LAMPIRAN A

AES CRYPTOGRAPHER

BASE TRANSFORM

```
    class BaseTransform
    {
        #region Convert text to hex
        public static string FromTextToHex(string text)
        {
            StringBuilder hexstring = new
StringBuilder(text.Length * 2);

            foreach (char word in text)
            {
                hexstring.Append(String.Format("{0:X}",
Convert.ToInt32(word)));
            }

            return hexstring.ToString();
        }
        #endregion

        #region convert hex & binary to text
        public static string FromHexToText(string hexstring)
        {
            StringBuilder text = new
StringBuilder(hexstring.Length / 2);

            for (int i = 0; i < (hexstring.Length / 2); i++)
            {
                string word = hexstring.Substring(i * 2, 2);
                text.Append((char)Convert.ToInt32(word, 16));
            }

            return text.ToString();
        }

        public static string FromBinaryToText(string binarystring)
        {
            StringBuilder text = new
StringBuilder(binarystring.Length / 8);

            for (int i = 0; i < (binarystring.Length / 8); i++)
            {
                string word = binarystring.Substring(i * 8, 8);
                text.Append((char)Convert.ToInt32(word, 2));
            }

            return text.ToString();
        }
        #endregion
        #region Convert integer to binary
        public static string FromDeciamlToBinary(int binary)
        {
            if (binary < 0)
            {
                Console.WriteLine("It requires a integer greater
than 0.");
                return null;
            }

            string binarystring = "";
            int factor = 128;

            for (int i = 0; i < 8; i++)
            {
                if (binary >= factor)
                {
                    binary -= factor;
                    binarystring += "1";
                }
                else
                {
                    binarystring += "0";
                }
                factor /= 2;
            }

            return binarystring;
        }

        public static byte FromBinaryToByte(string binary)
        {
            byte value = 0;
            int factor = 128;

            for (int i = 0; i < 8; i++)
            {
                if (binary[i] == '1')
                {
                    value += (byte)factor;
                }

                factor /= 2;
            }

            return value;
        }
        #endregion

        #region convert hex to binary
        public static string FromHexToBinary(string hexstring)
        {
            StringBuilder binarystring = new
StringBuilder(hexstring.Length * 4);

            try
            {
                for (int i = 0; i < hexstring.Length; i++)
                {
                    int hex =
Convert.ToInt32(hexstring[i].ToString(), 16);

                    int factor = 8;

                    for (int j = 0; j < 4; j++)
                    {
                        if (hex >= factor)
                        {
                            hex -= factor;
                            binarystring.Append("1");
                        }
                        else
                        {
                            binarystring.Append("0");
                        }
                        factor /= 2;
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message + " - wrong hexa
integer format.");
            }

            return binarystring.ToString();
        }
        #endregion

        #region Convert Binary to hex
        public static string FromBinaryToHex(string binarystring)
        {
            StringBuilder hexstring = new
StringBuilder(binarystring.Length / 4);

            for (int i = 0; i < binarystring.Length / 4; i++)
            {
                int word =
Convert.ToInt32(binarystring.Substring(i * 4, 4), 2);

                hexstring.Append(String.Format("{0:X}", word));
            }

            return hexstring.ToString();
        }
        #endregion

        #region Convert Text to binary
        public static string FromTextToBinary(string text)
        {
            StringBuilder binarystring = new
StringBuilder(text.Length * 8);
```

```
        foreach (char word in text)
        {
            int binary = word;
            int factor = 128;

            for (int i = 0; i < 8; i++)
            {
                if (binary >= factor)
                {
                    binary -= factor;
                    binarystring.Append("1");
                }
                else
                {
                    binarystring.Append("0");
                }
                factor /= 2;
            }
        }

        return binarystring.ToString();
    }
    #endregion

    #region Set Text Multiple Of 64bit
    public static string setTextMutipleOf64Bits(string text)
    {
        int maxLength = 0;

        if ((text.Length % 64) != 0)
        {
            maxLength = ((text.Length / 64) + 1) * 64;
        }

        text = text.PadRight(maxLength, '0');

        return text;
