Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Защита информации»

Профиль: «Разработка программно-информационных систем»

Семестр 5

ОТЧЕТ

по лабораторной работе №5

Тема: «Шифры обнаружения и коррекции ошибок»

Выполнил: студент группы АСУ-19-1б

Шеретов М.А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2022

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по применению шифров обнаружения и коррекции ошибок.

**ЗАДАНИЕ**

Реализовать проверку правильности передачи двоичной кодовой последовательности, используя метод CRC16 - метод вычисления циклических сумм.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Циклический избыточный код (англ. Cyclic redundancy check)— алгоритм нахождения контрольной суммы, предназначенный для проверки целостности данных. CRC является практическим приложением помехоустойчивого кодирования, основанным на определённых математических свойствах циклического кода.

В общем виде контрольная сумма представляет собой некоторое значение, вычисленное по определённой схеме на основе кодируемого сообщения. Проверочная информация при систематическом кодировании приписывается к передаваемым данным. На принимающей стороне абонент знает алгоритм вычисления контрольной суммы: соответственно, программа имеет возможность проверить корректность принятых данных.

При передаче пакетов по сетевому каналу могут возникнуть искажения исходной информации вследствие разных внешних воздействий: электрических наводок, плохих погодных условий и многих других. Сущность методики в том, что при хороших характеристиках контрольной суммы в подавляющем числе случаев ошибка в сообщении приведёт к изменению его контрольной суммы. Если исходная и вычисленная суммы не равны между собой, принимается решение о недостоверности принятых данных, и можно запросить повторную передачу пакета.

Алгоритм CRC базируется на свойствах деления с остатком двоичных многочленов, то есть многочленов над конечным полем GF(2). Значение CRC является по сути остатком от деления многочлена, соответствующего входным данным, на некий фиксированный порождающий многочлен.

Значение контрольной суммы в алгоритме с порождающим многочленом G(x) степени N определяется как битовая последовательность длины N, представляющая многочлен R(x), получившийся в остатке при делении многочлена P(x), представляющего входной поток бит, на многочлен G(x):

R(x)=P(x) \* x^N mod G(x), где

R(x) — многочлен, представляющий значение CRC;

P(x) — многочлен, коэффициенты которого представляют входные данные;

G(x) — порождающий многочлен;

N — степень порождающего многочлена.

Умножение x^N осуществляется приписыванием N нулевых битов к входной последовательности, что улучшает качество хеширования для коротких входных последовательностей.

Тем не менее, существует множество стандартизированных образующих многочленов, обладающих хорошими математическими и корреляционными свойствами (минимальное число коллизий, простота вычисления), некоторые из которых перечислены ниже.

Основная идея вычисления CRC заключается в следующем. Исходная последовательность байтов представляется единой последовательностью битов. Эта последовательность делится на некоторое фиксированное двоичное число. Интерес представляется остаток от деления, который и является значение CRC.

**Алгоритм вычисления контрольной суммы CRC:**

Создаётся массив (регистр), заполненный нулями, равный по длине разрядности (степени) полинома.

Исходное сообщение дополняется нулями в младших разрядах, в количестве, равном числу разрядов полинома.

В младший разряд регистра заносится один старший бит сообщения, а из старшего разряда регистра выдвигается один бит.

Если выдвинутый бит равен "1", то производится инверсия битов (операция XOR, исключающее ИЛИ) в тех разрядах регистра, которые соответствуют единицам в полиноме.

Если в сообщении ещё есть биты, переходим к шагу 3).

Когда все биты сообщения поступили в регистр и были обработаны этим алгоритмом, в регистре остаётся остаток от деления, который и является контрольной суммой CRC.

Назовём этот метод расчёта CRC метод побитового сдвига или простой метод.

**Расчёт контрольной суммы CRC табличным методом**

Для сокращения числа вычислений из предыдущего метода – метода побитового сдвига – придуманы некоторые оптимизации.

В частности, сдвигают не по одному биту за раз, а сразу по несколько. Наибольшую популярность снискали варианты, в которых сообщение сдвигается на число битов, кратное числу битов в байте: 8, 16 или 32, т.к. с байтами легче работать (не нужны дополнительные преобразования). При этом идея алгоритма осталась та же: сдвиг и исключающее ИЛИ с содержимым регистра.

Кроме того, оказывается, что часть расчётов можно провести заранее и записать в массив – таблицу, из которой по мере необходимости будет браться нужное число. Такой метод расчёта назвали табличный метод расчёта CRC.

**ХОД РАБОТЫ**

На рисунке 1 представлено окно программы вычисления CRC16-CCITT.

