**Минский колледж предпринимательства**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №18**

**по учебному предмету**

**«ЗАЩИТА КОМПЬЮТЕРНОЙ ИНФОРМАЦИИ»**

*technicaldocs.ru*

*Подпись и дата*

*Взам. инв. №*

*Инв. № дубл.*

*Инв. № подл.*

*Подпись и дата*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | Выполнила учащийся 3 курса группы П-2208 |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А.Ахрамович |
|  |  |  |
|  |  | Преподаватель |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.П.Кислюк |

2024

**Тема учебного занятия:** Криптографические хэш-функции

**Цель учебного занятия:** изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный в варианте алгоритм хэширования.

**Вариант 6**

I.Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие

действия.

1. Вычислять значение хэш-функции, заданной в варианте:

1) текст сообщения должен считываться из файла;

2)полученное значение хэш-функции должно представляться в шестнадцатеричном виде и сохраняться в файл;

3) при работе программы должна быть возможность просмотра и изменения считанного из файла сообщения и вычисленного значения хэш-функции.

2. Исследовать лавинный эффект на сообщении, состоящем из одного блока:

1) для бита, который будет изменяться, приложение должно позволять задавать его позицию (номер) в сообщении;

2) приложение должно уметь после каждого раунда (итерации цикла) вычисления хэш-функции подсчитывать число бит, изменившихся в хэше при изменении одного бита в тексте сообщения;

3) приложение может строить графики зависимости числа бит, изменившихся в хэше, от раунда вычисления хэш-функции, либо графики можно строить в стороннем ПО, но тогда приложение должно сохранять в файл необходимую для построения графиков информацию.

II. С помощью реализованного приложения выполнить следующие задания.

1. Протестировать правильность работы разработанного приложения.

2. Исследовать лавинный эффект при изменении одного бита в сообщении: для различных позиций изменяемого бита в сообщении построить графики зависимостей числа бит, изменившихся в хэше, от раунда вычисления хэш-функции (всего в отчѐте должно быть 2–3 графика).

3. Сделать выводы о проделанной работе.

**Ход работы**

Была разработана программа по заданному условию.

**Листинг программы**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

using System.Windows.Forms.DataVisualization;

using System.Security.Cryptography;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace RIPEMD160

{

public partial class Form1 : Form

{

List<int> changedBitsPerRound = new List<int>();

private string previousHash = "";

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

string filePath = openFileDialog.FileName;

string text = File.ReadAllText(filePath);

textBox1.Text = text;

}

}

private string ComputeRIPEMD160Hash(string input)

{

using (System.Security.Cryptography.RIPEMD160 ripemd = System.Security.Cryptography.RIPEMD160.Create())

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

byte[] hashBytes = ripemd.ComputeHash(inputBytes);

return BitConverter.ToString(hashBytes).Replace("-", "").ToLower();

}

}

private string HexToBinary(string hex)

{

StringBuilder binary = new StringBuilder();

foreach (char hexChar in hex)

{

binary.Append(Convert.ToString(Convert.ToInt32(hexChar.ToString(), 16), 2).PadLeft(4, '0'));

}

return binary.ToString();

}

private int CountChangedBits(string hash1, string hash2)

{

if (string.IsNullOrEmpty(hash1) || string.IsNullOrEmpty(hash2))

{

MessageBox.Show("Не все хеши загружены.");

return -1;

}

string binaryHash1 = HexToBinary(hash1);

string binaryHash2 = HexToBinary(hash2);

if (binaryHash1.Length != binaryHash2.Length)

{

MessageBox.Show("Длины хэшей не совпадают.");

return -1;

}

int changedBits = 0;

for (int i = 0; i < binaryHash1.Length; i++)

{

if (binaryHash1[i] != binaryHash2[i])

{

changedBits++;

}

}

return changedBits;

}

private string FlipBit(string input, int position)

{

string binaryString = string.Join("", input.Select(c => Convert.ToString(c, 2).PadLeft(8, '0')));

if (position < 0 || position >= binaryString.Length)

{

MessageBox.Show("Неверная позиция бита.");

return input;

}

char[] binaryArray = binaryString.ToCharArray();

binaryArray[position] = binaryArray[position] == '0' ? '1' : '0';

string modifiedBinaryString = new string(binaryArray);

StringBuilder modifiedString = new StringBuilder();

for (int i = 0; i < modifiedBinaryString.Length; i += 8)

{

string byteStr = modifiedBinaryString.Substring(i, 8);

modifiedString.Append((char)Convert.ToByte(byteStr, 2));

