Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра прикладной математики и компьютерной безопасности

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Современные симметричные шифры. AES-128/AES-256

Преподаватель \_\_\_\_\_\_\_\_\_\_ А. А. Сидарас

подпись, дата

Студент КИ19-09Б, 031940306 \_\_\_\_\_\_\_\_\_\_ С.А. Ману

подпись, дата

Красноярск 2021

**1 Задание к лабораторной работе**

Реализовать алгоритм шифрования/расшифровывания AES-256.

**2 Описание алгоритма шифрования**

Описание алгоритма шифрования/расшифровывания можно увидеть на сайте[1].

**3 Листинг программы**

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using [System.Windows](https://vk.com/away.php?utf=1&to=http%3A%2F%2FSystem.Windows" \t "_blank);  
  
namespace aes128  
{  
public partial class MainWindow : Window  
{  
private const int nb = 4;  
private const int nr = 10;  
private const int nk = 4;  
  
private static readonly byte[] sbox =  
{  
//0 1 2 3 4 5 6 7 8 9 A B C D E F  
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0  
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1  
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2  
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3  
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4  
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5  
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6  
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7  
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8  
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9  
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A  
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B  
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C  
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D  
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E  
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16  
};  
  
private static readonly byte[] InvSbox =  
{  
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,  
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,  
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,  
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,  
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,  
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,  
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,  
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,  
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,  
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,  
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,  
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,  
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,  
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,  
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,  
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d  
};  
  
private static readonly List<List<byte» rcon = new()  
{  
new() {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36},  
new() {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},  
new() {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},  
new() {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}  
};  
  
  
public MainWindow()  
{  
InitializeComponent();  
}  
  
private void EcnryptButton\_Click(object sender, RoutedEventArgs e)  
{  
var input =https://www.microsoft.com/en-us/windows

Encrypt\_TextBox.Text;  
var byteInput = Encoding.ASCII.GetBytes(input).ToList();  
  
var crypted\_data = new List<byte>();  
var crypted\_part = new List<byte>();  
  
var temp = new List<byte>();  
foreach (var b in byteInput)  
{  
temp.Add(b);  
if (temp.Count == 16)  
{  
crypted\_part = encrypt(temp, Cypher\_Key.Text);  
crypted\_data.AddRange(crypted\_part);  
temp.Clear();  
}  
}  
  
var count = temp.Count();  
if (count > 0 && count < 16)  
{  
var empty\_spaces = 16 - count;  
  
for (var i = 0; i < empty\_spaces - 1; i++) temp.Add(0x00);  
temp.Add(0x03);  
  
crypted\_part = encrypt(temp, Cypher\_Key.Text);  
crypted\_data.AddRange(crypted\_part);  
}  
  
Decrypt\_TextBox.Text = BitConverter.ToString(crypted\_data.ToArray())  
.Replace("-", " ");  
}  
  
  
private void DecryptButton\_Click(object sender, RoutedEventArgs e)  
{  
var input = Decrypt\_TextBox.Text;  
var strings = input.Split(' ');  
var cipher = input.Split(' ')  
.Select(num => Convert.ToByte(num, 16))  
.ToList();  
var decrypted = decrypt(cipher, Cypher\_Key.Text);  
Decrypt\_TextBox.Text = Encoding.ASCII.GetString(decrypted.ToArray());  
}  
  
public static List<byte> encrypt(List<byte> input\_bytes, string key)  
{  
var state = new List<List<byte»();  
  
for (var i = 0; i < 4; i++) state.Add(new List<byte>());  
  
for (var r = 0; r < 4; r++)  
for (var c = 0; c < nb; c++)  
state[r].Add(input\_bytes[r + 4 \* c]);  
  
var key\_schedule = key\_expansion(key);  
  
state = add\_round\_key(state, key\_schedule);  
  