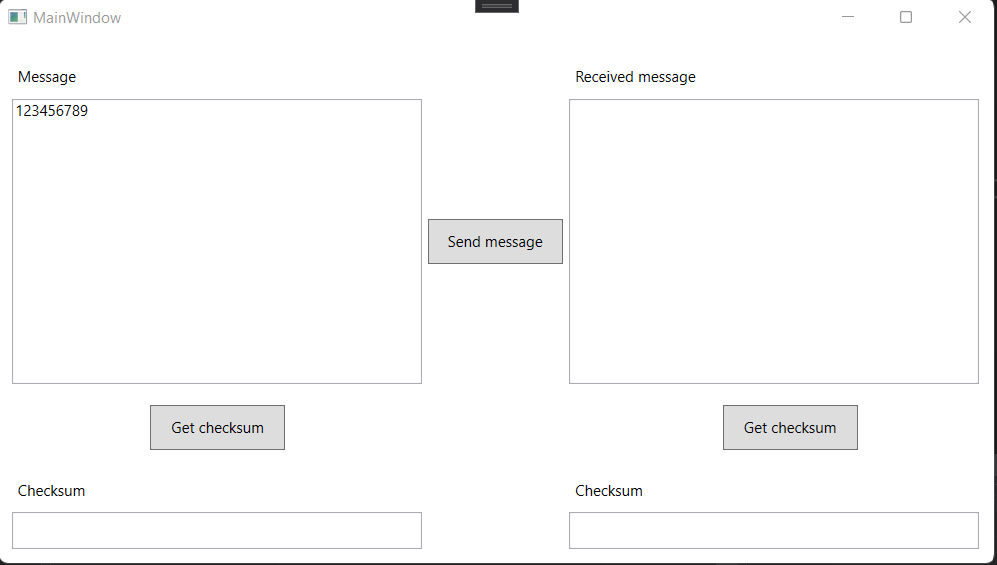


Рисунок 1. Окно программы

В первую форму вводится сообщение. Затем, при нажатии на кнопку будет вычислена контрольная сумма сообщения.

При нажатии на кнопку Send message, сообщение будет отправлено по вымышленному каналу связи в виде последовательности бит. При нажатии на кнопку под второй формой будет вычислена контрольная сумма полученного сообщения. Если обе КС совпадают, то сообщение отправлено без помех. Пример на рисунке 2.

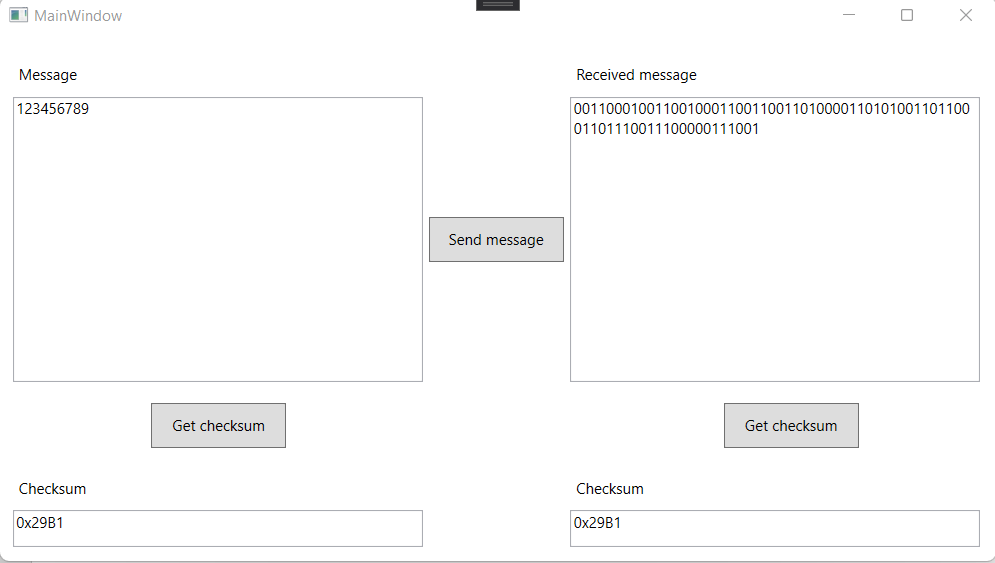


Рисунок 2. Пример работы программы

Если попробовать изменить любой бит с полученном сообщении и заново рассчитать контрольную сумму, то можно наблюдать несовпадение, что говорит о искажениях. Например, изменим последний бит (рисунок 3).

