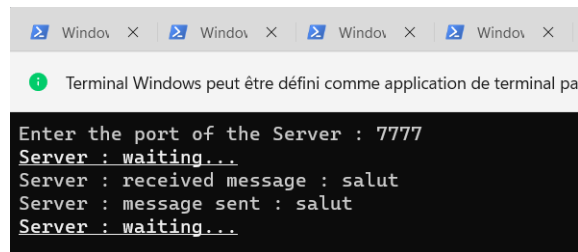


Figure 2.1: Global architecture

In the end, the objective is that when the client wants to communicate with the server, the latter will do so through 3 relays that will be chosen randomly. The relays will act in the same way as routers.

### 2.1.1 Server

The server is created using a library : *custom\_tor*. This library was created for this project and will be seen later (2.1.4). From this library, an object "server" is created thanks to *Server* class from the library. The server job is to be able to send back a message whenever he receives a message.



```

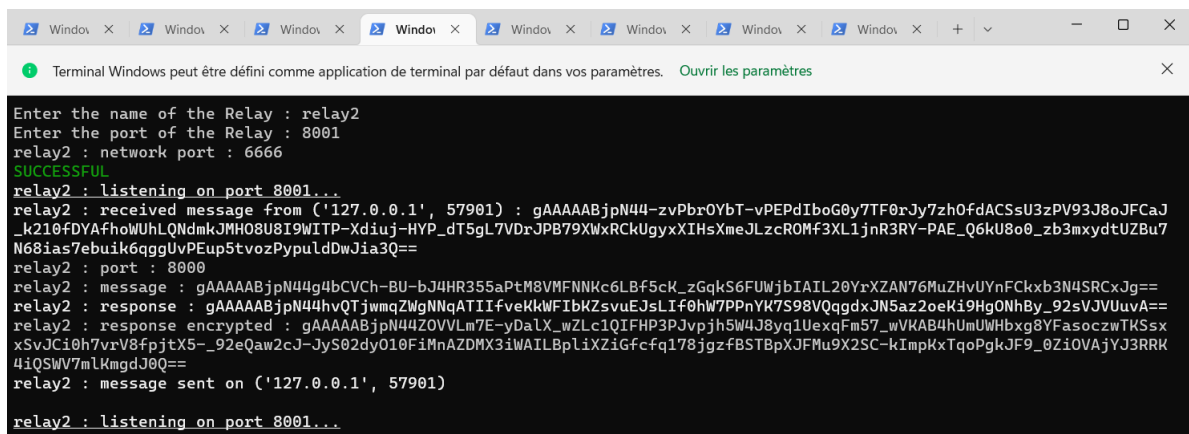
Enter the port of the Server : 7777
Server : waiting...
Server : received message : salut
Server : message sent : salut
Server : waiting...

```

Figure 2.2: Server terminal example

### 2.1.2 Relays

The same library (*custom\_tor*) is used to create relays. From this library, an object "relay" is created thanks to *TorRelay* class from the library. Of course, since the Tor network needs multiple relays, multiple object "relay" can be created at the same time and have all different proprieties. Furthermore, thanks to a function from the library (*joinTorPool*), all the relays can be present in the same Tor network if they have been well configured. Finally, the relays have been programmed to be in permanent listening mode.



```

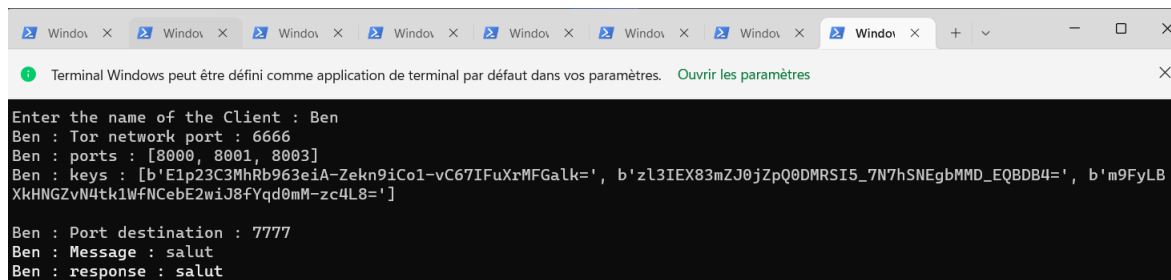
Enter the name of the Relay : relay2
Enter the port of the Relay : 8001
relay2 : network port : 6666
SUCCESSFUL
relay2 : listening on port 8001...
relay2 : received message from ('127.0.0.1', 57901) : gAAAAABjpN44-zvPbr0YbT-vPEPdIboG0y7TF0rJy7zh0fdACSsU3zPV93J8oJFCaJ_k210fDYAfhoWUHLQNdmkJMH08U8I9WITP-Xdiuj-HYP_dT5gL7VDrJPB79XWxRCKUgyxXIHSXmeJLzcROMf3XL1jnR3RY-PAE_Q6kU8o0_zb3mxydtUzBu7N68ias7ebuik6qggUvPEup5tvozPypuldDwJia3Q==
relay2 : port : 8000
relay2 : message : gAAAAABjpN44g4bCVCh-BU-bJ4HR355aPtM8VMFNKc6LBf5cK_zGqkS6FUWjbIAIL20YrXZAN76MuZHvUYnFCkxb3N4SRCxJg==
relay2 : response : gAAAAABjpN44hvQTjwmqZWgNNqATIIIfveKkWFibKZsvuEJsLI f0hW7PPnYK7S98VQggdxJN5az2oeKi9HgONhBy_92sVJVUuvA==
relay2 : response encrypted : gAAAAABjpN44ZOVVLm7E-yDaLX_wZLc1QIFHP3PJvpjh5W4J8yq1UexqFm57_wVKAB4hUmUWHbxg8YFasoczwTKSsx x5vJCi0h7vrV8fpjtX5-_92eQaw2cJ-JyS02dy010FiMnAZDMX3iWAILBpliXZiGfc-fq178jgzfB8TBpXJFMu9X2SC-kImpKxTqoPgkJF9_0Zi0VAjYJ3RRK4iQSWV7mLkmgdJ0Q==
relay2 : message sent on ('127.0.0.1', 57901)
relay2 : listening on port 8001...

