/\*

// public static void main(String[] args)

// {

// //AES-128

// String inputString = "00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff";

// //what the inputString string should turn into.

// String encryptString = "69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a";

// String keyString = "00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f";

//

//

// //AES-192

//// String inputString = "00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff";

//// String encryptString = "dd a9 7c a4 86 4c df e0 6e af 70 a0 ec 0d 71 91";

//// String keyString = "00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17";

//

// //AES-256

//// String inputString = "00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff";

//// String encrpytString = "8e a2 b7 ca 51 67 45 bf ea fc 49 90 4b 49 60 89";

//// String keyString = "00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f";

//

// //first we encode is

// encode(AES.stringToBytes(inputString), AES.stringToBytes(keyString));

// System.out.println("no decrypt yet!!!");

// //decode(AES.stringToBytes(encryptString), AES.stringToBytes(keyString));

// }

//

//

// public static void encode(byte[] stateArray, byte[] keyArray)

// {

// //put the array into a 2d byte array, then the Key too.---------------------------------------------------

// byte[][] state = new byte[stateArray.length/4][4];

// for(int i = 0; i < stateArray.length/4; i++)//column for bits

// {

// for(int j = 0; j < 4; j++)//column row coordinate going down

// {

// state[i][j] = stateArray[(i\*4) + j];

// }

// }

// //quick check

//// for(int i = 0; i < 4; i++)

//// {

//// System.out.println(byteArrayToString(state[i]));//show this column

//// }

//

// //put the key into a n by 4 grid for further use

// byte[][] allKeys = expandKeys(keyArray);

// //you need a unique key per round

// byte[][] key = getKey(0, allKeys);

// int rounds = getNumRounds(keyArray.length);

//

// System.out.println("Encoding " + keyArray.length \* 8);

// System.out.println("round[ 0].input: " + byteArrayToString(stateArray));

// System.out.println("round[ 0].k\_sch: " + byteArrayToString(keyArray));

//

// addRoundKey(state, key);

//

// //all of the rounds we'll do. Depends on how many we need. MUST OUTPUT WHAT ROUND WE ARE ON!!!

// for(int i = 1; i <= rounds; i++)

// {

// printLine(String.format("round[ " + i + "].start: ", i), state);

// //table lookup by hex ref

// state = subBytes(state);

// printLine(String.format("round[ " + i + "].s\_box: ", i), state);

// //shift rows based on row index

// state = shiftRows(state);

// printLine(String.format("round[ " + i + "].s\_row: ", i), state);

// if(i != rounds){

// state = mixColumns(state);

// printLine(String.format("round[ " + i + "].m\_col: ", i), state);

// }

// key = getKey(i, allKeys);

// state = addRoundKey(state, key);

// printLine(String.format("round[ " + i + "].k\_sch: ", i), key);

// }

// String finalState = printLine(String.format("round[%2d].output: ", rounds), state);

// //System.out.println(finalState);

// }

//

// public static void decode(byte[] stateArray, byte[] keyArray)

// {

// byte[][] state = new byte[stateArray.length/4][4];

// for(int i = 0; i < stateArray.length/4; i++)//column for bits

// {

// for(int j = 0; j < 4; j++)//column row coordinate going down

// {

// state[i][j] = stateArray[(i\*4) + j];

// }

// }

//

// //quick check

//// for(int i = 0; i < 4; i++)

//// {

//// System.out.println(byteArrayToString(state[i]));//show this column

//// }

//

// byte[][] allKeys = expandKeys(keyArray);

// int rounds = getNumRounds(keyArray.length);

//

// System.out.println("\nDecoding " + keyArray.length \* 8);

// System.out.println(String.format("round[%d].iinput: ", rounds) + byteArrayToString(stateArray));

// System.out.println(String.format("round[%d].ik\_sch: ", rounds) + byteArrayToString(keyArray));

//

// byte[][] key = getKey(rounds, allKeys);

// addRoundKey(state, key);

//

// //MUST COUNT THE ROUND WE ARE ON!!!

// for(int i = rounds - 1; i >= 0; i--)

// {

// printLine(String.format("round[%2d].istart: ", i), state);

// state = ishiftRows(state);

// printLine(String.format("round[%2d].is\_row: ", i), state);

// state = isubBytes(state);

// printLine(String.format("round[%2d].is\_box: ", i), state);

// key = getKey(i, allKeys);

// printLine(String.format("round[%2d].ik\_sch: ", i), key);

// state = addRoundKey(state, key);

// printLine(String.format("round[%2d].ik\_add: ", i), state);

// if(i != 0){

// imixColumns(state); printLine(String.format("round[%2d].im\_col: ", i), state);

// }

// }

// String finalState = printLine(String.format("round[%2d].ioutput:", 0), state);

// //System.out.println(finalState);

// }

//

// //cause we need this elsewhere

// private static int getNumRounds(int keyLen)

// {

// switch(keyLen){

// case 16:

// return 10;

// case 24:

// return 12;

// case 32:

// return 14;

// default:

// System.out.println("WRONG NUMBER OF KEYS!!!!");

// return 0;

// }

// }

//

// private static byte[][] expandKeys(byte[] key)

// {

// //KeyExpansion(byte key[4\*Nk], word w[Nb\*(Nr+1)], Nk)

// //define Nk, Nr, Nb etc.