    }
    #endregion

    #region Set text multiple of 128bit
    public static string setTextMutipleOf128Bits(string text)
    {
        if ((text.Length % 128) != 0)
        {
            int maxLength;
            maxLength = ((text.Length / 128) + 1) * 128;

            text = text.PadRight(maxLength, '0');
        }

        return text;
    }
    #endregion
}
```

TRANSFORM TABLES

```
class TransformTables
    {
        #region S-Box and Inverse S-Box
        public static readonly string [,] sbox = new string[,] {

{"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"}};
```

```
        public static readonly string[,] inverse_sbox = new
string[,] {

{"52","09","6a","d5","30","36","a5","38","bf","40","a3","9e","81",
"f3","d7","fb"},

{"7c","e3","39","82","9b","2f","ff","87","34","8e","43","44","c4",
"de","e9","cb"},

{"54","7b","94","32","a6","c2","23","3d","ee","4c","95","0b","42",
"fa","c3","4e"},

{"08","2e","a1","66","28","d9","24","b2","76","5b","a2","49","6d",
"8b","d1","25"},

{"72","f8","f6","64","86","68","98","16","d4","a4","5c","cc","5d",
"65","b6","92"},

{"6c","70","48","50","fd","ed","b9","da","5e","15","46","57","a7",
"8d","9d","84"},

{"90","d8","ab","00","8c","bc","d3","0a","f7","e4","58","05","b8",
"b3","45","06"},

{"d0","2c","1e","8f","ca","3f","0f","02","c1","af","bd","03","01",
"13","8a","6b"},

{"3a","91","11","41","4f","67","dc","ea","97","f2","cf","ce","f0",
"b4","e6","73"},

{"96","ac","74","22","e7","ad","35","85","e2","f9","37","e8","1c",
"75","df","6e"},

{"47","f1","1a","71","1d","29","c5","89","6f","b7","62","0e","aa",
"18","be","1b"},

{"fc","56","3e","4b","c6","d2","79","20","9a","db","c0","fe","78",
"cd","5a","f4"},

{"1f","dd","a8","33","88","07","c7","31","b1","12","10","59","27",
"80","ec","5f"},

{"60","51","7f","a9","19","b5","4a","0d","2d","e5","7a","9f","93",
"c9","9c","ef"},

{"a0","e0","3b","4d","ae","2a","f5","b0","c8","eb","bb","3c","83",
"53","99","61"},

{"17","2b","04","7e","ba","77","d6","26","e1","69","14","63","55",
"21","0c","7d"}};
        #endregion

        #region MixColumns Transform Matrix
        public static readonly Matrix MixColumnFactor = new
Matrix(BaseTransform.FromHexToBinary("0201010303020101010302010101
0302"));
```

```
        public static readonly Matrix Inverse_MixColumnFactor =
new
Matrix(BaseTransform.FromHexToBinary("0e090d0b0b0e090d0d0b0e09090d
0b0e"));
        #endregion

        #region Rcon
        public static List<Matrix> Rcon = new List<Matrix>(10);
        #endregion

        #region Transform Tables
        static TransformTables()
        {
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("01000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("02000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("04000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("08000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("10000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("20000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("40000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("80000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("1b000000"), 4, 1));
            Rcon.Add(new
Matrix(BaseTransform.FromHexToBinary("36000000"), 4, 1));
        }
        #endregion
```

KEYS

```
class Keys
    {
        public List<Matrix> RoundKeys = new List<Matrix>(11);


        public void setCipherKey(Matrix CipherKey)
        {
            if (RoundKeys.Count == 0)
            {
                RoundKeys.Add(CipherKey);
            }
            else
            {
                RoundKeys.Clear();
                RoundKeys.Add(CipherKey);
            }
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
            RoundKeys.Add(new Matrix(4, 4));
        }
    }
```

MATRIX

```
    class Matrix
    {
        #region Private Fields
        private string[,] matrix;
        private int rows = 0;
        private int columns = 0;
        #endregion

        #region Constructor
        public Matrix(int rows, int columns)
            : this("", rows, columns)
        {
        }

        public Matrix(string text)
            : this(text, 4, 4)
        {
        }

        public Matrix(string text, int rows, int columns)
        {
            if (text.Length != columns * rows * 8)
            {
                text = text.PadRight(columns * rows * 8 -
text.Length, '0');
            }

            matrix = new string[rows, columns];
            int count = 0;
            this.rows = rows;
            this.columns = columns;

            for (int i = 0; i < columns; i++)
            {
                for (int j = 0; j < rows; j++)
                {
                    matrix[j, i] = text.Substring(count * 8, 8);
                    count++;
                }
            }
        }
        #endregion

        public void setState(string text)
        {
            if (text.Length != columns * rows * 8)
            {
                throw new Exception("It's not equal size to
state");
            }

            int count = 0;

            for (int i = 0; i < columns; i++)
