Université Libre de Bruxelles - Bruface

ELEC-H-417 - Communication networks : protocols and architectures

Shallot routing
Project Report

Keneth Ubeda
Arthur Valingot
Shafagh Kashef
Razieh Manshadian

First semester 2017-2018

Contents

1 Introduction

The project within the framework of the Communication Network course consists to implement a Shallot network Project in Python. The goal is to familiarize with different notions of the course and more specifically, to know how a TOR-like network works.
This project is decomposed in four steps. The first step is the implementation of the network (Relay, Server, Client, ...) The second was the specific implemention of the Dijskstra Algoritm in order to have a randomized routing. The third one was about the diffie hellman algorithm. The last but not least was the AES encryption.

2 Step 1

To implement this project, we've built a set of classes which contains the different functionalities we'll use to implement Alice, Bob and the relay network.

2.1 Classes

Note: for the socket implementation we are using the socket default library from python.

```
import socket
```

2.1.1 Client

This class contains the logic to start a socket client connection indicating the host and port of the server you want to connect. Basically, we can set the connection parameters, start the connection, close the connection and send a message. For the send message functionality, **it's important to mention that the client is waiting until receives a response from the server it has connected. Other important observation is the fact that the client socket chose a random port to establish its communication with the server wants to connect, however is possible to specify the port you want to use to establish the connections using the binding option**.

```
self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.socket.connect((self.host, self.port))
```

- **AF_INET** address family is the address family for IPv4.
- **SOCK_STREAM** Supports reliable connection-oriented byte stream communication.

2.1.2 Server

This class contains the logic to start a socket server connection, that means we are going to bind the socket and start listen in the host and port we've configured. Essentially in this class we can set the connection parameters, start a new connection where the socket will listen for client requests. **It's important to mention that each time we are accepting a new connection from a client It's completely necessary create a new thread to accept more than one client connection**.

```
self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.socket.bind((self.host, self.port))
self.incoming_conn, self.incoming_addr = self.socket.accept()
```

2.1.3 Relay

The relay class is a combination of the 2 last ones, this is because a relay needs to be listening but to forward the information must be a client of another socket server. So, the relay structure is very similar to the server class with the only difference of the client implementation inside it. When a relay has forwarded a message it will wait until receive the response from the relay it has sent the message, or if the next it's not a relay but Bob, then it waits for the Bob response.

2.1.4 Loader

This class can read either the host.ini or topology.ini files. This is useful because each time either Alice, Bob or the relay network starts they must read the configurations file to start its connection in with the right parameters.

2.2 Executable

2.2.1 Alice

In this implementation happens many things:

- The first one is the key negotiation, consists in send a KEY_INIT message to each node in the network to get the key Alice must use to encrypt the message at each layer of the shallot.
- Then Alice is ready to send messages to Bob but, just before send the message Alice needs to build the shallot.
- Calculate the random Dijkstra's algorithm, this step is a requisite to build the shallot. Basically, Alice access to the class that calculate the lowest cost random path and receives an array with it.
- Building the shallot, this step consists in take the result path from the random Dijkstra's algorithm and starts to iterate backwards building the shallot from Bob to the nearest node.

So, Alice must be able to build RELAY_MESSAGES to send to the relay she is connected to. Once the shallot is completed she can send the message. She will wait until receive a response.

2.2.2 Bob

Basically, Bob is an instance of the Server class. He reads its address from the file using the Loader class, and he is listening for a message from Alice. He must be able to decrypt the message using the AES implementation. Bob also must be able to receive KEY_INIT messages and build a KEY_REPLY message to send Alice the key.

2.2.3 Relay network

In order to start the relay network, the first thing this implementation does, is read the host.ini file, to get the addresses and the number of relays, to launch the needed instances of the Relay class. The relays must be able to build KEY_REPLY and ERROR messages and to read the RELAY messages to. To forward the message they have to apply the AES decryption using its own key to get the next hop and the next message they have to send. **To launch all the instances, we are using a Thread pool**.
Note: for the Thread pool we are using the following library:

```
from multiprocessing.dummy import Pool as ThreadPool
```

3 Step 2 – randomized routing

After implementing the relays, the routing path from Alice to Bob should be selected. Alice will make use of a modified Dijkstra algorithm. The idea of randomizing the routing is assigning random costs to each link before sending message by Alice. Therefore, after loading the topology, the cost of each link will be randomly selected between 1 and 16. Next, Dijkstra's algorithm is used to compute the least cost path. The routing path contains the ordered list of relays to be used in that session

3.1 Functional description

In order to implement Dijkstra algorithm, we used class graph. The graph is represented by adjacency matrix. After creating the matrix, random costs is assigned. Remark that this implementation consists of 2 parts: 1.Fining the least cost path and 2.printing the path as Array.