// int Nr = getNumRounds(key.length); // number of rounds

// int Nb = 4; // number of rows in each column (words in a block)

// int Nk = key.length / 4; // number of columns

// byte[] temp;

// byte[][] w = new byte[Nb \* (Nr + 1)][4];

//

// int i = 0;

// for(;i < Nk; i++){

// byte[] nxt = {key[4\*i], key[4\*i+1], key[4\*i+2], key[4\*i+3]};

// w[i] = nxt;

// }

// for(i = Nk; i < Nb \* (Nr+1); i++){

// temp = Arrays.copyOf(w[i-1], w[i-1].length);

// if (i % Nk == 0){

// temp = rotWord(temp);

// temp = subWord(temp);

// temp[0] ^= rcon[i/Nk];

// }

// else if (Nk > 6 && i % Nk ==4)

// temp = subWord(temp);

// for(int j = 0; j < temp.length; j++)

// w[i][j] = (byte) (w[i-Nk][j] ^ temp[j]);

// }

// return w;

// }

//

// private static byte[][] getKey(int round, byte[][] allKeys)

// {

//

// return null;

// }

//

// private static byte[] rotWord(byte[] word)

// {

// //rotate the bytes to the left one. Just like in shiftRows

// byte tempy = word[0];

// for(int i = 0; i < 3; i++)

// {

// word[i] = word[i + 1];

// }

// word[3] = tempy;

//

// return word;

// }

//

// private static byte[] subWord(byte[] b)

// {

// //basically a smaller version of subBytes!

// //iterate through the table substituting

// for(int i = 0; i < b.length; i++)

// {

// // get left most 4 bits to hex / get rightmost to hex

// b[i] = sBox[(b[i] & 0XF0) >> 4][b[i] & 0x0F];

// }

//

// return b;

// }

//

// //Bug function 1

// private static byte[][] subBytes(byte[][] state)

// {

// //iterate through the table substituting

// for(int i = 0; i < state.length; i++)

// {

// for(int j = 0; j < state[i].length; j++)

// {

// // get left most 4 bits to hex / get rightmost to hex

// state[i][j] = sBox[(state[i][j] & 0XF0) >> 4][state[i][j] & 0x0F];

// }

// }

// return state;

// }

//

// //Big function 1.5

// private static byte[][] isubBytes(byte[][] state)

// {

// for(int i = 0; i < state.length; i++)

// for(int j = 0; j < state[0].length; j++)

// // get left most 4 bits to hex - get rightmost to hex

// state[i][j] = isBox[(state[i][j] & 0xF0) >> 4][state[i][j] & 0x0F];

// return state;

// }

//

// //Big function 2

// private static byte[][] shiftRows(byte[][] state)

// {

// //you shift all row contents by it's index

// //row1 1 time left

// byte p01 = state[0][1];

// for(int i = 0; i < 3; i++)

// {

// state[i][1] = state[i+1][1];

// }

// state[3][1] = p01;

//

// //row 2 2 times left;

// byte p02 = state[0][2];

// byte p12 = state[1][2];

// state[0][2] = state[2][2];

// state[1][2] = state[3][2];

// state[2][2] = p02;

// state[3][2] = p12;

//

// //row 3 3 times left

// byte p33 = state[3][3];

// for(int i = 3; i > 0 ; i--)

// {

// state[i][3] = state[i - 1][3];

// }

// state[0][3] = p33;

//

// return state;

// }

//

// //Big function 2.5

// private static byte[][] ishiftRows(byte[][] state)

// {

// //now we do right shifts based on inverse row indexes

// //shift row 1 3 times right

// byte row2 = state[3][1];

// for(int i = 3; i > 0; i--)

// {

// state[i][1] = state[i - 1][1];

// }

// state[0][1] = row2;

//

// //shift row 2 2 times right

// byte row3a = state[0][2];

// byte row3b = state[1][2];

// state[0][2] = state[2][2];

// state[1][2] = state[3][2];

// state[2][2] = row3a;

// state[3][2] = row3b;

//

// // shift row 3 1 time right

// byte row4 = state[0][3];

// for(int i = 0; i < 3; i++)

// {

// state[i][3] = state[i + 1][3];

// }

// state[3][3] = row4;

//

// return state;

// }

//

// //Big function 3

// private static byte[][] mixColumns(byte[][] state)

// {

// //we perform ffMultiply with each column in our 2d array. This is done with predefined hex values.

