Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и радиоэлектроники

Кафедра РТС

Отчет по лабораторной работе №6

# «ИССЛЕДОВАНИЕ СВОЙСТВ БЛОЧНЫХ КРИПТОСИСТЕМ»

Выполнил:

ст.гр.240102
shlom41k

Проверил:

Панькова В.В.

Минск 2015

# ***Алгоритм AES***

```
> restart;
```

### *Предварительные преобразования*

Для зашифрования данных мы основываемся на байтах. Для дальнейшей работы мы должны описать 8 битов байта с разных сторон: как целое число от 0 до 256; как строка из 8 битов в двоичном коде; как список из 8 бит; как полином наибольшей степени 7 от альфа и как число, предсталвенное в шестнадцатиричной форме.

### *Правила преобразования типов данных*

Преобразование числа в строку из 8 битов

```
> intToBits := intValue -> substring(convert(convert(intValue+256, binary), string), 2..9):
```

Преобразование строки в список битов

```
> bitToList :=  bitWord ->
  [seq(parse(substring(bitWord,i)), i=1..8)]:
```

Преобразование списка битов в полином

```
> listToPoly := bitList -> sort(sum(bitList[j]*alpha^(8-j), j=1..8)):
```

Преобразование полинома в десятичное число

```
> polyToInt := poly -> subs(alpha=2, poly):
```

Преобразование десятичного числа в шестнадцатиричное

```
> intToHex := intValue ->
   substring(convert(convert(intValue+256, hex), string), 2..3):
```

Преобразование шестнадцатиричного числа в двоичную форму

```
> hexTo8Bits := hexPair -> substring(convert(convert(
   convert(hexPair, decimal, hex)+256,binary),string),2..9):
```

Преобразование списка битов в строку

```
> listToBits := bitList -> cat(seq(convert(bitList[i],string),i=1..8)):
```

Преобразование строки в десятичное число

```
> bitToInt := bitWord -> convert(bitWord,decimal,binary):
```

Преобразование списка битов в десятичное число

```
> listToInt := bitList -> sum(bitList[j]*2^(8-j), j=1..8):
```

Преобразование полинома в список битов

```
> polyToList := poly -> [seq(coeff(poly, alpha, 8-j), j=1..8)]:
```

ВЗАИМОСВЯЗЬ ПРЕОБРАЗОВАНИЙ

```
> polyToBits := poly -> intToBits(polyToInt(poly)):
> intToPoly := intValue -> listToPoly(bitToList(intToBits(intValue))):
> bitToPoly := bitWord -> listToPoly(bitToList(bitWord)):
```

### *Преобразования между списками и матрицами*

```
> listToMatrix := list -> matrix(4,4,list):
```

```
> listToMatrix2 := list -> linalg[transpose](matrix(4,4,list)):
> matrixToList := matriX ->ListTools[Flatten](convert(matriX,listlist)):
> matrixToList2 := matriX -> ListTools[Flatten](convert(linalg[transpose](matriX),listlist)):
> matrixToHex := bitMatrix -> cat(op(matrixToList2(map(x ->
  intToHex(bitToInt(x)),bitMatrix)))):
```

### *Константы*

Представление байта в виде полинома основывается на байтах как элементах конечного поля GF(256). Конструкция этого поля требует неприводимого полинома степени 8.

```
> genPoly := alpha^8 + alpha^4 + alpha^3 + alpha + 1:
> XOR := (a,b) -> if (a=b) then "0" else "1" fi:
xorNbits := proc(a,b,N)
  local aString, bString:
  aString := convert(a,string):
  bString := convert(b,string):
  cat(seq(XOR(substring(aString,i), substring(bString,i)),i=1..N)):
end:
xor8 := (a,b) -> xorNbits(a,b,8):
```

### *Создание S-блока*

Рассмотрим конструкцию S-блока (или таблицы подстановки), который используется в операции замены байтов *SubBytes*( ). Таблицы замены S-блока являются инвертируемыми и построены из композиции следующих двух преобразований входного байта:

- получение обратного элемента относительно умножения в поле *GF(256)* с использованием расширенного алгоритма Эвклида; нулевой элемент '0' переходит сам в себя;
- применение следующего преобразования: умножаем некоторую циклическую матрицу на инвертированный байт и прибавляем некоторый вектор (матрица и вектор изначально определены для AES).

```
> s1Make := proc(intValue)
  local bitList, invPoly, extraPoly, polyValue;
  bitList := bitToList(intToBits(intValue)):
  if (intValue <> 0) then #invert nonzero entries.
    polyValue := listToPoly(bitList):
    Gcdex(polyValue, genPoly, alpha, 'invPoly', 'extraPoly') mod 2;
    bitList := (polyToList(invPoly)):
  end if:
  bitList;
end:
> s2Make := proc(bitList)  local shift1List, shift2List;
  shift1List := [1,1,1,1,1,0,0,0,1,1,1,1,1,0,0,0];
  shift2List := [0,1,1,0,0,0,1,1];
  [seq(linalg[dotprod](bitList,[seq(shift1List[9+i-j],i=1..8)])
       +shift2List[j] mod 2, j=1..8)];
end:
> sMakeIBits := intVal -> listToBits(s2Make(s1Make(intVal))):
```

Представление в виде таблицы

```
> SBoxTable := table([seq(intToBits(i)=sMakeIBits(i), i=0..255)]):
```

Представление таблицы в виде матрицы 16x16

```
> SBoxMatrix1 := matrix(16,16,[seq(SBoxTable[intToBits(i)],i=0..255)]):
```

Представление полученных элементов в виде двоичных чисел

```
> SBoxMatrix2 := map( bitToInt,SBoxMatrix1);
```

$$SBoxMatrix2 :=$$

$$\begin{bmatrix}
99 & 124 & 119 & 123 & 242 & 107 & 111 & 197 & 48 & 1 & 103 & 43 & 254 & 215 & 171 & 118 \\
202 & 130 & 201 & 125 & 250 & 89 & 71 & 240 & 173 & 212 & 162 & 175 & 156 & 164 & 114 & 192 \\
183 & 253 & 147 & 38 & 54 & 63 & 247 & 204 & 52 & 165 & 229 & 241 & 113 & 216 & 49 & 21 \\
4 & 199 & 35 & 195 & 24 & 150 & 5 & 154 & 7 & 18 & 128 & 226 & 235 & 39 & 178 & 117 \\
9 & 131 & 44 & 26 & 27 & 110 & 90 & 160 & 82 & 59 & 214 & 179 & 41 & 227 & 47 & 132 \\
83 & 209 & 0 & 237 & 32 & 252 & 177 & 91 & 106 & 203 & 190 & 57 & 74 & 76 & 88 & 207 \\
208 & 239 & 170 & 251 & 67 & 77 & 51 & 133 & 69 & 249 & 2 & 127 & 80 & 60 & 159 & 168 \\
81 & 163 & 64 & 143 & 146 & 157 & 56 & 245 & 188 & 182 & 218 & 33 & 16 & 255 & 243 & 210 \\
205 & 12 & 19 & 236 & 95 & 151 & 68 & 23 & 196 & 167 & 126 & 61 & 100 & 93 & 25 & 115 \\
96 & 129 & 79 & 220 & 34 & 42 & 144 & 136 & 70 & 238 & 184 & 20 & 222 & 94 & 11 & 219 \\
224 & 50 & 58 & 10 & 73 & 6 & 36 & 92 & 194 & 211 & 172 & 98 & 145 & 149 & 228 & 121 \\
231 & 200 & 55 & 109 & 141 & 213 & 78 & 169 & 108 & 86 & 244 & 234 & 101 & 122 & 174 & 8 \\
186 & 120 & 37 & 46 & 28 & 166 & 180 & 198 & 232 & 221 & 116 & 31 & 75 & 189 & 139 & 138 \\
112 & 62 & 181 & 102 & 72 & 3 & 246 & 14 & 97 & 53 & 87 & 185 & 134 & 193 & 29 & 158 \\
225 & 248 & 152 & 17 & 105 & 217 & 142 & 148 & 155 & 30 & 135 & 233 & 206 & 85 & 40 & 223 \\
140 & 161 & 137 & 13 & 191 & 230 & 66 & 104 & 65 & 153 & 45 & 15 & 176 & 84 & 187 & 22
\end{bmatrix}$$

```
> SBoxMatrix3 := map( intToHex,SBoxMatrix2);
```

