# Sistem distribuit de comunicare între stații

Studenți:
Chirica Andrei, grupa 1407A
Poclid Ionuț-Andrei, grupa 1406B

## ❖ Introducere

În era contemporană,comunicarea eficientă și sigură între dispozitive este absolut necesară. Sistemul nostru de comunicare își propune să faciliteze schimbul de informații,dar și securitatea datelor transmise între laptop-uri sau alte dispozitive.
Pentru a realiza acest proiect am folosit un algoritm de criptare simetric(RC6), protocolul de comunicație TCP, limbajul de programare Python precum și mediile de dezvoltare PyCharm și Visual Studio Code.

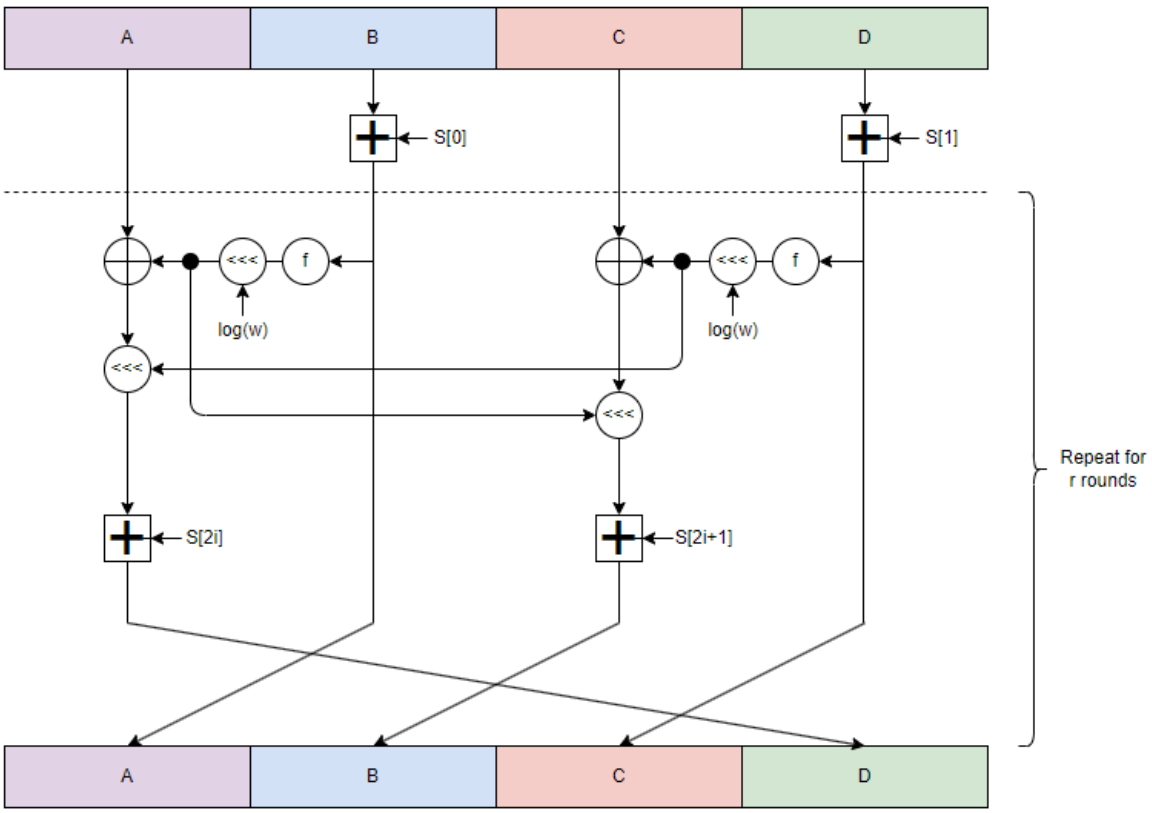
## ❖ Algoritmul RC6

În criptografie, RC6 (Rivest cipher 6) este un cifru bloc de chei simetrice derivat din RC5. A fost proiectat de Ron Rivest, Matt Robshaw, Ray Sidney și Yiqun Lisa Yin pentru a îndeplini cerințele competiției Advanced Encryption Standard (AES). Algoritmul a fost unul dintre cei cinci finaliști și, de asemenea, a fost depus proiectelor NESSIE și CRYPTREC. A fost un algoritm proprietar, brevetat de RSA Security.