3.1.1 Finding the least cost path by Dijkstra algorithm

The steps for implementing the Algorithm briefly:

- Defining the Adjacency matrix which represents the graph
- Assigning randomized cost values to each link by "from random import randint"
- Define a initially empty set(shortest path tree set) by using function "minDistance(self,dist,queue)" and using function "dijkstra(self, graph, src)"
- Assign a distance value to all vertices in the input graph(0 to source and infinity to other nodes)
- Pick the vertex with minimum distance value(source:0)
- Update distance values of its adjacent vertices
- Loop:repeat the above steps until the shortest path tree set doesn't include all vertices of given graph
- The result of these steps is the least cost Path Tree

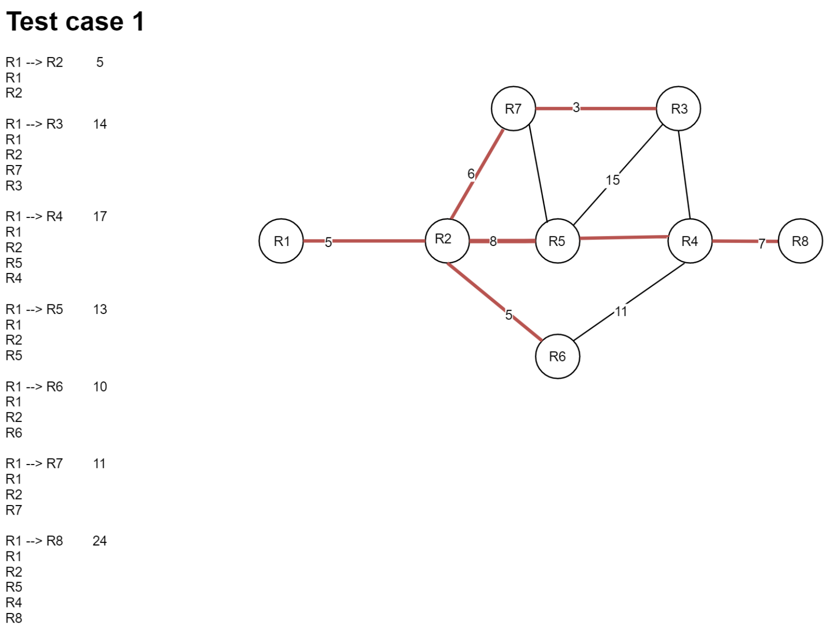
3.1.2 Printing the least cost path:

- creating a separate array parent[]
- Value of parent[j] for a vertex j, stores parent vertex of v in shortest path tree. Parent of source vertex(Alice) is -1. Whenever we find shorter path through another vertex, we make that vertex as parent of current vertex.
- Defining functions "printPath(self, parent, j)" and "printSolution(self, dist, parent)" to print the least cost path from Alice.

By this Algorithm, we can also see the least cost path from source to each vertex. It can be useful for another scenario, when we do not know which node is source and which one is destination and we can choose the least cost between 2 required nodes.

3.2 Test case

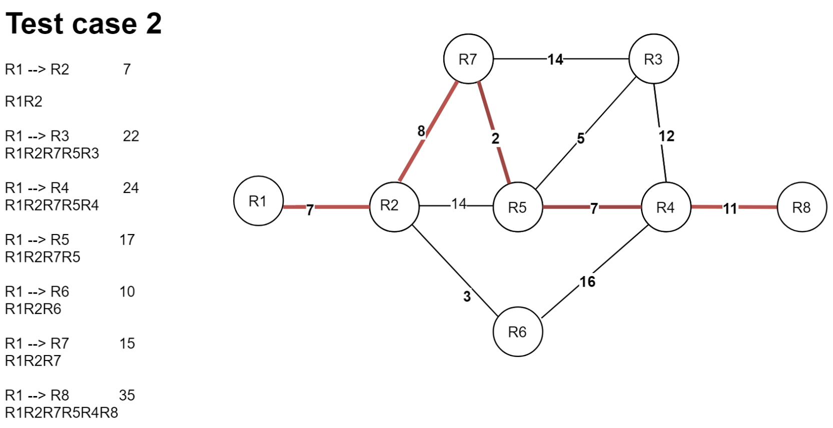
In order to check the validity of applied modified Dijkstra algorithm, some test cases are done and the results of 2 test cases are shown below:



Test case 1

```
R1 --> R2       5
R1
R2

R1 --> R3       14
R1
R2
R7
R3

R1 --> R4       17
R1
R2
R5
R4

R1 --> R5       13
R1
R2
R5

R1 --> R6       10
R1
R2
R6

R1 --> R7       11
R1
R2
R7

R1 --> R8       24
R1
R2
R5
R4
R8
```

Figure 1: Finding the least cost path by assigning randomized cost to each link



Test case 2

```
R1 --> R2          7

R1R2

R1 --> R3          22
R1R2R7R5R3

R1 --> R4          24
R1R2R7R5R4

R1 --> R5          17
R1R2R7R5

R1 --> R6          10
R1R2R6

R1 --> R7          15
R1R2R7

R1 --> R8          35
R1R2R7R5R4R8
```

Figure 2: Finding the least cost path by assigning randomized cost to each link

Remark: In this case the least cost path from R1 to R2 is through R1,R2,R7,R5,R8 by cost 35. It is shown in red. In our topology we considered R2 as the closest node to Alice and R4 the closest node to Bob.

3.3 Sequential diagram

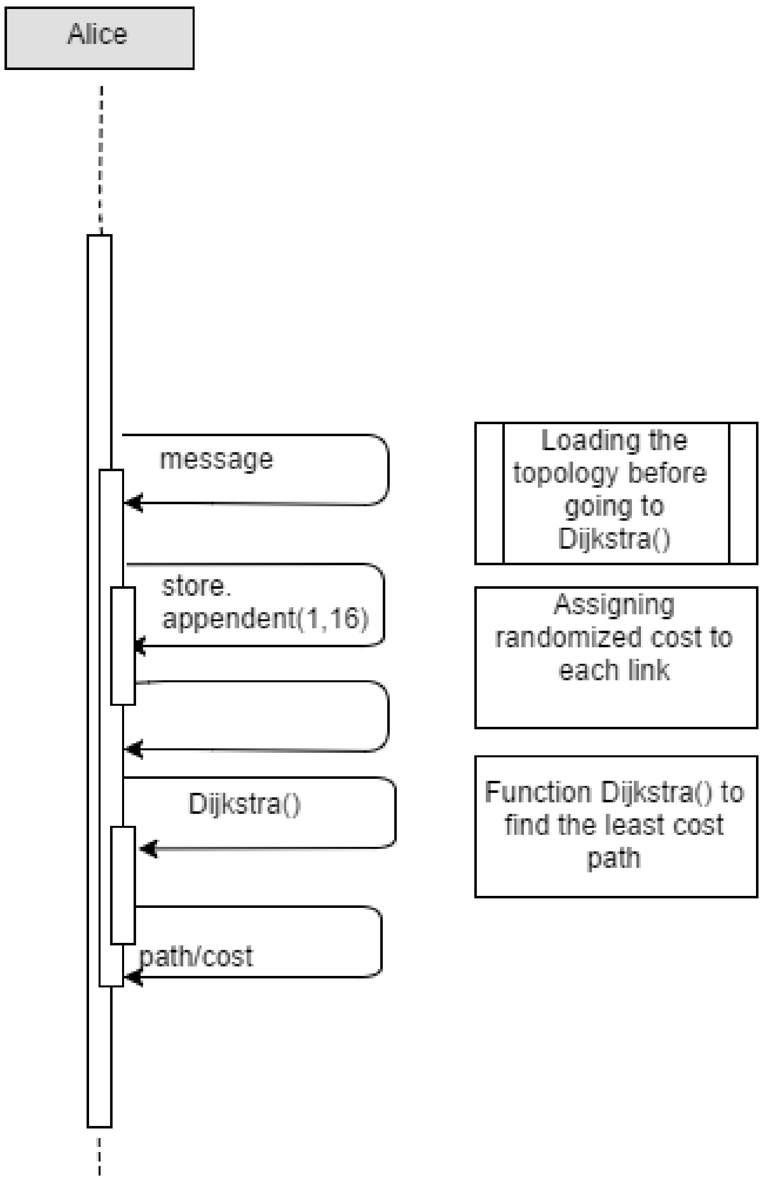

Alice
message
store.
appendent(1,16)
Dijkstra()
path/cost
Loading the topology before going to Dijkstra()
Assigning randomized cost to each link
Function Dijkstra() to find the least cost path

Figure 3: Selecting the least randomized cost route by applying modified Dijkstra algorithm

4 Step 3 - Diffie Hellman (Elliptic curve version)

In order to improve the security of our algorithm, we chose to implement a basic version of the Elliptic curve, with the coefficient of the curve which belong to GF(256) which is a Gallois Field with 256 elements.