            {
                for (int j = 0; j < rows; j++)
                {
                    matrix[j, i] = text.Substring(count * 8, 8);
                    count++;
                }
            }
        }

        #region Properties, Indexer
        public int Rows
        {
            get
            {
                return rows;
            }
        }

        public int Columns
        {
            get
            {
                return columns;
            }
        }

        public string this[int i, int j]
        {
            get
            {
                return matrix[i, j];
            }
            set
            {
                if (value.Length == 2)
                {
                    matrix[i, j] =
BaseTransform.FromHexToBinary(value);
                }
                else if (value.Length == 8)
                {
                    matrix[i, j] = value;
                }
            }
        }
        #endregion

        #region Overrided ToString method
        public override string ToString()
        {
            StringBuilder text = new StringBuilder(128);
            if (matrix != null)
            {
                for (int j = 0; j < columns; j++)
                {
                    for (int i = 0; i < rows; i++)
                    {
                        text.Append(matrix[i, j]);
                    }
                }
            }

            return text.ToString();
        }
        #endregion

        #region one row operation
        public string[] getRow(int rowindex)
        {
            string[] row = new string[columns];

            if (rowindex > rows)
            {
                throw new IndexOutOfRangeException("out of row
index error.");
            }

            for (int i = 0; i < columns; i++)
            {
                row[i] = matrix[rowindex, i];
            }

            return row;
        }

        public void setRow(string[] row, int rowindex)
        {
            if (rowindex > rows)
            {
                throw new IndexOutOfRangeException("out of row
index error.");
            }

            for (int i = 0; i < columns; i++)
            {
                matrix[rowindex, i] = row[i];
            }
        }
        #endregion

        #region one column operation
        public string[] getWord(int wordindex)
        {
            string[] word = new string[rows];

            if (wordindex > rows)
            {
                throw new IndexOutOfRangeException("out of column
index error.");
            }

            for (int i = 0; i < rows; i++)
            {
                word[i] = matrix[i, wordindex];
            }

            return word;
        }

        public void setWord(string[] word, int wordindex)
        {
            if (wordindex > rows)
            {
                throw new IndexOutOfRangeException("out of column
index error.");
            }

            for (int i = 0; i < rows; i++)
            {
                matrix[i, wordindex] = word[i];
            }
        }
        #endregion

    }
    #endregion

    #region Matrix Operations
    class MatrixMultiplication
    {
        public static Matrix Multiply(Matrix a, Matrix b, bool
IsMixColumns)
        {
            if (IsMixColumns)
            {
                return MixColumnsMultiply(a, b);
            }
            else
            {
                return null;
            }
        }
        public static Matrix XOR(Matrix a, Matrix b)
        {
            Matrix c = new Matrix(a.Rows, a.Columns);

            for (int i = 0; i < c.Rows; i++)
            {
                for (int j = 0; j < c.Columns; j++)
                {
                    c[i, j] = MultiplicativeInverse.XOR(a[i, j], 
b[i, j]);
                }
            }

            return c;
        }

        #region Matrix multiplication by MixColumns step
        private static Matrix MixColumnsMultiply(Matrix a, Matrix 
b)
        {
            /* If A is an m-by-n matrix and B is an n-by-p matrix, 
then their matrix product AB is the m-by-p matrix (m rows, p 
columns) */
            //A - m rows, n columns
            //B - n rows, p columns
            //AB - m rows, p columns
            Matrix c = new Matrix(a.Rows, b.Columns);
            //string temp2 = "";


            for (int j = 0; j < c.Columns; j++)
            {
                //temp.Remove(0, temp.Length);

                for (int i = 0; i < c.Rows; i++)
                {
                    StringBuilder temp = new StringBuilder(32);


temp.Append(MultiplicativeInverse.GetInverse(a[i, 0], b[0, j], 
"00011011", 8));

temp.Append(MultiplicativeInverse.GetInverse(a[i, 1], b[1, j], 
"00011011", 8));

temp.Append(MultiplicativeInverse.GetInverse(a[i, 2], b[2, j], 
"00011011", 8));

temp.Append(MultiplicativeInverse.GetInverse(a[i, 3], b[3, j], 
"00011011", 8));

                    string temp2;
                    temp2 = 
MultiplicativeInverse.XOR(temp.ToString().Substring(0, 8), 
temp.ToString().Substring(8, 8));

                    temp2 = MultiplicativeInverse.XOR(temp2, 