RC6 propriu-zis are o dimensiune de bloc de 128 de biți și acceptă dimensiuni de cheie de 128, 192 și 256 de biți până la 2040 de biți, dar, ca și RC5, poate fi parametrizat pentru a suporta o mare varietate de lungimi de cuvinte, dimensiuni de cheie și numărul de runde. RC6 este foarte asemănător cu RC5 ca structură, folosind rotații dependente de date, adunare modulară și operații XOR; de fapt, RC6 ar putea fi văzut ca împletind două procese paralele de criptare RC5, deși RC6 folosește o operație de multiplicare suplimentară care nu este prezentă în RC5 pentru a face ca rotația să depindă de fiecare bit dintr-un cuvânt și nu doar de câțiva biți mai puțin semnificativi.

*Schema algoritmului :*

The diagram below illustrates the encryption process:



*Criptare:*

```
B = B + S[0]
D = D + S[1]
for i = 1 to r:
    t = (B x (2 * B + 1)) <<< log(w)
    u = (D X (2 * D + 1)) <<< log(w)
    A = ((A ^ t) <<< u) + S[2 * i]
    C = ((C ^ u) <<< t) + S[2 * i + 1]
    A = B, B = C, C = D, D = A
A = A + S[t - 2]
C = C + S[t - 1]
```

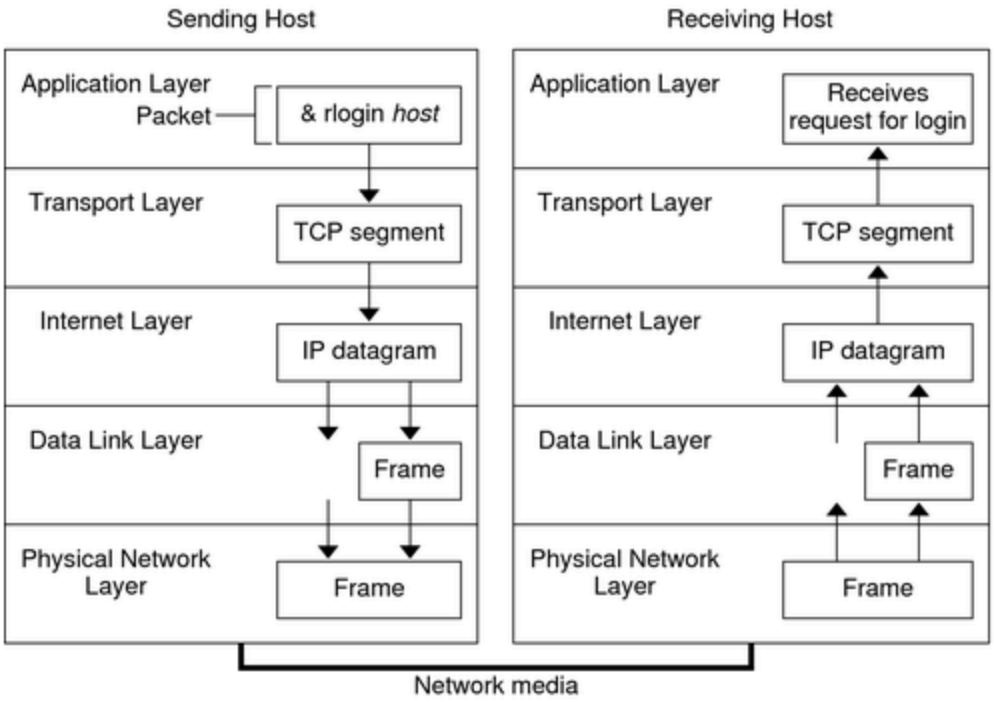
*Decriptare:*

```
C = C - S[t - 1]
A = A - S[t - 2]
for i = r to 1:
    A = D, D = C, C = B, D = C
    u = (D X (2 * D + 1)) <<< log(w)
    t = (B x (2 * B + 1)) <<< log(w)
    C = ((C - S[2 * i + 1]) >>> t) ^ u
    A = ((A - S[2 * i]) >>> u) ^ t
D = D - S[1]
B = B - S[0]
```