}

return modifiedString.ToString();

}

private void button3\_Click(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(textBox1.Text))

{

textBox1.ReadOnly = false;

}

}

private void button6\_Click(object sender, EventArgs e)

{

UpdateGraph();

}

private void UpdateGraph()

{

chart1.Series.Clear();

chart1.ChartAreas.Clear();

ChartArea chartArea = new ChartArea("ChartArea1");

chart1.ChartAreas.Add(chartArea);

Series series = new Series("Изменённые биты");

series.ChartType = SeriesChartType.Line;

if (listBox1.Items.Count > 0)

{

for (int i = 0; i < listBox1.Items.Count; i++)

{

string item = listBox1.Items[i].ToString();

string[] parts = item.Split(new string[] { "Изменено бит: " }, StringSplitOptions.None);

if (parts.Length > 1 && int.TryParse(parts[1], out int changedBits))

{

series.Points.AddXY(i + 1, changedBits);

}

}

}

else

{

MessageBox.Show("Нет данных для отображения на графике.");

}

chart1.Series.Add(series);

chart1.Invalidate();

}

private void button2\_Click(object sender, EventArgs e)

{

SaveHashToFile(textBox2.Text);

}

private void button4\_Click(object sender, EventArgs e)

{

string newHash = ComputeRIPEMD160Hash(textBox1.Text);

string oldHash = textBox2.Text;

textBox2.Text = newHash;

if (!string.IsNullOrEmpty(oldHash))

{

int changedBits = CountChangedBits(oldHash, newHash);

listBox1.Items.Add($"Изменено бит: {changedBits}");

}

previousHash = newHash;

}

private void button5\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBox1.Text))

{

MessageBox.Show("Введите сообщение");

return;

}

string lastHash = LoadLastHashFromFile();

if (string.IsNullOrEmpty(lastHash))

{

MessageBox.Show("Не найден предыдущий хеш для сравнения.");

return;

}

int position = (int)numericUpDown1.Value;

if (position < 0 || position >= textBox1.Text.Length \* 8)

{

MessageBox.Show("Неверная позиция бита");

return;

}

string text = textBox1.Text;

text = FlipBit(text, position);

textBox1.Text = text;

string newHash = ComputeRIPEMD160Hash(text);

textBox2.Text = newHash;

int changedBits = CountChangedBits(lastHash, newHash);

listBox1.Items.Add($"Изменено бит: {changedBits}");

SaveHashToFile(newHash);

previousHash = newHash;

}

private string LoadLastHashFromFile()

{

string filePath = "hashes.txt";

if (File.Exists(filePath))

{

string[] hashes = File.ReadAllLines(filePath);

if (hashes.Length > 0)

{

return hashes[hashes.Length - 1];

}

}

return null;

}

private void SaveHashToFile(string newHash)

{

string filePath = "hashes.txt";

File.AppendAllText(filePath, newHash + Environment.NewLine);

}

}

}

**Тесты:**

На рисунке 1 изображено вычисление хеша.