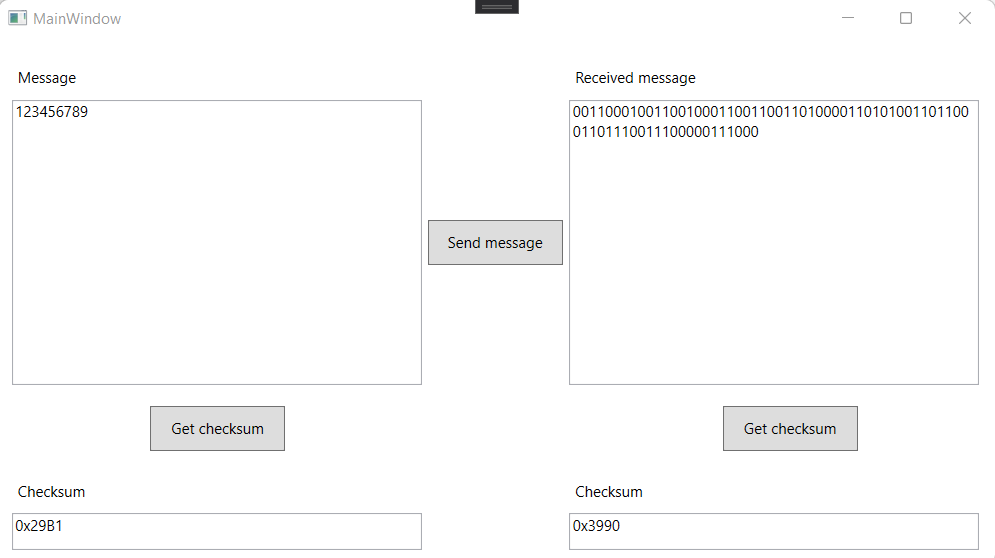


Рисунок 3. Результат изменения полученного сообщения

**ПРИЛОЖЕНИЕ А**

**Листинг А1 – Классы CRC16**

class CRC16

{

static ushort seed = 0xFFFF;

public ushort Checksum(byte[] bytes)

{

ushort crc = seed;

foreach (byte b in bytes)

{

crc = (ushort)((crc << 8) ^ table[(crc >> 8) ^ b]);

}

return crc;

}

static ushort[] table = {

0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,

0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,

0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,

0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,

0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,

0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,

0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,

0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,

0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,

0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,

0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,

0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,

0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,

0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,

0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,

0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,

0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,

0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,

0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,

0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,

0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,

0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,

0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,

0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,

0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,

0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,

0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,

0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,

0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,

0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,

0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,

0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0

};

}

**Листинг А2 – Класс MainWindow**

public partial class MainWindow : Window

{

static CRC16 crc16 = new CRC16();

public MainWindow()

{

InitializeComponent();

}

public static string ToBinary(byte[] data)

{

return string.Join("", data.Select(byt => Convert.ToString(byt, 2).PadLeft(8, '0')));

}

public static byte[] ToBytes(string binary)

{

int numOfBytes = binary.Length / 8;

byte[] bytes = new byte[numOfBytes];

for (int i = 0; i < numOfBytes; ++i)

{

bytes[i] = Convert.ToByte(binary.Substring(8 \* i, 8), 2);

}

return bytes;

}

private ushort calcChecksum(byte[] bytes)

{

return crc16.Checksum(bytes);

}

private void onSendMessage(object sender, RoutedEventArgs e)

{

var sendMessage = messageToSendTextBox.Text;

if (sendMessage != string.Empty)

{

var bytes = Encoding.ASCII.GetBytes(sendMessage);

var recievedMessageBinary = ToBinary(bytes);

messageToRecieveTextBox.Text = recievedMessageBinary;

}

}

private void onChecksumSend(object sender, RoutedEventArgs e)

{

var sendMessage = messageToSendTextBox.Text;

var bytes = Encoding.ASCII.GetBytes(sendMessage);

var chs = calcChecksum(bytes);

firstChecksumTextBox.Text = $"0x{chs:X}";

}

private void onChecksumRecieve(object sender, RoutedEventArgs e)

{

var recievedMessageBinary = messageToRecieveTextBox.Text;

var recievedBytes = ToBytes(recievedMessageBinary);

var chs = calcChecksum(recievedBytes);

secondSendTextBox.Text = $"0x{chs:X}";

}

}

**Листинг А3 – Интерфейс**

<Window x:Class="zi5.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:zi5"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<TextBox x:Name="messageToSendTextBox" HorizontalAlignment="Left" Height="228" Margin="10,50,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="328"/>

<TextBox x:Name="messageToRecieveTextBox" HorizontalAlignment="Left" Height="228" Margin="456,50,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="328"/>

<TextBox x:Name="firstChecksumTextBox" HorizontalAlignment="Left" Height="29" Margin="10,381,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="328"/>

<Label Content="Message" HorizontalAlignment="Left" Margin="10,19,0,0" VerticalAlignment="Top" Width="328"/>

<Label Content="Checksum" HorizontalAlignment="Left" Margin="10,350,0,0" VerticalAlignment="Top" Width="328"/>

<Label Content="Received message" HorizontalAlignment="Left" Margin="456,19,0,0" VerticalAlignment="Top" Width="328"/>

<TextBox x:Name="secondSendTextBox" HorizontalAlignment="Left" Height="29" Margin="456,381,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="328"/>

<Label Content="Checksum" HorizontalAlignment="Left" Margin="456,350,0,0" VerticalAlignment="Top" Width="328"/>

<Button Content="Send message" HorizontalAlignment="Left" Margin="343,146,0,0" VerticalAlignment="Top" Width="108" Height="36" Click="onSendMessage"/>

<Button Content="Get checksum" HorizontalAlignment="Left" Margin="121,295,0,0" VerticalAlignment="Top" Width="108" Height="36" Click="onChecksumSend"/>

<Button Content="Get checksum" HorizontalAlignment="Left" Margin="579,295,0,0" VerticalAlignment="Top" Width="108" Height="36" Click="onChecksumRecieve"/>

</Grid>

</Window>