```

Figure 2.3: Relay terminal example

### 2.1.3 Client

Once again, the library (*custom\_tor*) is used to create this time an object "client". It is done thanks to the *Client* class from the library. Also this object, can send message through the Tor network thanks to the function *SendViaTor* from the library.



```


Enter the name of the Client : Ben
Ben : Tor network port : 6666
Ben : ports : [8000, 8001, 8003]
Ben : keys : [b'E1p23C3MhRb963eiA-Zekn9iCo1-vC67IFuXrMFGalk=', b'z13IEX83mZ0jZpQ0DMRSI5_7N7hSNEgbMMD_EQBDB4=', b'm9FyLBXkHNGZvN4tk1WfNCebE2wiJ8fYqd0mM-zc4L8=']
Ben : Port destination : 7777
Ben : Message : salut
Ben : response : salut

```

Figure 2.4: Client terminal example

### 2.1.4 Custom Tor Network

All this objects that can be created needs to be linked somehow : here comes the 'custom\_tor' library that they all share. First of all, it is necessary to know which are the other libraries that are used.



```

import random
import os
from cryptography.fernet import Fernet
from socket import socket, AF_INET, SOCK_STREAM
from termcolor import colored

```

Figure 2.5: Libraries used for customTor script

The most important library is *socket*, that allows the creation and reception of sockets that are necessary for the communication. The library *cryptography.fernet* is used for encryption/decryption of messages. As seen in the figure 2.1, the 'custom\_tor' script is the skeleton of the Tor network. As seen above, 3 classes need to be created and also multiple functions need to be associated to them.

First of all, the class 'Client' needs to be defined. It has several attributes such as its name, key and port. Without going into too much detail, different functions are associated with this class.

A 'ConnectToTorNetwork' function, which allows as its name indicates the connection of the client object to the Tor network. It only requires the knowledge of the port of the Tor Network since we work locally. The 'SendViaTor' function allows to send an encrypted message via the Tor Network by knowing the destination port.

Secondly, the TorRelay class has as attributes a name (which is null by default), a port (which is also null by default), a key, an encryptor (similar to the one used for sending client messages), and a listening socket. In order to work properly, functions have been created. The 'listen' function allows this relay to go to a port that the user specifies and to listen if a message is transmitted. Depending on several possible scenarios, the relay acts in a different way (there is an error management). The 'joinTorPool' function allows to give a name and port attribute (only if it is available) to the relay so that it joins the Tor network.