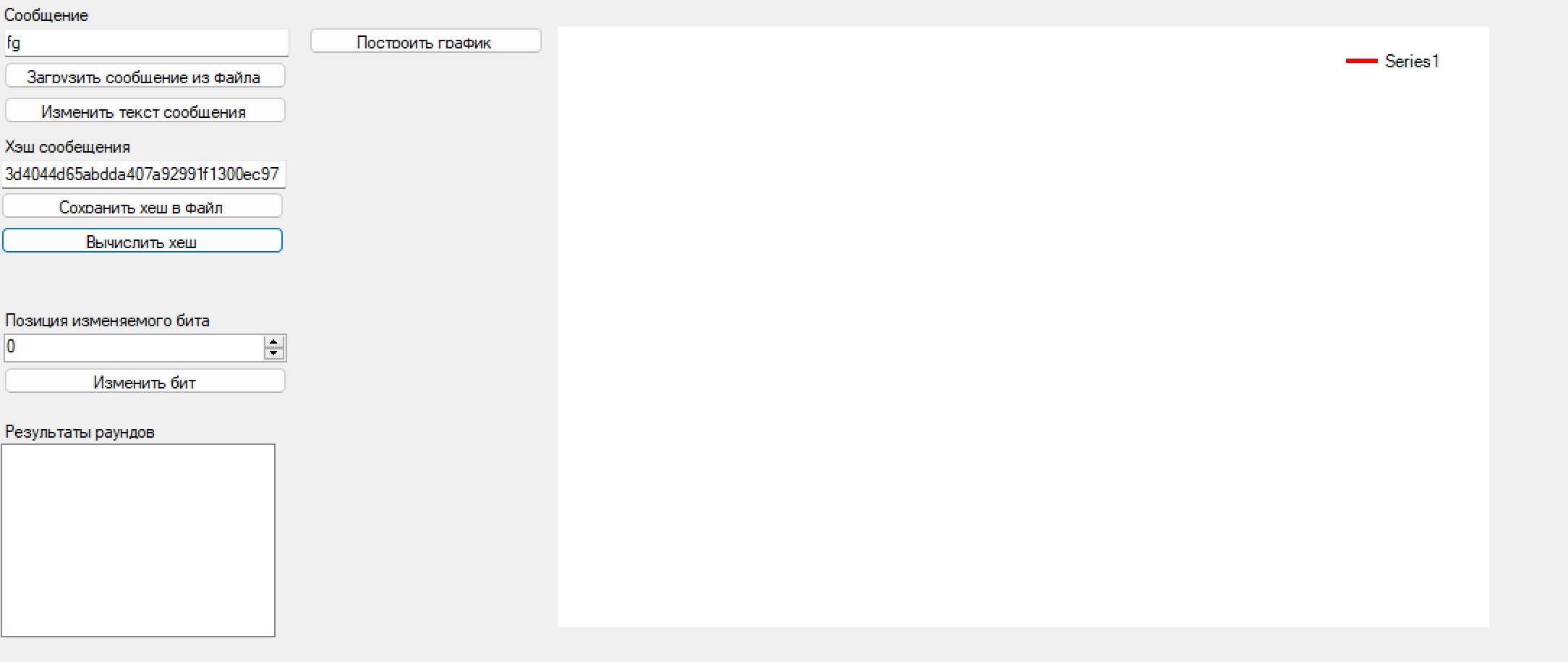


Рис. 1. Вычисление хеша.

На рисунке 2 изображено изменение бита.

