Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

Лабораторная работа № 2

по дисциплине «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

КРИПТОГРАФИЧЕСКИЕ ХЭШ-ФУНКЦИИ

Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	6
Студенты:	Сидоров Даниил,
	Дюков Богдан
Преподаватели:	Авдеенко Татьяна Владимировна,
	Кутузова Ирина Александровна.

Новосибирск

2026

1. Цель работы

Изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный в варианте алгоритм хэширования.

2. Задача

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 отчете должно быть 2– 3 графика).
3. Сделать выводы о проделанной работе.

Вариант	Алгоритм
6	RIPEMD–160

3. Метод решения задачи

1) Алгоритм RIPEMD–160

RIPEMD-160 - криптографическая хеш-функция, генерирующая 160-разрядное хеш-значение для произвольного входного сообщения.

Описание каждого шага алгоритма:

Пусть имеется сообщение M длиной в b бит, хэш-функцию которого нужно вычислить.

1. **Добавление дополнительных бит**. Сообщение дополняется до нужной длины. Сначала к сообщению в двоичном представлении добавляется единичный бит, затем добавляются нулевые биты до тех пор, пока остаток от деления числа бит дополненного сообщения на 512 не даст 448.
2. **Преобразование к little-endian**. Каждое 4-байтовое слово в сообщении преобразуется к формату little-endian, то есть к порядку байтов, при котором младшие байты ставятся вперед (левее).
3. **Добавление исходной длины сообщения**. К сообщению добавляются младшие 64 бита из b (исходная длина сообщения). При этом сначала дописываются младшие 4 байта, а затем – старшие. В итоге получается сообщение, длина которого кратна 512 битам, т.е. полученное сообщение можно разбить на блоки, каждый из которых представляется в виде шестнадцати 32-битных слов.
4. **Определение констант и используемых функций**. Определяются константы и функции, которые будут использоваться в основном цикле алгоритма. Также инициализируются так называемые переменные состояния хэш-функции (h0, h1, h2, h3, h4).
5. **Основной цикл**. Здесь происходит основная часть работы алгоритма. Каждый 512-битный блок сообщения обрабатывается в цикле, в котором выполняется 80 раундов различных операций, таких как сложение, циклический сдвиг, побитовые операции. После каждого обработанного блока обновляется состояние хэш-функции. Таким образом, обработав каждый блок сообщения, мы получаем промежуточные значения хэша, которые хранятся в переменных h0, h1, h2, h3, h4.
6. **Преобразование частей хэша в результат**. Полученные значения хэша объединяются в одну шестнадцатеричную строку, представляющую собой итоговый хэш.

2) **Лавинный эффект**

Для криптографических хэш-функций важно, чтобы при малейшем изменении аргумента значение функции сильно изменялось. Это свойство называется **лавинным эффектом**.

Хороший алгоритм шифрования всегда должен удовлетворять следующему соотношению:

Avalanche effect > 50%

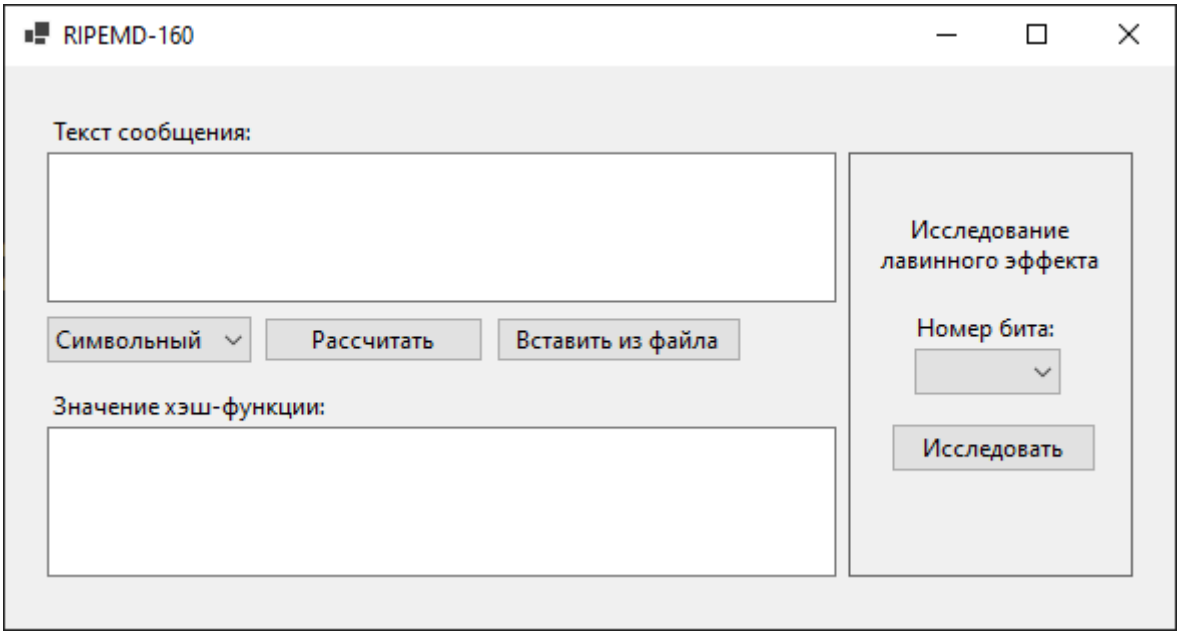
Эффект гарантирует, что злоумышленник не сможет легко предсказать обычный текст с помощью статистического анализа.

