**Tema 1 :**

*Comunicarea SERVER-CLIENTI :*

Pentru a realiza comunicarea server-client am recurs la implementarea thread-urilor. Stiind ca am doar 2 clienti am memorat descriptorii acestora intr-un vector *int descriptor[2]*, folosindu-i mai apoi pentru scriere si citire.

In server memorez initial toate cheile si vectorul de initializare. La fel si in client initializez cheia K3, care este comuna pentru toate cele 3 noduri.

Dupa ce realizez conexiunea cu cele 2 noduri. **Important :** Mai intai se conecteaza clientul A, apoi clientul B.

Functii SERVER :

static void \*treat(void \*arg) – este apelata la crearea unui thread

void raspunde(void \*arg) – initiaza comunicarea cu cele 2 noduri, dandu-le posibilitatea de a alege modul de criptare. Este important ca Clientul A sa aleaga inaite clientului B, dupa 10 secunde de la conectarea clientului B, deoarece am indicat un sleep(10) , astfel il asteapta pe A sa aleaga si lui ii revine optiunea ramasa; acestea utilizand moduri diferite de operare

void transmitere\_chei(void \*arg) – se apeleaza exact dupa alegerea modului de operare si se transmit mai intai cheile si vectorul de initializare criptate cu K3;apoi serverul primeste un mesaj de confirmare de la client, criptat cu cheia primita anterior. La acest pas au intervenit niste eori, care nu am reusit sa descopar de unde apar. Din acest motiv nu am reusit sa continuu cu initierea comunicatiei si citirea din fisier. 

Fac diferentierea intre cheile pe care trebuie sa le transmit cu ajutorul varabilei *nr\_optiune*, pe care o primesc de la clientul A..siin baza ei ma orientez ce moduri de operare vor folosi clientii.

int cripteaza\_cheia(unsigned char \*plaintext, int plaintext\_len, unsigned char \*key,unsigned char \*ciphertext ) – aceasta functie o folosesc pentru criptarea AES, in mod ECB pe 128 biti a cheilor K1, K2 si a IV cu ajutorul cheii K3. Aceeasi functie o folosesc si in clientii A si B.

int decriptare\_chei(unsigned char \*ciphertext, int ciphertext\_len, unsigned char \*key,unsigned char \*plaintext) - aceasta functie o folosesc pentru decriptarea AES, in mod ECB pe 128 biti a cheilor K1, K2 si a IV cu ajutorul cheii K3. Aceeasi functie o folosesc si in clientii A si B.

unsigned char \*\*impartire\_blocuri(unsigned char \*mesaj, int nr\_blocuri) – functie comuna atat pentru SERVER, cat si pentru clientii A, B. Intentionam sa o folosesc la impartirea pe blocuri a mesajului citit din fisier, desigur incluzand si padarea. Padarea presupunea ca intr-un bloc incomplet (strlen(bloc)<16) adaugam la acesta un caracter char care corespundea nr de spatii insuficiente in bloc. In cazurile cand ultimul bloc era gol (cazul cand lungimea mesajului e multiplu de 16) mai adaugam un bloc pur de padare cu unsigned char(16). Astfel cand as fi ajuns la depadare trebuia doar sa citesc ultimul caracter, sa-l castez la int si aflu cate caractere trebuie sa elimin ca sa obtin blocul de mesaj.

unsigned char \*xor\_function(unsigned char \*plaintext, unsigned char \*initialization\_vector) - functia xor pe care o folosesc la implementarea modului de operare CBC si CFB

int cripteaza\_cheia\_cbc(unsigned char \*plaintext, int plaintext\_len, unsigned char \*key,

unsigned char \*iv, unsigned char \*ciphertext) – modul de criptare cbc pe 128 biti cu criptosistemul AES. Aceeasi functie o folosesc si in clientii A si B

int decriptare\_cheia\_cbc(unsigned char \*ciphertext, int ciphertext\_len, unsigned char \*key,

unsigned char \*iv, unsigned char \*plaintext)- modul de decriptare cbc pe 128 biti cu criptosistemul AES. Aceeasi functie o folosesc si in clientii A si B

int cripteaza\_cheia\_cfb(unsigned char \*plaintext, int plaintext\_len, unsigned char \*key,

unsigned char \*iv, unsigned char \*ciphertext) – modul de criptare cfb pe 128 biti cu criptosistemul AES. Aceeasi functie o folosesc si in clientii A si B

int decriptare\_cheia\_cfb(unsigned char \*ciphertext, int ciphertext\_len, unsigned char \*key,

unsigned char \*iv, unsigned char \*plaintext)- modul de decriptare cfb pe 128 biti cu criptosistemul AES. Aceeasi functie o folosesc si in clientii A si B

Functii CLIENT A/B:

Multe din functiile si modul de lucru al acestora a fost deja descris printre functiile serverului. In clientul A, am indicat o incercare de a implementa modul de criptare cfb si cbc.

void cbc1(unsigned char \*\*blocuri, int nr\_blocuri, unsigned char \*key, unsigned char \*iv, unsigned char \*cbc\_final) – primeste ca parametru blocurile de mesaj impartie cu ajutorul functiei impartire blocuri si apeleaza pentru cate un bloc functia *implementare\_cbc*, care realizeaza criptarea pe blocul pe care s-a fc deja XOR cu VI/ ciphertextul de la pasul anterior

Tin sa mentionez ca in functiile cu criptare mod ecb, cu ajutorul criptosistemului AES am adaugat linia

*EVP\_CIPHER\_CTX\_set\_padding(ctx, 0); -* ceea ce imi elimina padarea by default a ecb si am nevoie de acest lucru atunci cand implementez cfb/cbc, ca atunci cand apelez criptarea ecb a blocurilor padate de mine sa nu am o padare in plus din partea ecb-ului.