for (var rnd = 1; rnd < nr; rnd++)  
{  
state = sub\_bytes(state);  
state = shift\_rows(state);  
state = mix\_columns(state);  
state = add\_round\_key(state, key\_schedule, rnd);  
}  
  
state = sub\_bytes(state);  
state = shift\_rows(state);  
state = add\_round\_key(state, key\_schedule, nr);  
  
  
var output = Enumerable.Repeat<byte>(0x00, 4 \* nb).ToList();  
  
for (var r = 0; r < 4; r++)  
for (var c = 0; c < nb; c++)  
output[r + 4 \* c] = state[r][c];  
  
return output;  
}  
  
public static List<byte> decrypt(List<byte> cipher, string key)  
{  
var state = new List<List<byte»();  
  
for (var i = 0; i < 4; i++) state.Add(new List<byte>());  
  
for (var r = 0; r < 4; r++)  
for (var c = 0; c < nb; c++)  
state[r].Add(cipher[r + 4 \* c]);  
  
var key\_schedule = key\_expansion(key);  
  
state = add\_round\_key(state, key\_schedule, nr);  
  
var rnd = nr - 1;  
while (rnd >= 1)  
{  
state = shift\_rows(state, true);  
state = sub\_bytes(state, true);  
state = add\_round\_key(state, key\_schedule, rnd);  
state = mix\_columns(state, true);  
  
rnd -= 1;  
}  
  
state = shift\_rows(state, true);  
state = sub\_bytes(state, true);  
state = add\_round\_key(state, key\_schedule, rnd);  
  
var output = Enumerable.Repeat<byte>(0x00, 4 \* nb).ToList();  
  
for (var r = 0; r < 4; r++)  
for (var c = 0; c < nb; c++)  
output[r + 4 \* c] = state[r][c];  
  
return output;  
}  
  
  
private static List<List<byte» key\_expansion(string key)  
{  
var key\_symbols = Encoding.ASCII.GetBytes(key).ToList();  
  
var key\_length = key\_symbols.Count();  
if (key\_length < 4 \* nk)  
for (var i = 0; i < 4 \* nk - key\_length; i++)  
key\_symbols.Add(0x01);  
  
var key\_schedule = new List<List<byte»  
{  
new(),  
new(),  
new(),  
new()  
};  
  
for (var r = 0; r < 4; r++)  
for (var c = 0; c < nk; c++)  
key\_schedule[r].Add(key\_symbols[r + 4 \* c]);  
  
  
for (var col = nk; col < nb \* (nr + 1); col++)  
{  
var tmp = new List<byte>();  
if (col % nk == 0)  
{  
for (var row = 1; row < 4; row++) tmp.Add(key\_schedule[row][col - 1]);  
tmp.Add(key\_schedule[0][col - 1]);  
  
for (var j = 0; j < tmp.Count(); j++)  
{  
var sbox\_row = tmp[j] / 0x10;  
var sbox\_col = tmp[j] % 0x10;  
var sbox\_elem = sbox[16 \* sbox\_row + sbox\_col];  
tmp[j] = sbox\_elem;  
}  
  
for (var row = 0; row < 4; row++)  
{  
var s = (byte) (key\_schedule[row][col - 4] ^ tmp[row] ^ rcon[row][col / nk - 1]);  
key\_schedule[row].Add(s);  
}  
}  
else  
{  
for (var row = 0; row < 4; row++)  
{  
var s = (byte) (key\_schedule[row][col - 4] ^ key\_schedule[row][col - 1]);  
key\_schedule[row].Add(s);  
}  
}  
}  
  
return key\_schedule;  
}

private static List<List<byte» add\_round\_key(List<List<byte» state, List<List<byte» key\_schedule,  
int round = 0)  
{  
for (var col = 0; col < nb; col++)  
{  
var s0 = (byte) (state[0][col] ^ key\_schedule[0][nb \* round + col]);  
var s1 = (byte) (state[1][col] ^ key\_schedule[1][nb \* round + col]);  
var s2 = (byte) (state[2][col] ^ key\_schedule[2][nb \* round + col]);  
var s3 = (byte) (state[3][col] ^ key\_schedule[3][nb \* round + col]);  
  