## ❖ Protocolul TCP

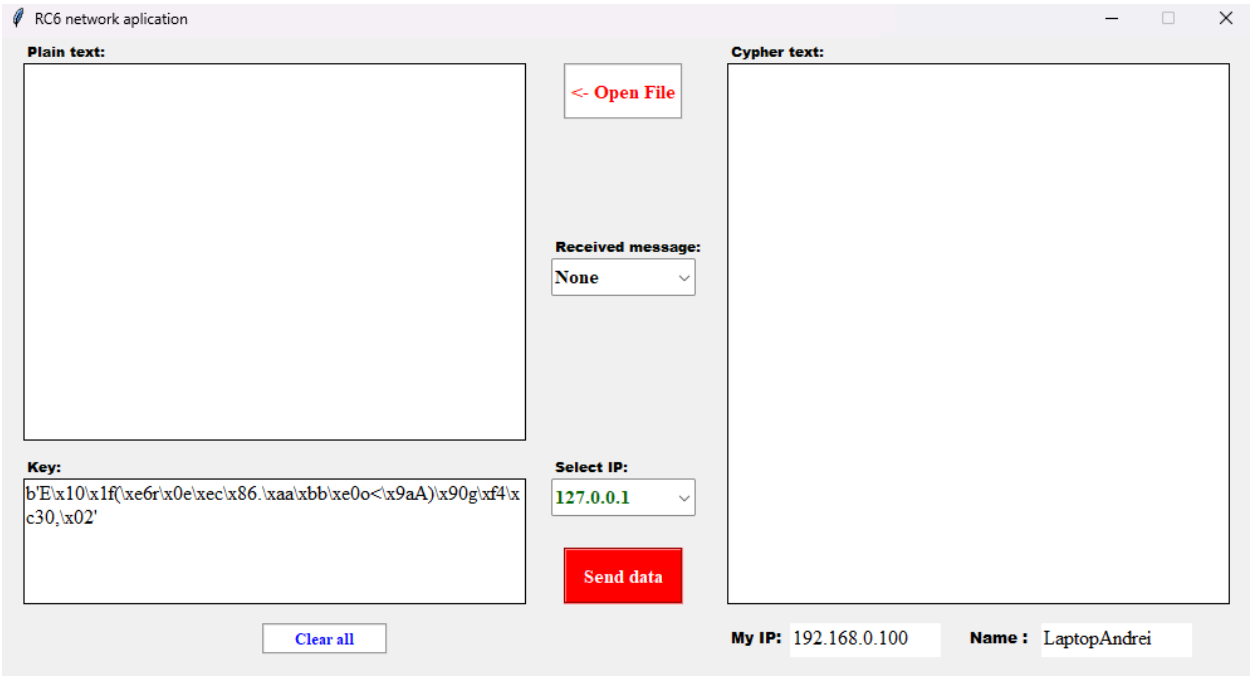
## TCP IP Model Diagram

Here is the architecture diagram of TCP IP model in computer networking:

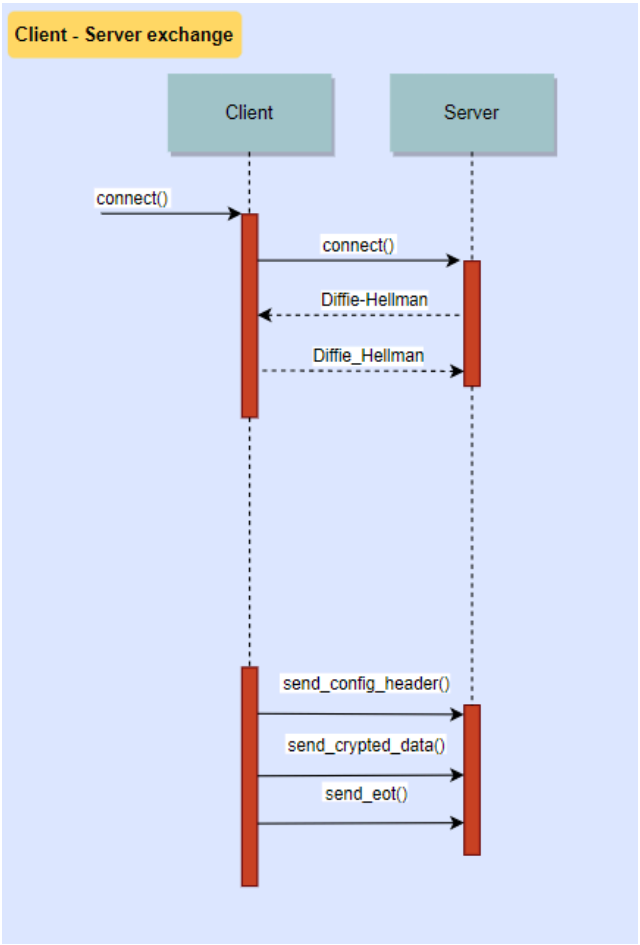



## ❖ Rezultate obținute

1. Aspect și design aplicație:




## 2. Comunicare între stații



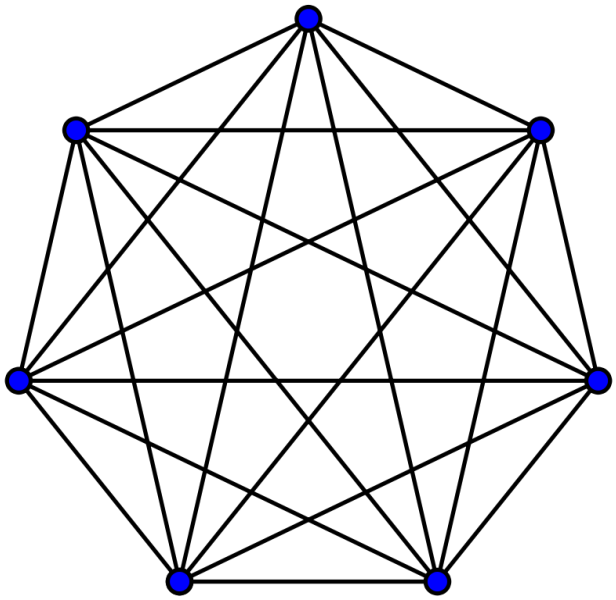
În momentul în care unul din clienți se conectează la un server din topologie acesta face un schimb de mesaje pentru a stabili cheia comuna prin algoritmul Diffie-Hellman.

După ce se stabilește cheia comuna oricare dintre cele doua parti poate trimite fișiere text spre cealalta parte in modul urmator.

1. Se trimite un pachet header de configurare care conține ***numărul de biți de 0*** de padding din mesajul criptat pentru a putea fi eliminați din mesajul decriptat și ***lungimea fisierului*** criptat în octeți
2. Se trimite mesajul criptat în calupuri de maxim 1376 de octeti
3. Se trimite un mesaj de EOT “end_of_transmission” pentru a putea anunța finalul unei tranzacții

3. Topologia de comunicare

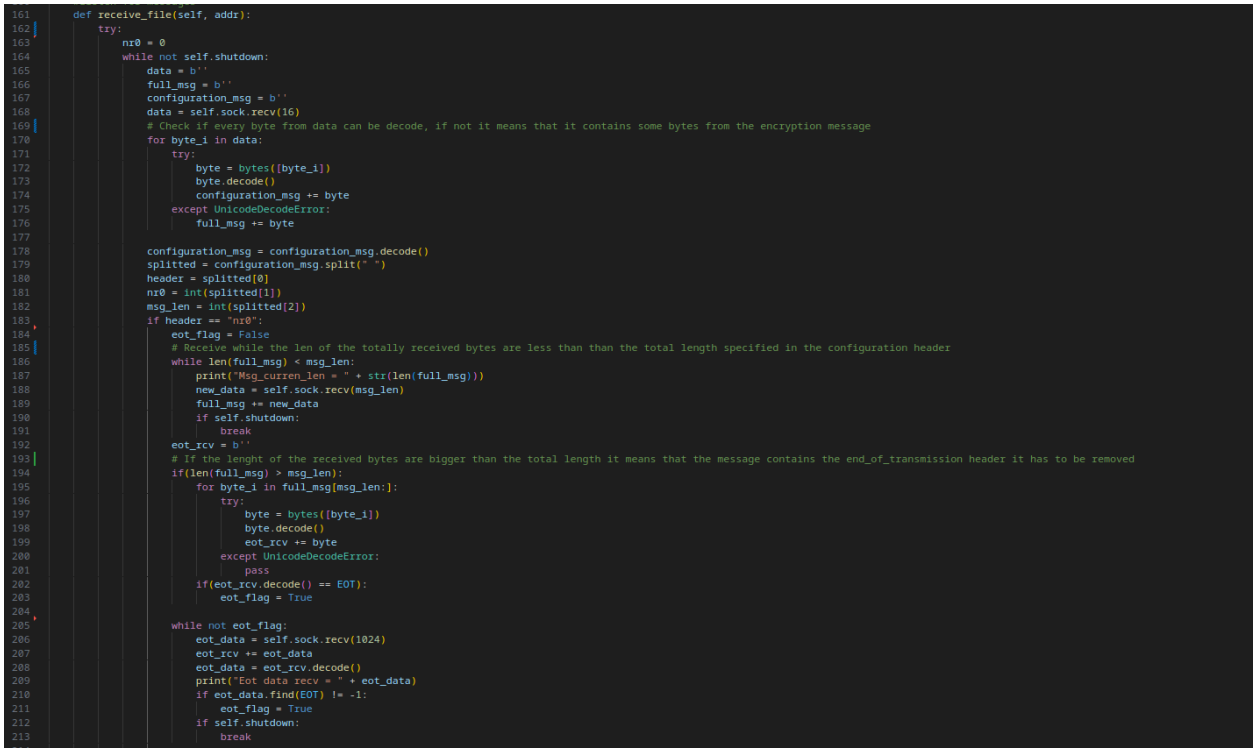
Topologia în care se pot conecta stațiile este de tip graf complet



