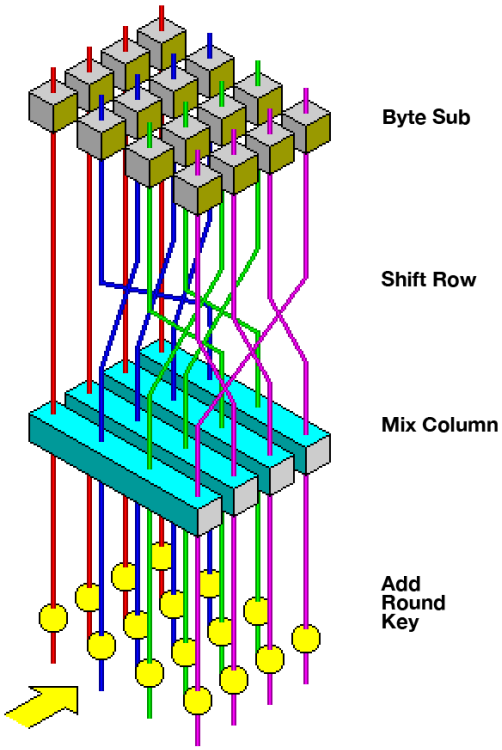
# Sistem distribuit de comunicare intre statii (Socket), de transmitere point-to-point a mesajelor criptate utilizand algoritmul simetric de criptare AES (Advanced Encryption Standard)




Proiect realizat de: Bors Bogdan,
Chirila Andrei
1408A

Proiectul ales (ID#3) a fost sistemul distribuit de comunicare intre statii (folosind Socket, utilizand algoritmul de criptare simetric Advanced Encryption Standard /Rijndael). In cadrul documentatiei sunt prezentate snippet-uri de cod pentru algoritm in ambele sensuri (encryption/decryption) si metodei de comunicare folosita si schema proiectului

```
298
299  void AES(unsigned char* message, unsigned char* key)
300  {
301      unsigned char state[16];
302      for (int i = 0; i < 16; i++)
303          state[i] = message[i];
304
305      int nrRounds = 9;
306      unsigned char expandedKey[176];
307      KeyExpansion(key, expandedKey);
308      //InitialRound();
309      AddRoundKey(state, key);
310
311      for (int i = 0; i < nrRounds; i++)
312      {
313          SubBytes(state);
314          ShiftRows(state);
315          MixColumns(state);
316          AddRoundKey(state, expandedKey + (16 * (i + 1)));
317      }
318      SubBytes(state);
319      ShiftRows(state);
320      AddRoundKey(state, expandedKey + 160);
321
322      for (int i = 0; i < 16; i++)
323          message[i] = state[i];
324  }
325
```

```
400  void AES_dec(unsigned char* message, unsigned char* key)
401  {
402      unsigned char state[16];
403      for (int i = 0; i < 16; i++)
404          state[i] = message[i];
405
406      int nrRounds = 9;
407      //InitialRound();
408      AddRoundKey(state, key + 160);
409
410      for (int i = nrRounds; i > 0; i--)
411      {
412          //InvSubBytes(state);
413          InvShiftRows(state);
414          InvSubBytes(state);
415          AddRoundKey(state, key + (16 * i));
416          InvMixColumns(state);
417      }
418      InvShiftRows(state);
419      InvSubBytes(state);
420      AddRoundKey(state, key);
421
422      for (int i = 0; i < 16; i++)
423          message[i] = state[i];
424  }
```

Algoritmul simetric urmareste o structura similara in ambele sensuri:

Key -> Expansiunea cheii -> Runda initiala (in cazul nostru, AddRoundKey) -> Rundele care se repeta (in cazul nostru, avem 10 cicluri, pentru un mesaj de 128 de biti, cea mai rapida comparativ cu variantele de 192/256 biti si 12/14 cicluri, respectiv) -> Runda finala

Rundele care se repeta, la randul lor, sunt impartite in 4 operatii (SubBytes, ShiftRows, MixColumns, AddRoundKey), iar runda finala in 3 (SubBytes, ShiftRows, AddRoundKey)

SubByte:

```
void SubBytes(unsigned char* state)
{
    for (int i = 0; i < 16; i++)
    {
        state[i] = s[state[i]];
    }
}
```