В нашем приложении мы вычисляем значения хэш-функций исходного и модифицированного (с одним измененным битом) сообщений в каждом раунде алгоритма. Для каждых соответствующих хэшей вычисляется число различий в битах. По итогу получается список пар вида: (Номер раунда, Число различий), по которым можно построить график и оценить криптостойкость алгоритма хеширования.

4. Разработанное программное средство

Разработанное программное средство представляет собой приложение Windows Forms.

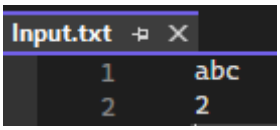
Интерфейс главного меню:



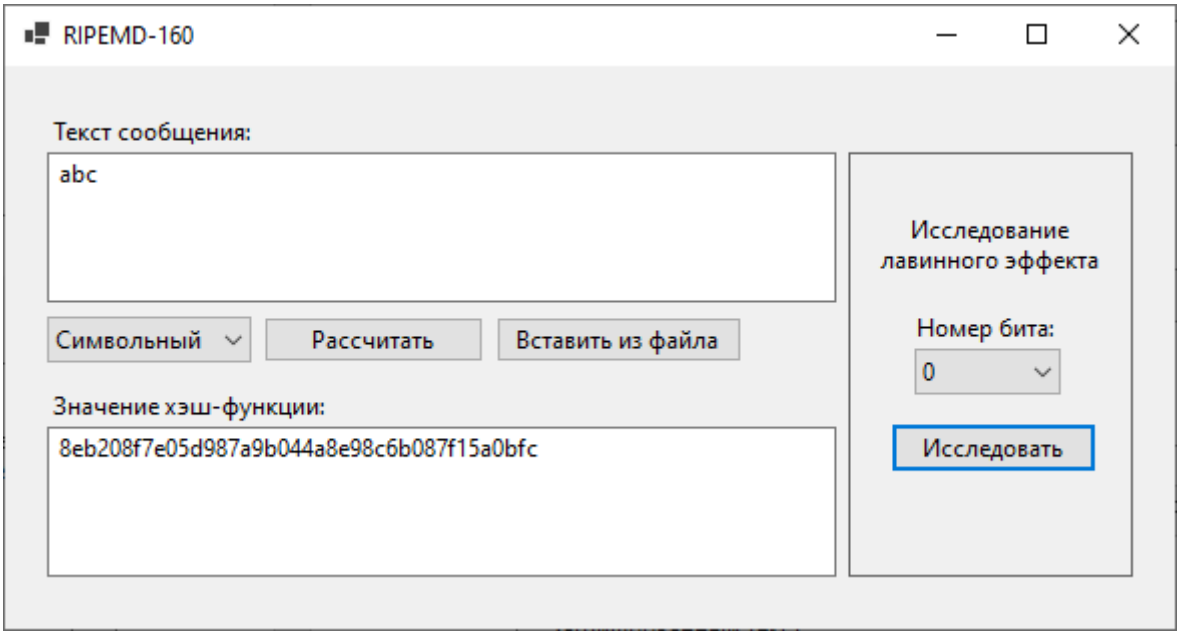
Он содержит текстовые поля для исходного сообщения и полученного значения хэш-функции. Имеется переключатель для просмотра и изменения исходного сообщения в двоичном, шестнадцатеричном и символьном видах. Предусмотрена возможность вставки из файла исходного сообщения.