4. Cod semnificativ:

Funcția send_file va trimite un fisier criptat la adresa primita ca parametru. Trimiterea va fi făcută fie de client, fie de server în funcție de cum fost făcută conexiunea către cealalta stație. Acest fapt este stocat în dicționarul ***connections*** care este de forma: {IP_Address : (Shared_Key, Connection_Type)}

```
266     def send_file(self, addr, file_name):
267         #Open file and read it
268
269         with open(file_name, 'r') as file:
270             text = file.read()
271
272         #Encrypt data
273         encrypted, nr0 = encrypt_variable_length(text.encode(), connections[addr][0])
274         en_splited = functions.split_in_pack_1376B(encrypted)
275
276         header_nr0 = "nr0 " + str(nr0) + " " + str(len(encrypted))
277         header_eot = EOT
278
279         #While there is still encrypted data, send data
280         #For each 1376 blocks of data sendall
281         if(connections[addr][1] == 'client'):
282             # Send configuration header
283             self.client.sock.sendall(header_nr0.encode())
284             time.sleep(0.1)
285             # For each block of 1376 bytes send data
286             for pack in en_splited:
287                 self.client.sock.sendall(pack)
288             # Send EOT header to end current transaction
289             self.client.sock.sendall(header_eot.encode())
290
291         elif(connections[addr][1] == 'server'):
292             # Send configuration header
293             self.server.clients[addr].sendall(header_nr0.encode())
294             time.sleep(0.1)
295             for pack in en_splited:
296                 # For each block of 1376 bytes send data
297                 self.server.clients[addr].sendall(pack)
298             # Send EOT header to end current transaction
299             self.server.clients[addr].sendall(header_eot.encode())
300
```



```
def receive_file(self, addr):
    try:
        nr0 = 0
        while not self.shutdown:
            data = b''
            full_msg = b''
            configuration_msg = b''
            data = self.sock.recv(16)
            # Check if every byte from data can be decode, if not it means that it contains some bytes from the encryption message
            for byte_i in data:
                try:
                    byte = bytes([byte_i])
                    byte.decode()
                    configuration_msg += byte
                except UnicodeDecodeError:
                    full_msg += byte

            configuration_msg = configuration_msg.decode()
            splitted = configuration_msg.split(" ")
            header = splitted[0]
            nr0 = int(splitted[1])
            msg_len = int(splitted[2])
            if header == "nr0":
                eot_flag = False
                # Receive while the len of the totally received bytes are less than than the total length specified in the configuration header
                while len(full_msg) < msg_len:
                    print("Msg_curren_len = " + str(len(full_msg)))
                    new_data = self.sock.recv(msg_len)
                    full_msg += new_data
                    if self.shutdown:
                        break
                eot_rcv = b''
                # If the lenght of the received bytes are bigger than the total length it means that the message contains the end_of_transmission header it has to be removed
                if(len(full_msg) > msg_len):
                    for byte_i in full_msg[msg_len:]:
                        try:
                            byte = bytes([byte_i])
                            byte.decode()
                            eot_rcv += byte
                        except UnicodeDecodeError:
                            pass
                    if(eot_rcv.decode() == EOT):
                        eot_flag = True

                while not eot_flag:
                    eot_data = self.sock.recv(1024)
                    eot_rcv += eot_data
                    eot_data = eot_rcv.decode()
                    print("Eot data recv = " + eot_data)
                    if eot_data.find(EOT) != -1:
                        eot_flag = True
                    if self.shutdown:
                        break
```

Atât serverul cât și clientul au aceeași implementare a funcției receive_file, singura diferență fiind socket-ul de pe care se face apelul funcției recv.

## ❖ Bibiliografie


[1] https://www.tutorialsfreak.com/ethical-hacking-tutorial/tcp-ip-model
[2] https://people.csail.mit.edu/rivest/pubs/RRSY98.pdf
[3] https://www.educative.io/answers/how-rc6-encryption-algorithm-works
[4] https://nmap.org/
[5] https://docs.python.org/3/library/socket.html
[6] https://docs.python.org/3/howto/sockets.html
[7] Link proiect: https://github.com/DeepSweeter/CSD-Proiect