*SBoxMatrix3* :=

["63", "7C", "77", "7B", "F2", "6B", "6F", "C5", "30", "01", "67", "2B", "FE", "D7", "AB", "76"]
["CA", "82", "C9", "7D", "FA", "59", "47", "F0", "AD", "D4", "A2", "AF", "9C", "A4", "72", "C0"]
["B7", "FD", "93", "26", "36", "3F", "F7", "CC", "34", "A5", "E5", "F1", "71", "D8", "31", "15"]
["04", "C7", "23", "C3", "18", "96", "05", "9A", "07", "12", "80", "E2", "EB", "27", "B2", "75"]
["09", "83", "2C", "1A", "1B", "6E", "5A", "A0", "52", "3B", "D6", "B3", "29", "E3", "2F", "84"]
["53", "D1", "00", "ED", "20", "FC", "B1", "5B", "6A", "CB", "BE", "39", "4A", "4C", "58", "CF"]
["D0", "EF", "AA", "FB", "43", "4D", "33", "85", "45", "F9", "02", "7F", "50", "3C", "9F", "A8"]
["51", "A3", "40", "8F", "92", "9D", "38", "F5", "BC", "B6", "DA", "21", "10", "FF", "F3", "D2"]
["CD", "0C", "13", "EC", "5F", "97", "44", "17", "C4", "A7", "7E", "3D", "64", "5D", "19", "73"]
["60", "81", "4F", "DC", "22", "2A", "90", "88", "46", "EE", "B8", "14", "DE", "5E"

, "0B", "DB"]
["E0", "32", "3A", "0A", "49", "06", "24", "5C", "C2", "D3", "AC", "62", "91", "95", "E4", "79"]
["E7", "C8", "37", "6D", "8D", "D5", "4E", "A9", "6C", "56", "F4", "EA", "65", "7A", "AE", "08"]
["BA", "78", "25", "2E", "1C", "A6", "B4", "C6", "E8", "DD", "74", "1F", "4B", "BD", "8B", "8A"]
["70", "3E", "B5", "66", "48", "03", "F6", "0E", "61", "35", "57", "B9", "86", "C1", "1D", "9E"]
["E1", "F8", "98", "11", "69", "D9", "8E", "94", "9B", "1E", "87", "E9", "CE", "55", "28", "DF"]
["8C", "A1", "89", "0D", "BF", "E6", "42", "68", "41", "99", "2D", "0F", "B0", "54", "BB", "16"]

Инверсный S-блок

```
> InvSBoxTable := table([seq(sMakeIBits(i)=intToBits(i), i=0..255)]):
> InvSBoxMatrix1 := matrix(16,16,[seq(InvSBoxTable[intToBits(i)],i=0..255)]):
> InvSBoxMatrix2 := map( bitToInt,InvSBoxMatrix1):
> InvSBoxMatrix3 := map( intToHex,InvSBoxMatrix2):
```

### *Алгоритм выработки ключей*

Раундовый ключ получается из ключа шифра посредством таблицы ключей. Она состоит из расширения ключа и выбора раундового ключа. Основной принцип заключается в следующем:
- суммарное число битов раундового ключа равно длине блока, умноженной на число раундов плюс 1 (т.е. для длины блока 128 бит и 10 раундов для раундового ключа необходимо 1408 бит);
- ключ шифра расширяется до расширенного ключа;
- раундовые ключи выбираются из этого расширенного ключа следующим образом: первый раундовый ключ состоит из первых Nb слов, второй - из следующих Nb слов и т.д.