temp.ToString().Substring(16, 8));
                    temp2 = MultiplicativeInverse.XOR(temp2, 
temp.ToString().Substring(24, 8));

                    c[i, j] = temp2;
                }
            }

            return c;
        }
        #endregion
    }
        #endregion
```

PROCESS AES

```
    class Matrix
    {
        #region Private Fields
        private string[,] matrix;
        private int rows = 0;
        private int columns = 0;
        #endregion

        #region Constructor
        public Matrix(int rows, int columns)
            : this("", rows, columns)
        {
        }

        public Matrix(string text)
            : this(text, 4, 4)
        {
        }

        public Matrix(string text, int rows, int columns)
        {
            if (text.Length != columns * rows * 8)
            {
                text = text.PadRight(columns * rows * 8 - 
text.Length, '0');
            }

            matrix = new string[rows, columns];
            int count = 0;
            this.rows = rows;
            this.columns = columns;

            for (int i = 0; i < columns; i++)
            {
                for (int j = 0; j < rows; j++)
                {
                    matrix[j, i] = text.Substring(count * 8, 8);
                    count++;
                }
            }
        }
        #endregion

        public void setState(string text)
        {
            if (text.Length != columns * rows * 8)
            {
                throw new Exception("It's not equal size to
state");
            }

            int count = 0;

            for (int i = 0; i < columns; i++)
            {
                for (int j = 0; j < rows; j++)
                {
                    matrix[j, i] = text.Substring(count * 8, 8);
                    count++;
                }
            }
        }

        #region Properties, Indexer
        public int Rows
        {
            get
            {
                return rows;
            }
        }

        public int Columns
        {
            get
            {
                return columns;
            }
        }

        public string this[int i, int j]
        {
            get
            {
                return matrix[i, j];
            }
            set
            {
                if (value.Length == 2)
                {
                    matrix[i, j] =
BaseTransform.FromHexToBinary(value);
                }
                else if (value.Length == 8)
                {
                    matrix[i, j] = value;
                }
            }
        }
        #endregion

        #region Overrided ToString method
        public override string ToString()
        {
            StringBuilder text = new StringBuilder(128);
            if (matrix != null)
            {
                for (int j = 0; j < columns; j++)
                {
                    for (int i = 0; i < rows; i++)
                    {
                        text.Append(matrix[i, j]);
                    }
                }
            }

            return text.ToString();
        }
        #endregion

        #region one row operation
        public string[] getRow(int rowindex)
        {
            string[] row = new string[columns];

            if (rowindex > rows)
            {
                throw new IndexOutOfRangeException("out of row
index error.");
            }

            for (int i = 0; i < columns; i++)
            {
                row[i] = matrix[rowindex, i];
            }

            return row;
        }

        public void setRow(string[] row, int rowindex)
        {
            if (rowindex > rows)
            {
                throw new IndexOutOfRangeException("out of row
index error.");
```

```
            }

            for (int i = 0; i < columns; i++)
            {
                matrix[rowindex, i] = row[i];
            }
        }
        #endregion

        #region one column operation
        public string[] getWord(int wordindex)
        {
            string[] word = new string[rows];

            if (wordindex > rows)
            {
                throw new IndexOutOfRangeException("out of column
index error.");
            }

            for (int i = 0; i < rows; i++)
            {
                word[i] = matrix[i, wordindex];
            }

            return word;
        }

        public void setWord(string[] word, int wordindex)
        {
            if (wordindex > rows)
            {
                throw new IndexOutOfRangeException("out of column
index error.");
            }

            for (int i = 0; i < rows; i++)
            {
                matrix[i, wordindex] = word[i];
            }
        }
        #endregion

    }
    #endregion

    #region Matrix Operations
    class MatrixMultiplication
    {
        public static Matrix Multiply(Matrix a, Matrix b, bool
IsMixColumns)
        {
            if (IsMixColumns)
            {
                return MixColumnsMultiply(a, b);
            }
            else
            {
                return null;
            }
        }