state[0][col] = s0;  
state[1][col] = s1;  
state[2][col] = s2;  
state[3][col] = s3;  
}  
  
return state;  
}  
  
private static List<List<byte» sub\_bytes(List<List<byte» state, bool inv = false)  
{  
List<byte> box;  
  
if (inv == false) // encrypt else decrypt  
box = sbox.ToList();  
else  
box = InvSbox.ToList();  
  
for (var i = 0; i < state.Count(); i++)  
for (var j = 0; j < state[i].Count(); j++)  
{  
var row = state[i][j] / 0x10;  
var col = state[i][j] % 0x10;  
  
var box\_elem = box[16 \* row + col];  
state[i][j] = box\_elem;  
}  
  
return state;  
}  
  
private static List<List<byte» shift\_rows(List<List<byte» state, bool inv = false)  
{  
var count = 1;  
  
if (inv == false) // encrypting else decrypting  
for (var i = 1; i < nb; i++)  
{  
state[i] = left\_shift(state[i], count);  
count++;  
}  
else  
for (var i = 1; i < nb; i++)  
{  
state[i] = right\_shift(state[i], count);  
count++;  
}  
  
return state;  
}  
  
private static List<List<byte» mix\_columns(List<List<byte» state, bool inv = false)  
{  
byte s0, s1, s2, s3;  
  
for (var i = 0; i < nb; i++)  
{  
if (inv == false) // encryption else decryption  
{  
s0 = (byte) (mul\_by\_02(state[0][i]) ^ mul\_by\_03(state[1][i]) ^ state[2][i] ^ state[3][i]);  
s1 = (byte) (state[0][i] ^ mul\_by\_02(state[1][i]) ^ mul\_by\_03(state[2][i]) ^ state[3][i]);  
s2 = (byte) (state[0][i] ^ state[1][i] ^ mul\_by\_02(state[2][i]) ^ mul\_by\_03(state[3][i]));  
s3 = (byte) (mul\_by\_03(state[0][i]) ^ state[1][i] ^ state[2][i] ^ mul\_by\_02(state[3][i]));  
}  
else  
{  
s0 = (byte) (mul\_by\_0e(state[0][i]) ^ mul\_by\_0b(state[1][i]) ^ mul\_by\_0d(state[2][i]) ^  
mul\_by\_09(state[3][i]));  
s1 = (byte) (mul\_by\_09(state[0][i]) ^ mul\_by\_0e(state[1][i]) ^ mul\_by\_0b(state[2][i]) ^  
mul\_by\_0d(state[3][i]));  
s2 = (byte) (mul\_by\_0d(state[0][i]) ^ mul\_by\_09(state[1][i]) ^ mul\_by\_0e(state[2][i]) ^  
mul\_by\_0b(state[3][i]));  
s3 = (byte) (mul\_by\_0b(state[0][i]) ^ mul\_by\_0d(state[1][i]) ^ mul\_by\_09(state[2][i]) ^  
mul\_by\_0e(state[3][i]));  
}  
  
state[0][i] = s0;  
state[1][i] = s1;  
state[2][i] = s2;  
state[3][i] = s3;  
}  
  
return state;  
}  
  
private static List<T> left\_shift<T>(List<T> list, int count)  
{  
var res = list;  
  
for (var i = 0; i < count; i++)  
{  
List<T> tmp;  
tmp = res.GetRange(1, res.Count() - 1);  
tmp.Add(res[0]);  
res = tmp;  
}  
  
return res;  
}  
  
private static List<T> right\_shift<T>(List<T> list, int count)  
{  
var res = list;  
  