### *Расширение ключа*

Расширенный ключ - это одномерный массив четырехбайтовых слов, обозначаемый W[Nb*(Nr+1)]. Первые Nk слов содержат ключ шифра. Все остальные слова определяются рекурсивно через слова с меньшими индексами.

```
> roundFudge := int -> Rem(alpha^(int-1),genPoly,alpha) mod 2:
polyToInt := poly -> subs(alpha=2, poly):
roundFudgeWord := int -> polyToBits(roundFudge(int)):
randKeyGenerator := () ->
  map(intToBits, [seq(rand(0..255)(),i=1..16)]):
hex32ToKey: hexWord ->map(hexTo8Bits,
   [seq(substring(hexWord,2*i-1..2*i),i=1..16)]):
> keyExpander := proc(keyList)
  local keyExpanded, i, j, k, fudgeWord:
  keyExpanded := matrix(4,44):
  for j from 1 to 4 do
    for i from 1 to 4 do
      keyExpanded[i,j] := keyList[(j-1)*4+i];
    end do:
  end do:
  for i from 1 to 10 do
```

```
    fudgeWord := roundFudgeWord(i);
    keyExpanded[1,4*i+1] :=
      xor8(keyExpanded[1,4*i-3],SBoxTable[keyExpanded[2,4*i]]);
    keyExpanded[2,4*i+1] :=
      xor8(keyExpanded[2,4*i-3],SBoxTable[keyExpanded[3,4*i]]);
    keyExpanded[3,4*i+1] :=
      xor8(keyExpanded[3,4*i-3],SBoxTable[keyExpanded[4,4*i]]);
    keyExpanded[4,4*i+1] :=
      xor8(keyExpanded[4,4*i-3],SBoxTable[keyExpanded[1,4*i]]);
    keyExpanded[1,4*i+1] := xor8(keyExpanded[1,4*i+1],fudgeWord);
    for j from 2 to 4 do
      for k from 1 to 4 do
        keyExpanded[k,4*i+j]
          :=xor8(keyExpanded[k,4*i+j-4],keyExpanded[k,4*i+j-1]):
      end do:
    end do:
  end do:
  keyExpanded;
end:
```

***Раундовое преобразование***

Раундовое преобразование состоит из четырех различных преобразований: замены байтов*SubBytes*( ), сдвига строк *ShiftRows*( ), перемешивания столбцов *MixColumns*( ) и добавления раундового ключа *AddRoundKey*( ).

Преобразование *SubBytes*( ) - это нелинейная замена байтов, применяемая отдельно в каждому байту состояния. В этом случае используется таблица подстановок, созданная выше. Инверсная операция использует инверсную таблицу замен.

```
> SubBytes := byteMatrix -> map(x->SBoxTable[x], byteMatrix):
InvSubBytes := byteMatrix -> map(x->InvSBoxTable[x], byteMatrix):
```

При *ShiftRows*( ) строки Состояния циклически сдвигаются на различное число байтов. Строка 0 не сдвигается, строка 1 сдвигается на 1 байт, строка 2 - на 2 байта и строка 3 - на 3 байта. Инверсная операция производит сдвиг 4-х строк на 0, 3, 2 и 1 байт соответственно.

```
> ShiftRows := proc(byteMatrix)
local bytelist, rotlist:
byteList := convert(byteMatrix,listlist):
  rotList :=[byteList[1], ListTools[Rotate](byteList[2],1),
     ListTools[Rotate](byteList[3],2), ListTools[Rotate](byteList[4],3)]:
  convert(rotList, matrix);
end:

InvShiftRows := proc(byteMatrix)
local byteList, rotList:
  byteList := convert(byteMatrix,listlist):
  rotList :=[byteList[1], ListTools[Rotate](byteList[2],3),
     ListTools[Rotate](byteList[3],2), ListTools[Rotate](byteList[4],1)]:
  convert(rotList, matrix);
end:
```

```
Warning, `byteList` is implicitly declared local to procedure `ShiftRows`
Warning, `rotList` is implicitly declared local to procedure `ShiftRows`
```

При *MixColumns*( ) столбцы Состояния рассматриваются как многочлены над GF(256) и умножаются на фиксированную перемешивающую матрицу. Инверсной операции соответствует умножение на инверсную перемешивающую матрицу.

```
> MixMatrix := map(intToPoly,
   matrix(4,4,[2,3,1,1,1,2,3,1,1,1,2,3,3,1,1,2])):
MixColumns := proc(byteMatrix)
  local product1, polyMatrix:
  polyMatrix := map(bitToPoly, byteMatrix):
  product1 := linalg[multiply](MixMatrix,polyMatrix):
        map(x->polyToBits(sort(Rem(expand(x),genPoly,alpha)mod 2)), product1);
end:

InvMixMatrix := map(intToPoly,
  matrix(4,4,[14,11,13,9,9,14,11,13,13,9,14,11,11,13,9,14])):
InvMixColumns := proc(byteMatrix)
  local product1, polyMatrix:
  polyMatrix := map(bitToPoly, byteMatrix):
  product1 := linalg[multiply](InvMixMatrix,polyMatrix):
        map(x->polyToBits(sort(Rem(expand(x),genPoly,alpha)mod 2)), product1);
end:
```