Также в главном меню есть окно для исследования лавинного эффекта. Выбрав номер бита из списка и нажав кнопку “Исследовать”, откроется отдельная форма с графиком зависимости числа бит, изменившихся в хэше, от раунда вычисления хэш-функции.

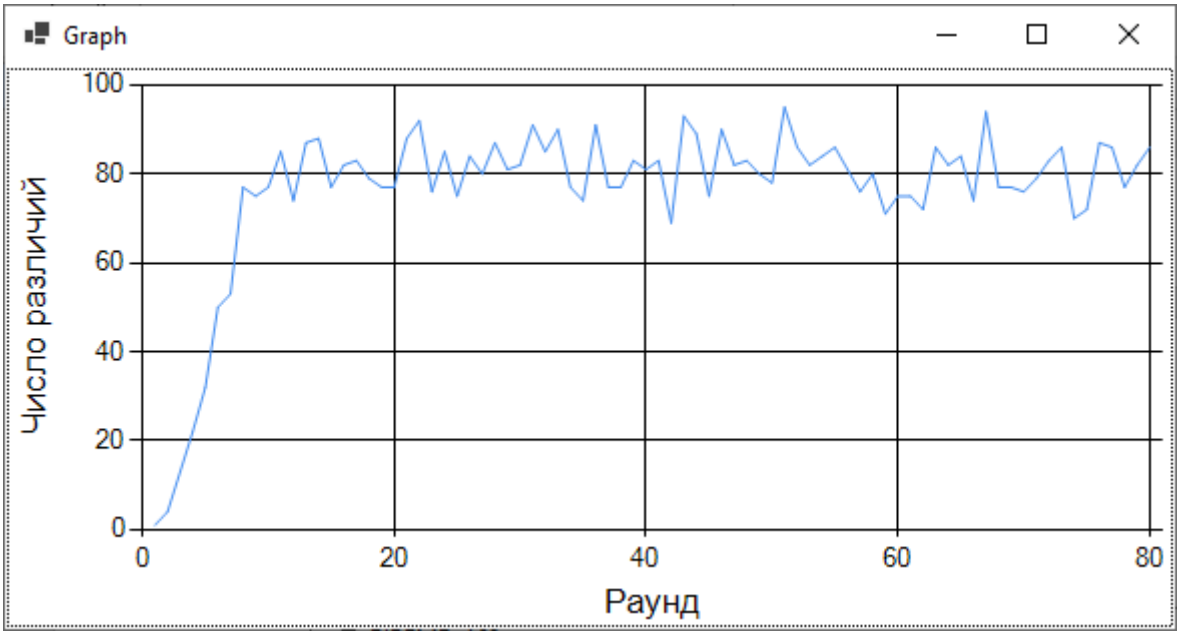
Рассмотрим тестовый пример из методических указаний. Выполним вставку сообщения из файла, генерацию хэша и исследование лавинного эффекта (изменяем бит под номером 0):



(2 означает индекс символьного формата)