        public static Matrix XOR(Matrix a, Matrix b)
        {
            Matrix c = new Matrix(a.Rows, a.Columns);

            for (int i = 0; i < c.Rows; i++)
            {
                for (int j = 0; j < c.Columns; j++)
                {
                    c[i, j] = MultiplicativeInverse.XOR(a[i, j],
b[i, j]);
                }
            }

            return c;
        }

        #region Matrix multiplication by MixColumns step
        private static Matrix MixColumnsMultiply(Matrix a, Matrix
b)
        {
            /* If A is an m-by-n matrix and B is an n-by-p matrix,
then their matrix product AB is the m-by-p matrix (m rows, p
columns) */
            //A - m rows, n columns
            //B - n rows, p columns
            //AB - m rows, p columns
            Matrix c = new Matrix(a.Rows, b.Columns);
            //string temp2 = "";


            for (int j = 0; j < c.Columns; j++)
            {
                //temp.Remove(0, temp.Length);

                for (int i = 0; i < c.Rows; i++)
                {
                    StringBuilder temp = new StringBuilder(32);


temp.Append(MultiplicativeInverse.GetInverse(a[i, 0], b[0, j],
"00011011", 8));

temp.Append(MultiplicativeInverse.GetInverse(a[i, 1], b[1, j],
"00011011", 8));

temp.Append(MultiplicativeInverse.GetInverse(a[i, 2], b[2, j],
"00011011", 8));

temp.Append(MultiplicativeInverse.GetInverse(a[i, 3], b[3, j],
"00011011", 8));
```

```
                    string temp2;

                    temp2 =
MultiplicativeInverse.XOR(temp.ToString().Substring(0, 8),
temp.ToString().Substring(8, 8));

                    temp2 = MultiplicativeInverse.XOR(temp2,
temp.ToString().Substring(16, 8));
                    temp2 = MultiplicativeInverse.XOR(temp2,
temp.ToString().Substring(24, 8));

                    c[i, j] = temp2;
                }
            }

            return c;
        }
        #endregion
    }
    #endregion
```

MULTIPLICATIVE INVERSE

```
class MultiplicativeInverse
    {
        #region For Multiplication of GP(2n)

        #region Non-circular left shift
        public static string LeftShift2(string text, int level)
        {
            StringBuilder shifted = new
StringBuilder(text.Length);
            shifted.Append(text.Substring(1) + "0");

            if (!level.Equals(8))
            {
                for (int i = 0; i <= text.Length - (1 + level);
i++)
                {
                    shifted[i] = '0';
                }
            }

            return shifted.ToString();
        }
        #endregion

        #region Calculate Multiplicative Inverse
        public static string GetInverse(string text1, string
text2, string mx, int n)
        {
            string[] multiplyTable = new string[n];
            if (text1.IndexOf('1') > text2.IndexOf('1'))
            {
                string temp = text2;
                text2 = text1;
                text1 = temp;
            }

            multiplyTable[0] = text1;

            for (int i = 1; i < n; i++)
            {
                multiplyTable[i] = LeftShift2(multiplyTable[i -
1], n);

                if (multiplyTable[i - 1][text1.Length -
n].Equals('1'))
                {
                    multiplyTable[i] = XOR(multiplyTable[i], mx);
                }
            }

            string Mul_Inverse = "";

            for (int i = 0; i < text2.Length; i++)
            {
                if (text2[i].Equals('1'))
                {
                    if (Mul_Inverse.Equals(""))
                    {
                        Mul_Inverse = multiplyTable[(text2.Length
- 1/*2*/) - i];
                    }
                    else
                    {
                        Mul_Inverse = XOR(Mul_Inverse,
multiplyTable[(text2.Length - 1) - i]);
                    }
                }
            }

            if (Mul_Inverse.Equals(""))
            {
                Mul_Inverse = "00000000";
            }

            return Mul_Inverse;
        }
        #endregion

        #region XOR Operation
        public static string XOR(string text1, string text2)
        {
            if (text1.Length != text2.Length)
            {
                int count = Math.Abs(text1.Length - text2.Length);
                string temp = "";
                for (int i = 0; i < count; i++)
                {
                    temp += "0";
                }

                if (text1.Length > text2.Length)
                {
                    text2 = temp + text2;
                }
                else
                {
                    text1 = temp + text1;
                }
            }

            StringBuilder XORed_Text = new
StringBuilder(text1.Length);

            for (int i = 0; i < text1.Length; i++)
            {
                if (text1[i] != text2[i])
                {
                    XORed_Text.Append("1");
                }
                else
                {
                    XORed_Text.Append("0");
                }
            }

            return XORed_Text.ToString();
        }
        #endregion

        #endregion
    }
}
```