Finally, the last class created is the server. It takes as attribute a port. Once a server object is created and with a valid port assigned, it will permanently listen if it receives a message and in this case send a message back.

```

Enter port where the Network will listen : 6666
Tor network : waiting...
Request received from ('127.0.0.1', 57853) : JOIN pool 8000 b'E1p23C3MhRb963eiA-Zekn9iCo1-vC67IFuXrMFGalk='
New relay in the network
List of relays :
8000 b'E1p23C3MhRb963eiA-Zekn9iCo1-vC67IFuXrMFGalk='
Tor network : waiting...
Request received from ('127.0.0.1', 57864) : JOIN pool 8001 b'zL3iEX83mZJ0jZpQ0DMRSI5_7N7hSNEgbMMD_EQBDB4='
New relay in the network
List of relays :
8000 b'E1p23C3MhRb963eiA-Zekn9iCo1-vC67IFuXrMFGalk='
8001 b'zL3iEX83mZJ0jZpQ0DMRSI5_7N7hSNEgbMMD_EQBDB4='
Tor network : waiting...
Request received from ('127.0.0.1', 57873) : JOIN pool 8003 b'm9FyLBXkHNGZvN4tk1WfNCebE2wiJ8fYqd0mM-zc4L8='
New relay in the network
List of relays :
8000 b'E1p23C3MhRb963eiA-Zekn9iCo1-vC67IFuXrMFGalk='
8001 b'zL3iEX83mZJ0jZpQ0DMRSI5_7N7hSNEgbMMD_EQBDB4='
8003 b'm9FyLBXkHNGZvN4tk1WfNCebE2wiJ8fYqd0mM-zc4L8='
Tor network : waiting...
Request received from ('127.0.0.1', 57888) : GET relays
[('8000', "b'E1p23C3MhRb963eiA-Zekn9iCo1-vC67IFuXrMFGalk="), ('8001', "b'zL3iEX83mZJ0jZpQ0DMRSI5_7N7hSNEgbMMD_EQBDB4="), ('8003', "b'm9FyLBXkHNGZvN4tk1WfNCebE2wiJ8fYqd0mM-zc4L8=")]
Tor network : waiting...

```

Figure 2.6: Tor Network terminal example

## 2.2 Challenge Response

It was requested to perform an authentication based on challenge-response. The goal is that a client can authenticate correctly with a password without leaking it. In short, the client addresses the server through the Tor network, receives a challenge (a password request for example) and answers it. If he answers correctly, he is authenticated otherwise he is denied access to the server.

A more interesting challenge-response technique uses a function to calculate the response from the challenge. Suppose a client wants to access a resource controlled by the server. To authenticate the client, the server sends a challenge (for example a string) and expects in return a response. The response is computed on basis of the given challenge and is then compared to an expected value (computed by the server). If the value is the same, the access is not denied. Otherwise another challenge is sent to the client. In a real protocol, the calculation of the response is quite complex (for instance using SHA encryption). As the challenge sent to the client is random, knowing the response to a previous challenge does not help an adversary who wants to impersonate a legitimate user. This was the challenge-response based authentication implemented. However, another option was added and will be explained in the chapter 3.

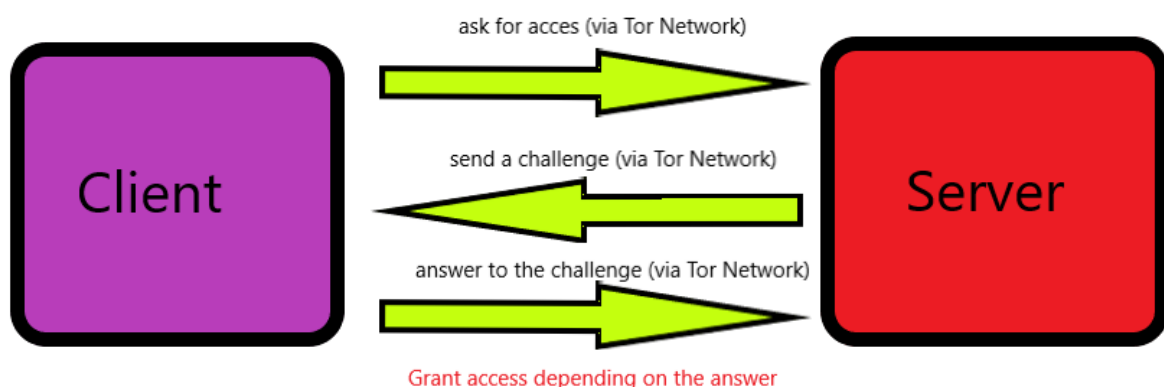


Figure 2.7: Challenge-Response principle

### 3.1 Multiple terminals

One of the functionalities that was added to the original project is the following : A commands windows script was created that allows to open multiple terminals. Each terminal allows the user to configure the relays, the client and the server. This was possible thanks to the use of the OS library. The library allows to display lines of text (it can be a question or a sentence indicating the state of the network) and wait for an input from the user. Of course all the terminals are working simultaneously. It's a good way to visualize the network interactions. Finally, different font and text color have been used for the terminal display.

```
Tor network : waiting...
Request received from ('127.0.0.1', 54231) : JOIN pool 8000 b'V4yq2U7LZ8z11swEr6H9UjbN1GXfdh6SgjFvsgMLsIg='
New relay in the network
List of relays :
8000 b'V4yq2U7LZ8z11swEr6H9UjbN1GXfdh6SgjFvsgMLsIg='

Tor network : waiting...
Request received from ('127.0.0.1', 54251) : JOIN pool 8002 b'eT24___9oLimhRKMV7IhLgy6SNpaFBoVI-NY7-YBVFY='
New relay in the network
List of relays :
8000 b'V4yq2U7LZ8z11swEr6H9UjbN1GXfdh6SgjFvsgMLsIg='
8002 b'eT24___9oLimhRKMV7IhLgy6SNpaFBoVI-NY7-YBVFY='

Tor network : waiting...
```