*fiecare byte e schimbat cu un alt byte in functie de cheie utilizand un tabel numit Rijndael S-Box, de dimensiune 16x16. Tabelul poate fi generat, dar noi am ales sa-l preluam direct de pe Wikipedia

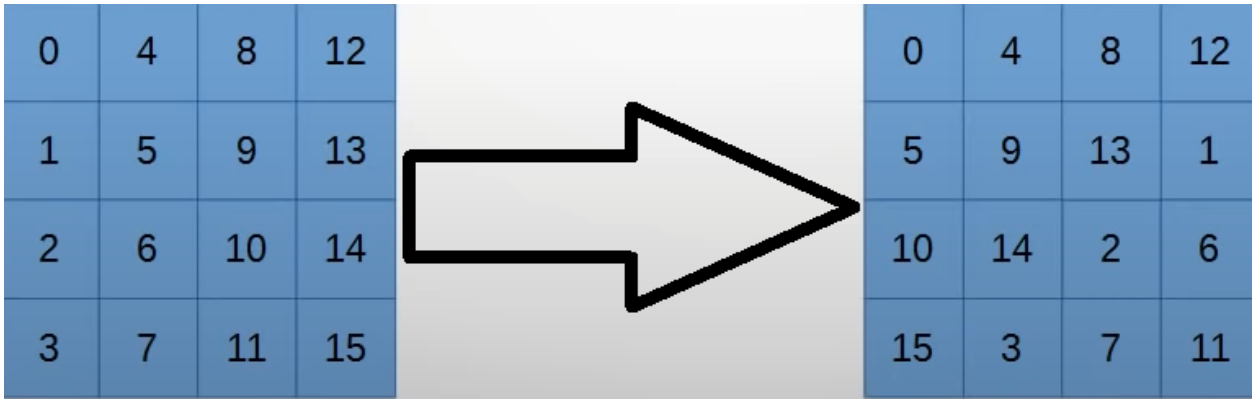
ShiftRows:

```
void ShiftRows(unsigned char* state)
{
    unsigned char tmp[16];
    //tmp[0] = state[0];
    //tmp[1] = state[5];
    //tmp[2] = state[10];
    //tmp[3] = state[15];

    for (int i = 0; i < 16; i++)
    {
        tmp[i] = state[(i + i % 4 * 4) % 16];
    }

    for (int i = 0; i < 16; i++)
    {
        state[i] = tmp[i];
    }
}
```

*rotim la stanga fiecare linie cu 0, 1, 2, respectiv 3 pozitii. Poza pentru vizualizare a operatiei, before&after (ca si pozitii!):




AddRoundKey:

```
void AddRoundKey(unsigned char* state, unsigned char* roundKey)
{//Galois(?) Fields e XOR; alternativ, x OPER. y % BAZA
    for (int i = 0; i < 16; i++)
    {
        state[i] ^= roundKey[i];
    }
}
```

*”Adunam” state + cheie folosind o aritmetica diferita (Galois Fields, sau corpuri Galois). In cazul nostru e un XOR (deoarece MOD 2!!!)

MixColums:

```
void MixColumns(unsigned char* state)
{
    //2 3 1 1
    //1 2 3 1
    //1 1 2 3
    //3 1 1 2
    unsigned char tmp[16];

    tmp[0] = (unsigned char)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);
    tmp[1] = (unsigned char)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);
    tmp[2] = (unsigned char)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);
    tmp[3] = (unsigned char)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

    tmp[4] = (unsigned char)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);
    tmp[5] = (unsigned char)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);
    tmp[6] = (unsigned char)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);
    tmp[7] = (unsigned char)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

    tmp[8] = (unsigned char)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);
    tmp[9] = (unsigned char)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);
    tmp[10] = (unsigned char)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);
    tmp[11] = (unsigned char)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

    tmp[12] = (unsigned char)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);
    tmp[13] = (unsigned char)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);
    tmp[14] = (unsigned char)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);
    tmp[15] = (unsigned char)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

    for (int i = 0; i < 16; i++)
        state[i] = tmp[i];
}
```