for (var i = 0; i < count; i++)  
{  
var tmp = new List<T>();  
tmp.Add(res.Last());  
tmp.AddRange(res.GetRange(0, res.Count() - 1));  
res = tmp;  
}  
  
return res;  
}  
  
private static byte mul\_by\_02(byte num)  
{  
byte res;  
  
if (num < 0x80)  
res = (byte) (num « 1);  
else  
res = (byte) ((num « 1) ^ 0x1b);  
  
return (byte) (res % 0x100);  
}  
  
private static byte mul\_by\_03(byte num)  
{  
return (byte) (mul\_by\_02(num) ^ num);  
}  
  
private static byte mul\_by\_09(byte num)  
{  
return (byte) (mul\_by\_02(mul\_by\_02(mul\_by\_02(num))) ^ num);  
}  
  
private static byte mul\_by\_0b(byte num)  
{  
return (byte) (mul\_by\_02(mul\_by\_02(mul\_by\_02(num))) ^ mul\_by\_02(num) ^ num);  
}  
  
private static byte mul\_by\_0d(byte num)  
{  
return (byte) (mul\_by\_02(mul\_by\_02(mul\_by\_02(num))) ^ mul\_by\_02(mul\_by\_02(num)) ^ num);  
}  
  
private static byte mul\_by\_0e(byte num)  
{  
return (byte) (mul\_by\_02(mul\_by\_02(mul\_by\_02(num))) ^ mul\_by\_02(mul\_by\_02(num)) ^ mul\_by\_02(num));  
}  
}  
}

**4 Контрольные примеры работы программы**

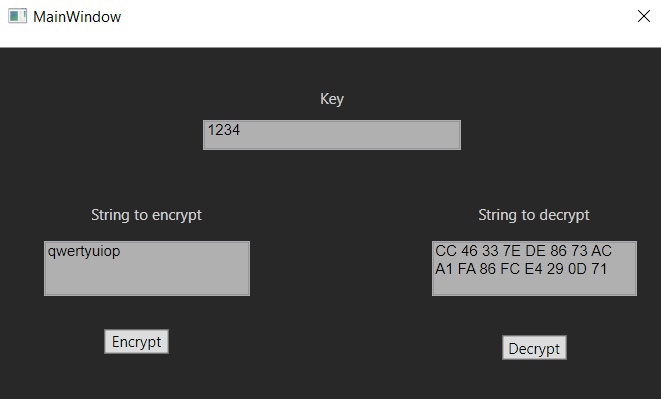


Рисунок 1 – Пример работы программы

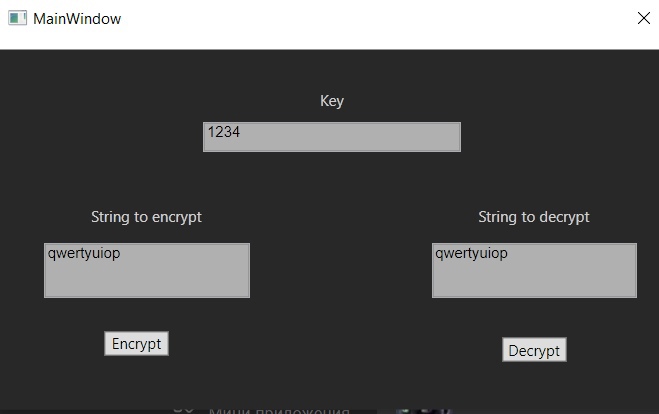


Рисунок 2 – Пример работы программы

**ЗАКЛЮЧЕНИЕ**

В результате выполнения данной лабораторной работы мне удалось изучить алгоритм шифрования/расшифровывания AES-256, реализовать его, а также проверить его работу на практике.

**ПРИЛОЖЕНИЕ**

Электронные ресурсы

1. AES-128. Детали и реализация на python [Электронный ресурс] / Кирилл Костюхин // Хабр – 2014. - Режим доступа: https://habr.com/ru/post/212235.