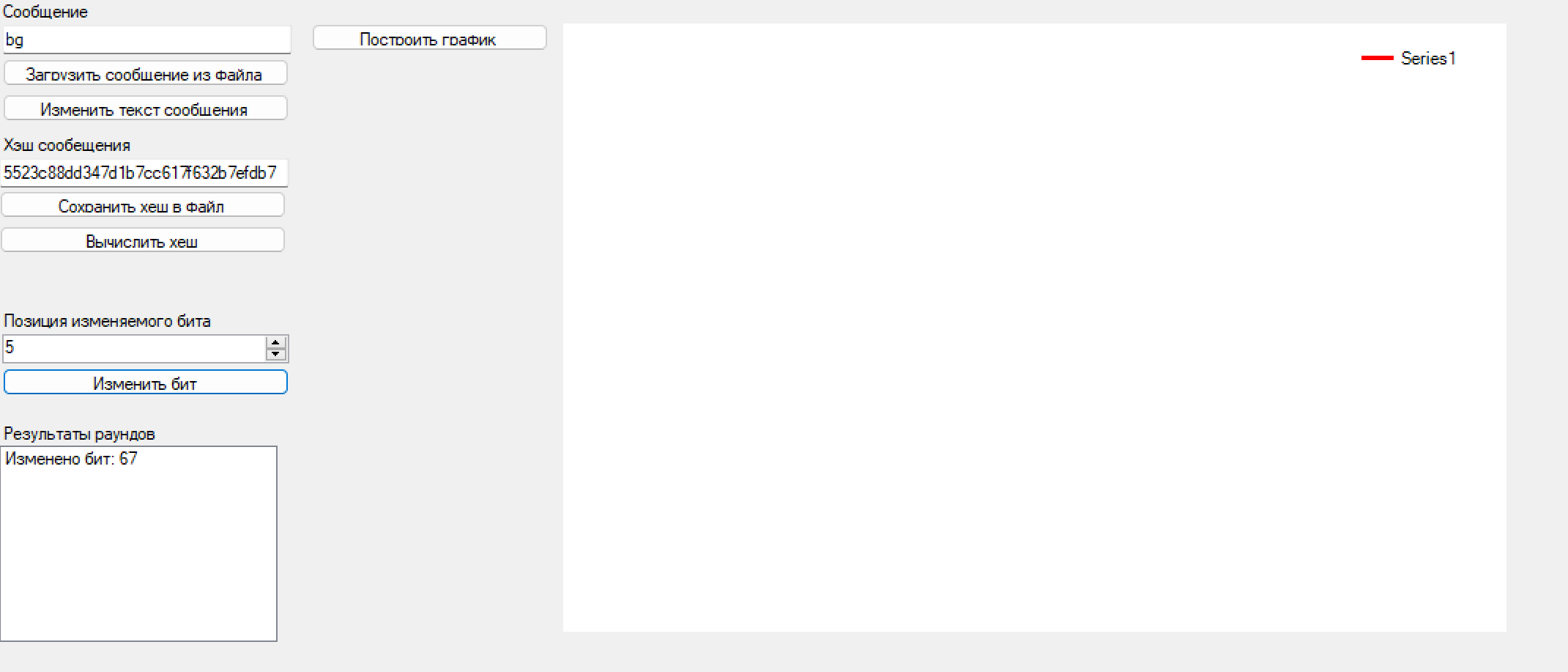


Рис. 2. Изменение бита.

На рисунке 3 изображено построение графика.

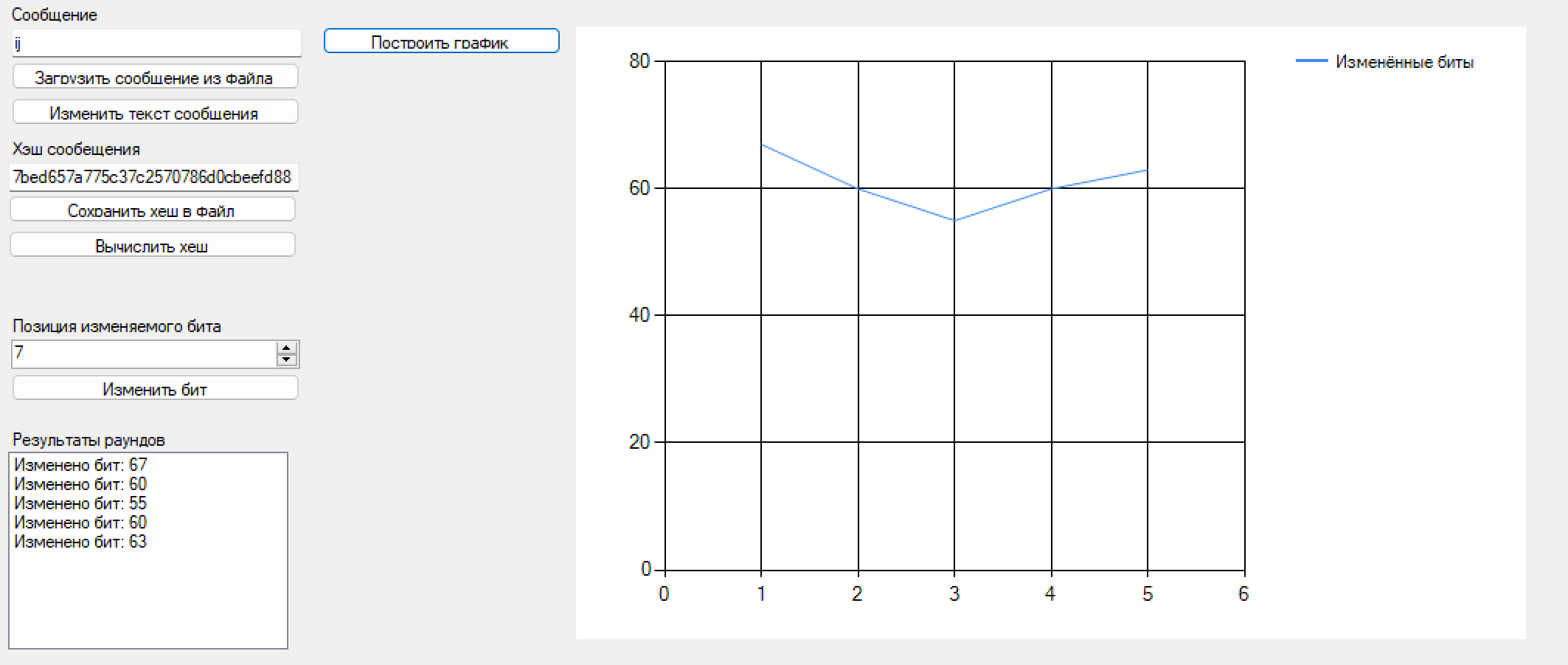


Рис. 3.2. Построение графика.

**Вывод:** изучили существующие алгоритмы вычисления дайджестов сообщений и написали программу, реализующую заданный в варианте алгоритм хэширования.