*folosim tabele (mul2 si mul3) preluate de pe Wikipedia si utilizam o matrice predefinita (la noi, notata in comentarii) pentru realizarea operatiei (XOR element cu element). Precizam ca in teorie, multiplicarea corpurilor Galois, reducerea lor polinomiala, “adaugarea” lor, inmultirea a doua polinoame (poate) care rezulta in depasirea unui byte, motiv pentru care procedura initiala foloseste reducerea mod x^8+x^4+x^3+x^1+x^0 pana cand se ajunge la rezultatul dorit (<mod 100011011) si se

retine, etc sunt operatii costisitoare, motiv pentru care folosim matrici predefinite

**KeyExpansionCore:

```
void KeyExpansionCore(unsigned char* in, unsigned char i)
{
    unsigned int* q = (unsigned int*)in;
    // Left rotate bytes
    *q = (*q >> 8 | ((*q & 0xff) << 24));

    //s-box

    in[0] = s[in[0]];
    in[1] = s[in[1]];
    in[2] = s[in[2]];
    in[3] = s[in[3]];

    in[0] ^= rcon[i];
}
```

*3 operatii: Rotatia la stanga, schimbarea fiecarui byte cu cea corespunzatoare valorii din S-Box, xor cu Rcon - o alta tabela preluata de pe Wikipedia (in realitate avem nevoie de doar primele 10 sau 11 valori pentru codul folosit)

KeyExpansion:

```
void KeyExpansion(unsigned char* inputKey, unsigned char* expandedKeys)
{
    //og key
    for (int i = 0; i < 16; i++)
    {
        expandedKeys[i] = inputKey[i];
    }

    int bytesGenerated = 16;
    int rconIteration = 1;
    unsigned char temp[4];

    while (bytesGenerated < 176)
    {
        for (int i = 0; i < 4; i++)
            temp[i] = expandedKeys[i + bytesGenerated - 4];


        if (bytesGenerated % 16 == 0)
        {
            KeyExpansionCore(temp, rconIteration); //rconIteration++
            rconIteration++;
        }

        for (unsigned char a = 0; a < 4; a++)
        {
            expandedKeys[bytesGenerated] = expandedKeys[bytesGenerated - 16] ^ temp[a];
            bytesGenerated++;
        }
    }
}
```

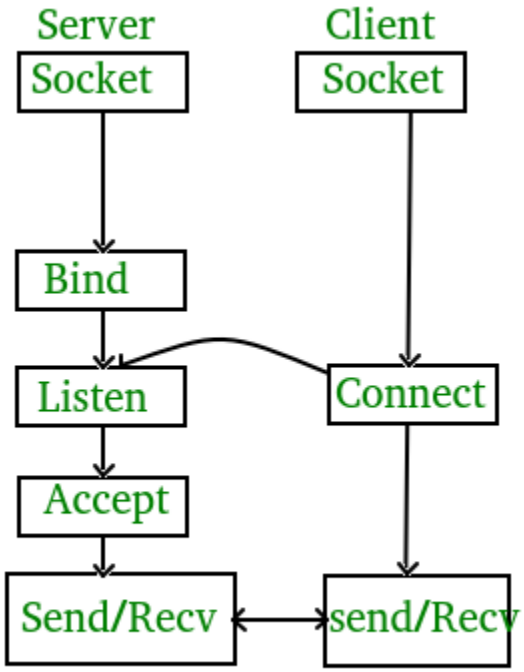
*primii 16 bytes sunt cheia originala. Tinem cont ca suntem la iteratia 1, citim primii 4 bytes si dupa apelam KeyExpansionCore cand atingem conditia, altfel continuam expansiunea si incrementam nr de bytes generati. Am realizat ca ar fi fost mai bine sa facem aceasta operatie in afara algoritmului de encriptie propriu zis abia la final

Operatiile InvShiftRows, InvSubBytes si InvMixColumns sunt asemanatoare, atata doar ca pentru

InvSubBytes se foloseste tabela S-Box inversata, InvShiftRows urmareste procesul invers pentru rotatie (ie 9 devine 5, in loc de 5 devine 9), iar InvMixColumns foloseste tabelele mul14, mul11, mul13, mul9 (toate tabelele fiind preluate de pe Wikipedia) pentru a realiza operatia matriceala

De asemenea, mesajului initial i se face padding (adaugam bytes de 0) pana cand devine un bloc de dimensiunea dorita (16 biti) pentru prelucrare

Conexiunea client/server este una simpla si urmeaza urmatoarea schema:




Cu mentiunea ca detectam adresa clientului folosind functia inet_ntoa si o afisam (dupa accept).