Операция добавления раундового ключа *AddRoundKey*( ) выполняет поразрядное сложение XOR текущего состояния матрицы с рандовыми ключом. Обратная операция такая же.

```
> AddRoundKey := proc(byteMatrix, expandedKey, roundNum)
  local roundKey:
 roundKey := linalg[transpose]
(matrix([linalg[col](expandedKey,(roundNum*4+1)..(roundNum*4+4))])):
  zip(xor8,byteMatrix,roundKey);
end:
```

***Задание сообщения и ключа***

```
> Message:= "0123456789ACBDEFFEDBCA98765432100";
HexList := [seq(substring(Message,2*i-1..2*i),i=1..16)];
```

*Message* := "0123456789ACBDEFFEDBCA9876543210

*HexList* := ["01", "23", "45", "67", "89", "AC", "BD", "EF", "FE", "DB", "CA", "98", "76", "54", "32", "10"]

```
> ByteList := map(hexTo8Bits, HexList);
messMatrix := listToMatrix2(ByteList);
```

*ByteList* := ["00000001", "00100011", "01000101", "01100111", "10001001", "10101100", "10111101", "11101111", "11111110", "11011011", "11001010", "10011000", "01110110", "01010100", "00110010", "00010000"]

$$messMatrix := \begin{bmatrix} "00000001" & "10001001" & "11111110" & "01110110" \\ "00100011" & "10101100" & "11011011" & "01010100" \\ "01000101" & "10111101" & "11001010" & "00110010" \\ "01100111" & "11101111" & "10011000" & "00010000" \end{bmatrix}$$

```
> Key := ["00000001", "00100011", "01000101", "01100111", "10001001", "10101011",
"11001101", "11101111", "00000001", "00100011", "01000101", "01100111", "10001001",
"10101011", "11001101", "11101111"];
```

```
Key := ["00000001", "00100011", "01000101", "01100111", "10001001", "10101011",
        "11001101", "11101111", "00000001", "00100011", "01000101", "01100111", "10001001",
        "10101011", "11001101", "11101111"]
```

```
> KeyExpanded := keyExpander(Key);
```

$$KeyExpanded := keyExpanded$$

***Зашифрование и расшифрование раунд за раундом***

```
> cipher := AddRoundKey(messMatrix, KeyExpanded, 0):
 hexState := matrixToHex(cipher):
> for i from 1 to 9 do
  cipher := AddRoundKey(MixColumns(ShiftRows(SubBytes(cipher))),KeyExpanded,i);
  hexState := matrixToHex(cipher);
end do:

> cipher := AddRoundKey(ShiftRows(SubBytes(cipher)),KeyExpanded,10):
cipherHex := matrixToHex(cipher):
```

***Расшифрование сообщения***

```
> ListCipher := [seq(substring(cipherHex,2*i-1..2*i), i = 1..16)]:
cipherByteMatrix := map(hexTo8Bits,listToMatrix2(ListCipher)):
```

Теперь мы отменяем 10 раундов шифрования.

```
> plain := cipherByteMatrix:
plain := InvSubBytes(InvShiftRows(AddRoundKey(plain,
KeyExpanded,10))):
for i from 1 to 9 do
plain := InvSubBytes(InvShiftRows(InvMixColumns(AddRoundKey(plain,KeyExpanded,10-
i))));
end do:
decryptMatrix := AddRoundKey(plain, KeyExpanded, 0):
```

Теперь представим сообщение обратно в шестнадцатиричную форму, чтобы сравнить его с оригинальным

```
> matrixToHex(decryptMatrix):
```