// //this is like doing a dot product with this column

// byte[] hexTable = { 0x00, 0x01, 0x02, 0x03 };

// for(int i = 0; i < 4; i++){

// byte[] col = state[i];

// byte[] resCol = new byte[4];

// resCol[0] = (byte) (ffMult(hexTable[2], col[0]) ^ ffMult(hexTable[3], col[1]) ^ ffMult(hexTable[1], col[2]) ^ ffMult(hexTable[1], col[3]));

// resCol[1] = (byte) (ffMult(hexTable[1], col[0]) ^ ffMult(hexTable[2], col[1]) ^ ffMult(hexTable[3], col[2]) ^ ffMult(hexTable[1], col[3]));

// resCol[2] = (byte) (ffMult(hexTable[1], col[0]) ^ ffMult(hexTable[1], col[1]) ^ ffMult(hexTable[2], col[2]) ^ ffMult(hexTable[3], col[3]));

// resCol[3] = (byte) (ffMult(hexTable[3], col[0]) ^ ffMult(hexTable[1], col[1]) ^ ffMult(hexTable[1], col[2]) ^ ffMult(hexTable[2], col[3]));

// state[i] = resCol;

// }

// return state;

// }

//

// //Big funciton 3.5

// private static byte[][] imixColumns(byte[][] state)

// {

// //we perform ffMultiply with each column in our 2d array. This is done with MORE predefined hex values.

// byte[] hexTable = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e };

// for(int i = 0; i < 4; i++)

// {

// byte[] col = state[i];

// byte[] newCol = new byte[4];

// newCol[0] = (byte) (ffMult(hexTable[0x0e], col[0]) ^ ffMult(hexTable[0x0b], col[1]) ^ ffMult(hexTable[0x0d], col[2]) ^ ffMult(hexTable[0x09], col[3]));

// newCol[1] = (byte) (ffMult(hexTable[0x09], col[0]) ^ ffMult(hexTable[0x0e], col[1]) ^ ffMult(hexTable[0x0b], col[2]) ^ ffMult(hexTable[0x0d], col[3]));

// newCol[2] = (byte) (ffMult(hexTable[0x0d], col[0]) ^ ffMult(hexTable[0x09], col[1]) ^ ffMult(hexTable[0x0e], col[2]) ^ ffMult(hexTable[0x0b], col[3]));

// newCol[3] = (byte) (ffMult(hexTable[0x0b], col[0]) ^ ffMult(hexTable[0x0d], col[1]) ^ ffMult(hexTable[0x09], col[2]) ^ ffMult(hexTable[0x0e], col[3]));

// state[i] = newCol;

// }

// return state;

// }

//

// //part of Big function 3 and 3.5. Does our XOR operation

// private static byte ffMult(byte val1, byte val2)

// {

// byte temp = val2;

// byte total = 0x00;

// //for each bit

// for(int i = 0; i < 8; i++)

// {

// //if you left shift and and it with 0x01 to get 0x01, your total should be XORed with temp for a new total;

// if(((val1 >> (i))& 0x01) == 0x01) total ^= temp;

// //if anding with 0x80 ==0x80, you XOR with 0X1B if not, you bit shift by 1.

// temp = (((temp & 0x80) == 0x80)? (byte) ((temp << 1) ^ 0x1B): (byte) (temp << 1));

// }

// return (byte) total;

// }

//

// //Big function 4

// private static byte[][] addRoundKey(byte[][] state, byte[][] key)

// {

// for(int i = 0; i < state.length; i++)

// {

// for(int j = 0; j < state[0].length; j++)

// {

// //Round Key XOR-d with the State for a new result of State

// state[i][j] ^= key[i][j];

// }

// }

// return state;

// }

//

// public static byte[] stringToBytes(String input)

// {

// String[] parts = input.split(" ");

// byte[] result = new byte[parts.length];

// for(int i = 0; i < parts.length; i++)

// result[i] = (byte)Integer.parseInt(parts[i], 16);

// return result;

// }

//

// public static String byteArrayToString(byte[] bytes)

// {

// String result = "";

// for(int i = 0; i < bytes.length; i++)

// {

// if(i != 0) result += " ";

// result += String.format("%02x", bytes[i]);

// }

// return result;

// }

//

// public static String printLine(String message, byte[][] bytes)

//{

// String result = message + " ";

// // String result = message + " \n";

// for(int i = 0; i < bytes.length; i++){

// for(int j = 0; j < bytes[0].length; j++)

// result += String.format("%02x", bytes[i][j]) + " ";

// }

// System.out.println(result);

// return result;

// }

//

// //public static String printBlock(byte[][] bytes){ return printBlock("", bytes);}

// public static String printBlock(String message, byte[][] bytes)

// {

// String result = message + " \n";

// for(int j = 0; j < bytes[0].length; j++)

// {

// for(int i = 0; i < bytes.length; i++)

// {

// result += String.format("%02x", bytes[i][j]) + " ";

// }

// result += "\n";

// }

// System.out.println(result);

// return result;

// }

//}

\*/