4.1 Finite Field (GF(256))

This part has been inspired by the following document. This document explains how to implement the basic operation as multiplication and addition in a Gallois Field. The GF(256) is represented with the following idea $\frac{\frac{\mathbb{Z}}{2\mathbb{Z}}}{X^8+X^4+X^3+X+1}$ this means that the polynomial with their coefficient $\frac{\mathbb{Z}}{2\mathbb{Z}}$ will be considered, but for each operation, only the rest of the result by the polynomial $X^8 + X^4 + X^3 + X + 1$ will be considered. In the code, the reader will find the class FiniteField, the main idea of this class is to overload the following basic methods:

- __truediv__ this method uses a simple polynomial division with binaries coefficients and return the division
- __mod__ this method uses a simple polynomial division with binaries coefficients and returns the rest
- __mul__ this method uses the first algorithm explain the document above, it coub be improved.
- __add__ this method uses the addition of binary coefficients coordinate by coordinate

This way has been chosen, because it allows the programmer to use the following code. Let a and b two FiniteField elements, it's possible to directly write a/b, a*b, a%b, a+b in the code. This notation will respectively call the function __truediv__, __mul__, __mod__, __add__.

The operation of the FiniteField has been tested. The main test was to find every inverse element for each element in GF(256), and compare the result with the table given in the document above.

In the following section, only the finitefield will be used to implement the operation on the ellipticCurve.

4.2 Elliptic Curve

The main operation on the elliptic curve are described on the following web site, and the multication algorithm has been mostly inspired by this document. The first document describe the kind of elliptic curve which has been used in this implementation.

The reader will find in the code a file named EllipticCurve, in this file they will find three classes.

The first one is EllipticCurvePoint which describe every mathematical operation use between two points as the addition and the multiplication. In order to implement these both operation, the algorithms have been used, and the same way as before with the FiniteField, this means that the method __mul__ and __add__ have been overloaded.

Let's remark that the algorithm use to the multiplication with a constant is not the simplest one which consist in the idea to sum n times the EllipticCurvePoint with itself to have the good result. The algorithm describe in the second document given above describes a much faster way to implement itr.

The second class is EllipticCurveNeutralEl, this class has been created to behave as a neutral element should do. Let a an EllipticCurvePoint and b EllipticCurveNeutralEl, thus $a + b = a$. For most of the algorithms, this neutral element is required.

The third and last class is EllipticCurve, this class contains each element use to define the an elliptic curve, which are two finite field elements a and b (the coefficients of the elliptic curve). In order to be faster and compute a random EllipticCurve fast, the coefficient b is computed by the method workout_b, and three argument will be given to the elliptic curve x, y and a. x and y are two finite field randomly generated their represent the two coordinates of an EllipticCurvePoint.

4.3 DiffieHellman

Everything is properly defined to implement the Elliptic version of DiffieHellman. Alice has an EllipticPoint P, and EllipticCurve G and a random integer n_a. Alice will compute the $A = n_a P$. Alice will send to bob P, G, and A. Bob will take a random integer n_b and compute $B = n_b P$.

Bob will send B to Alice. At this time, Bob can compute $C = n_bA$ and Alice can compute $C = n_aB$. They will have shared a key together.

5 Step 4

5.1 AES

In this step the new message is encrypted using the AES algorithm and the key negotiated by Alice and R. The Advanced Encryption Standard (AES) is the most widely used symmetric cipher today. Even though the term "Standard" in its name only refers to US government applications, the AES block cipher is also mandatory in several industry standards and is used in many commercial systems. Among the commercial standards that include AES are the Internet security standard IPsec, TLS, the Wi-Fi encryption standard IEEE 802.11i, the secure shell network protocol SSH (Secure Shell), the Internet phone Skype and numerous security products around the world. To date, there are no attacks better than brute-force known against AES

5.2 Functional description of solution

To run this program we installed "pip" it does not come pre-installed with python."pip" is an easy installer extremely simple to install modules.