***Зашифрование и расшифрование текста ACSII***

```
> encryptAESascii := proc(message, keyList)
  local expandedKey, cipher, cipherHexList, cipherHex, i,messMatrix:
  expandedKey := keyExpander(keyList):
  messMatrix := listToMatrix2(map(intToBits, convert(message,bytes))):
  cipher := AddRoundKey(messMatrix, expandedKey, 0):
  for i from 1 to 9 do
    cipher := AddRoundKey(MixColumns(ShiftRows(SubBytes(cipher))),expandedKey,i):
```

```
  end do:
  cipher := AddRoundKey(ShiftRows(SubBytes(cipher)),expandedKey,10):
  cipherHexList := matrixToList2(
     map(x -> intToHex(bitToInt(x)),cipher)):
  cipherHex := cat(seq(cipherHexList[i],i=1..16)):
end:

> decryptAESascii := proc(cipherText, keyList)
  local expandedKey, ListCipher, cipherByteMatrix, plain,
    i,decryptMatrix, decryptList:
  expandedKey := keyExpander(keyList):
  ListCipher := [seq(substring(cipherText,2*i-1..2*i), i = 1..16)];
  cipherByteMatrix := map(hexTo8Bits,listToMatrix2(ListCipher));
  plain := cipherByteMatrix;
  plain := InvSubBytes(InvShiftRows(AddRoundKey(plain, expandedKey,10)));
  for i from 1 to 9 do
    plain :=
InvSubBytes(InvShiftRows(InvMixColumns(AddRoundKey(plain,expandedKey,10-i))));
  end do;
  decryptMatrix := AddRoundKey(plain, expandedKey, 0);
  decryptList := matrixToList2(map(bitToInt,decryptMatrix));
  convert(decryptList,bytes);
end:

> mess := "HAVE A NICE DAY!";
  testKey1 := ["10100111", "00000101", "01011000", "11011011", "01111101", "01001100",
"00100001", "00111111", "00000100", "01110010", "10110110", "01011011", "00100100",
"00110001", "01101100", "00101111"];
  cipherText := encryptAESascii(mess, testKey1);
  decryptAESascii(cipherText, testKey1);
```

*mess* := "HAVE A NICE DAY!'

*testKey1* := ["10100111","00000101","01011000","11011011","01111101","01001100",
"00100001","00111111","00000100","01110010","10110110","01011011","00100100",
"00110001","01101100","00101111"]

*cipherText* := "E6B48BC66FA892756DB9FD254D69A4D
"HAVE A NICE DAY!'

```
> messMatrix := listToMatrix2(map(intToBits, convert(mess,bytes))):
> KeyExpanded := keyExpander(testKey1):
> cipher := AddRoundKey(messMatrix, KeyExpanded, 0):
 hexState := matrixToHex(cipher):
> for i from 1 to 9 do
    cipher :=AddRoundKey(MixColumns(ShiftRows(SubBytes(cipher))),KeyExpanded,i);
    hexState := matrixToHex(cipher);
  end do:

> cipher := AddRoundKey(ShiftRows(SubBytes(cipher)),KeyExpanded,10):
cipherHex := matrixToHex(cipher):
```

***Расчет спектра***

```
> s:=rand(1..155):
```

```
> m:=s();
```

$$m := 93$$

```
> for i from 1 to 155 do kl[i]:=randKeyGenerator();od:
> #for i from 1 to 10 do k[i]:=convert(kl[i],  decimal, binary);od;
> #map(bitToInt,kl[i]):
```

Сообщение из 16 знаков

```
> mess;
```

"HAVE A NICE DAY!"

```
> testKey1:=kl[m];
```

*testKey1* := ["00011101", "10100011", "11000000", "01110000", "00010101", "11011100", "10101100", "11011101", "10110010", "01010010", "11101100", "01110101", "11000110", "00000101", "10101100", "01001111"]

```
> map(bitToInt,testKey1);
```

[29, 163, 192, 112, 21, 220, 172, 221, 178, 82, 236, 117, 198, 5, 172, 79]

```
> cipherText := encryptAESascii(mess, testKey1);
```

*cipherText* := "7DF6295E3FDB3B7350E44AAC9071C12

```
> for j from 1 to 32 do c[j]:= bitToInt(hexTo8Bits(cipherText[j]));od:
> z:=seq(c[i],i=1..32);
```

*z* := 7, 13, 15, 6, 2, 9, 5, 14, 3, 15, 13, 11, 3, 11, 7, 3, 5, 0, 14, 4, 4, 10, 10, 12, 9, 0, 7, 1, 12, 1, 2, 9

```
> for i from 1 to 32 do b[0,i-1]:=z[i]; od:

> for i from 1 to 32 do d[0,i-1]:=8; d[1,i-1]:=4; d[2,i-1]:=2; d[3,i-1]:=1; od:
> for j from 0 to 31 do
  for i from 1 to 4 do
    b[i,j]:=b[i-1,j]-d[i-1,j];
    if b[i,j]<0
      then c[4*j+i]:=0; b[i,j]:=b[i-1,j]
    else c[4*j+i]:=1
    end if;
  end do;
end do;
> seq(c[i],i=1..128);
```

0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1

# ВЫВОД

В данной лабораторной работе были изучены основные свойства алгоритма шифрования AES.