Figure 3.1: Example of terminal display

### 3.2 Challenge Response : key

For the challenge response, another way different from the one presented above has been realized. The customer has a password this time. When he wants to contact the server he will receive another challenge (which changes each time), the client will answer by concatenating his password with the challenge and encrypting it with the SHA-256 algorithm. This allows two levels of security and ensures a unique key. Obviously, the server must know the client's password though. The user will only enter his password, the client takes care of the rest.

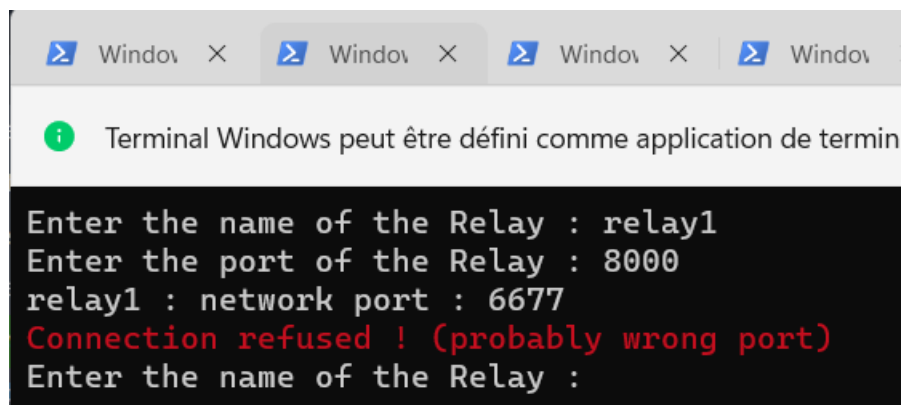


### 3.3 Http links opener

An option that has been added is the opening of a web page on the default browser of the user's computer. When the user asks the server for a web page, the server sends it back and asks the client to open the web page concerned. This was done using the *webbrowser* library.

### 3.4 Error message management

Error handling was added to the code. Indeed, if the user enters values that are not correct or not the one requested by the terminal, then errors will be displayed in red on the terminal with the potential cause. In addition, if there is an error, the user is sent back to the starting point to enter the correct data this time. Several scenarios have been thought and allow the code to work whatever the data entered by the user.

A screenshot of a terminal window. At the top, there are four window tabs, each labeled 'Window' with a blue icon and a close button. Below the tabs, there is a green information icon followed by the text 'Terminal Windows peut être défini comme application de termin'. The main area of the terminal is black with white text. It shows the following sequence of input and output: 'Enter the name of the Relay : relay1', 'Enter the port of the Relay : 8000', 'relay1 : network port : 6677', 'Connection refused ! (probably wrong port)' (where the error message is in red), and 'Enter the name of the Relay :'.

```
Window X Window X Window X Window X
Terminal Windows peut être défini comme application de termin

Enter the name of the Relay : relay1
Enter the port of the Relay : 8000
relay1 : network port : 6677
Connection refused ! (probably wrong port)
Enter the name of the Relay :
```

Figure 3.2: Example of error displayed on terminal

## 4.1 Libraries

The first difficulty was finding libraries that were useful for the project. It was forbidden to use complete Tor Network libraries, and these are the most common. This difficulty was the easiest to solve with some simple research, however. A number of libraries were found, including ones for sockets, encryption, and color management. The set of libraries used are :

- *hashlib* : allows hashing (hashing is a method of generating a fixed-size, unique output called a hash value or hash code from an input called the key )
- *random* : allows creating random values
- *os* : allows to interact with the operating system
- *socket* : allows the use of sockets
- *cryptography.fernet* : provides a simple and secure way to encrypt and decrypt data using symmetric encryption
- *termcolor* : allows to use color and formatting text in the terminal
- *webbrowser* : allows to open a web page from an URL

## 4.2 Socket management

The biggest difficulty encountered is the management of sockets. At the beginning, when the sockets were used the python script returned errors. Many tests have been done to find out where the problem was. The most conclusive of them was to send a single message and then several messages towards the server. What was interesting was to notice that for the single message, the python script did not return an error but for several messages it returned an error. This is when the problem was revealed: it is not possible to reuse a socket for several communications with the server, which was done in the script. Therefore the problem was solved by creating a new socket for each new communication with the server. The mistake that was made was to use the sockets for the server in the same way as for the client. The client can reuse the sockets unlike the server.

## **Access to the GitHub**

You can find below the link to the GitHub:

<https://github.com/T0m-DC/ELEC-H417-Tor-Network>