First of all we imported something from Crypto.Cipher import AES we've used AES encryption and we use an online generated key. We have Message variable. For key component we need to create a cipher object from Pycrypto cipher = AES.new(key) so we have a cipher object that we can decrypt and encrypts things with it. we've used a function called "pad", it basically gets the length of our message and it adds the amount of curly brackets to the message to make it divisible by 16 the reason we have to do this is because a AES encryption only takes data with length of 16 divisible lengths so we defined our function called it pad with a 1 parameter def pad(s): return s + ((16-len(s)

It get us the number of characters that we need to make our link divisible by 16 and it's going to multiply that by this curly bracket so it'll put that many curly brackets onto the end of "s". Then we made the encrypt function"def encrypt(plaintext)":global cipher return cipher.encrypt(pad(plaintext))

For decrypt function we put "utf-8" otherwise it give us the bytes version of message.We don't need the curly braces on there that we attached earlier so l = dec.count('') with this code we count how many of those we have now it returns everything up until the first curly brace so we're not actually going to get any of the curly braces so now we're going to say print message, encrypted message

print("Message:", message) encrypted = encrypt(message) decrypted = decrypt (encrypted) print("Encrypted:",encrypted) print("Decrypted", decrypted)

5.3 Sequential diagram

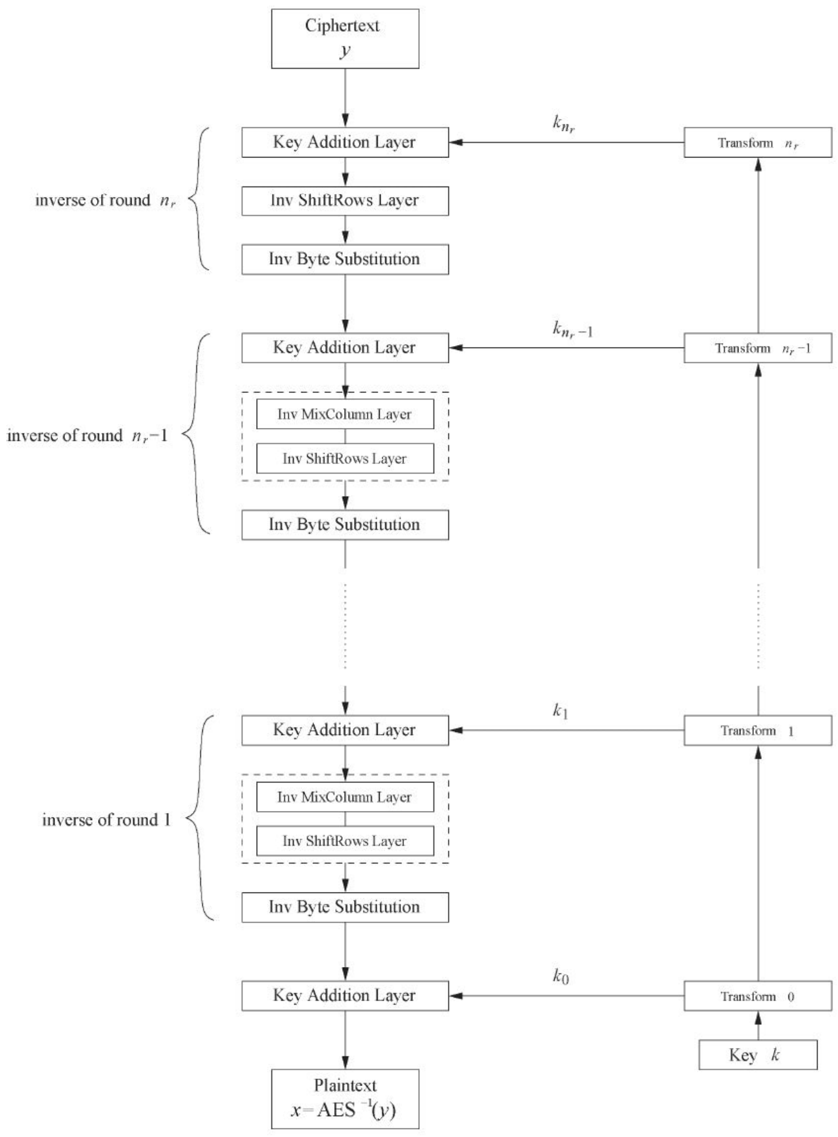


Figure 4: ASE algorithm

6 Conclusion

The project was very interesting because of a lot of concepts linked to the communication network. For example the importance of Combination of network with cryptography in order to have a secure network. We encountered some problems during working on it. One of the difficulties was the socket connection. It didn't work sometimes as we expected, but we could manage it starting with a simple connection and then start adding complexity to the solution. As far as we split the tasks in the begin of project, we had some problems with merging four steps but we could solve that after meeting. The key to success this project was starting with simple things and then adding complexity step by step.