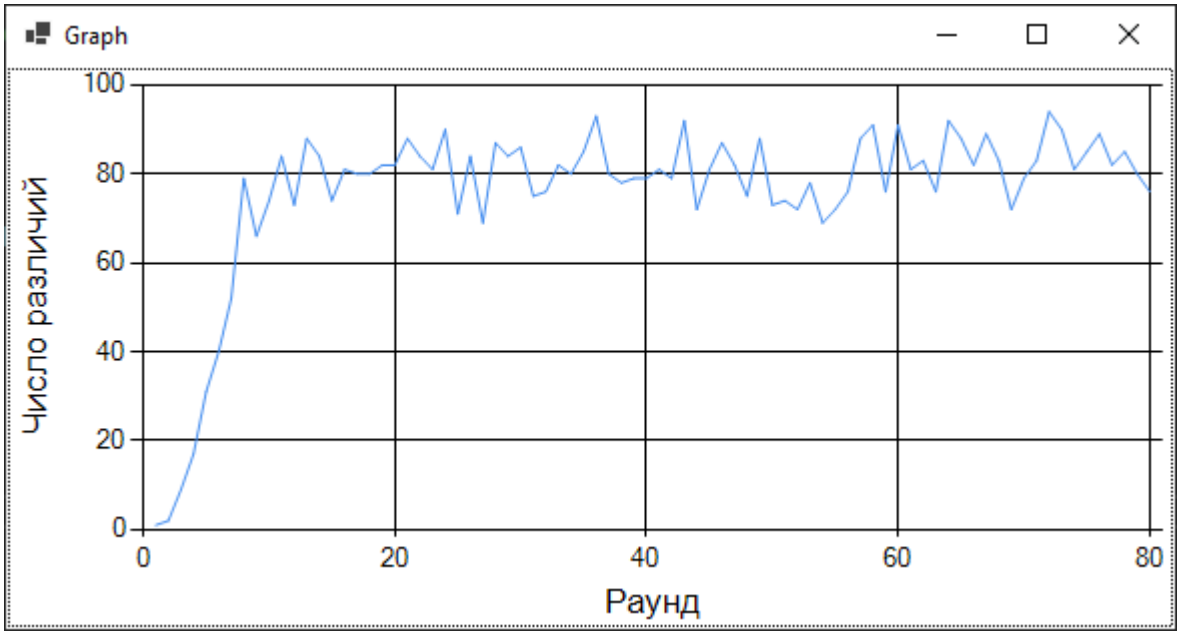
Отметим, что текст исходного сообщения и полученное значения хэш-функции сохраняются в файлы (input.txt и output.txt соответственно). Исследование:



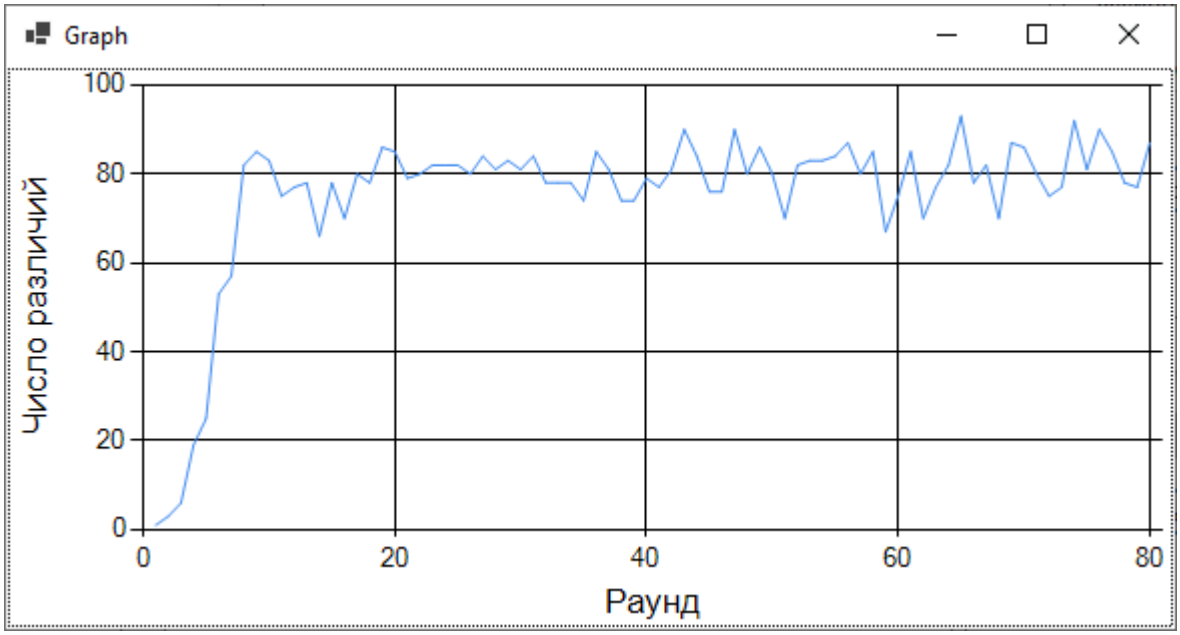
Значение хэш-функции совпадает со значением из методических указаний. А на основе графика можно сказать, что в среднем число различий в битах, начиная с ~7 раунда, колеблется у отметки в 50% (сравниваются 160 битные сообщения).

5. Исследование лавинного эффекта

Продолжим построение графиков для тестового сообщения “abc”, меняя номер изменяемого бита. Изменяя бит под номером 11:



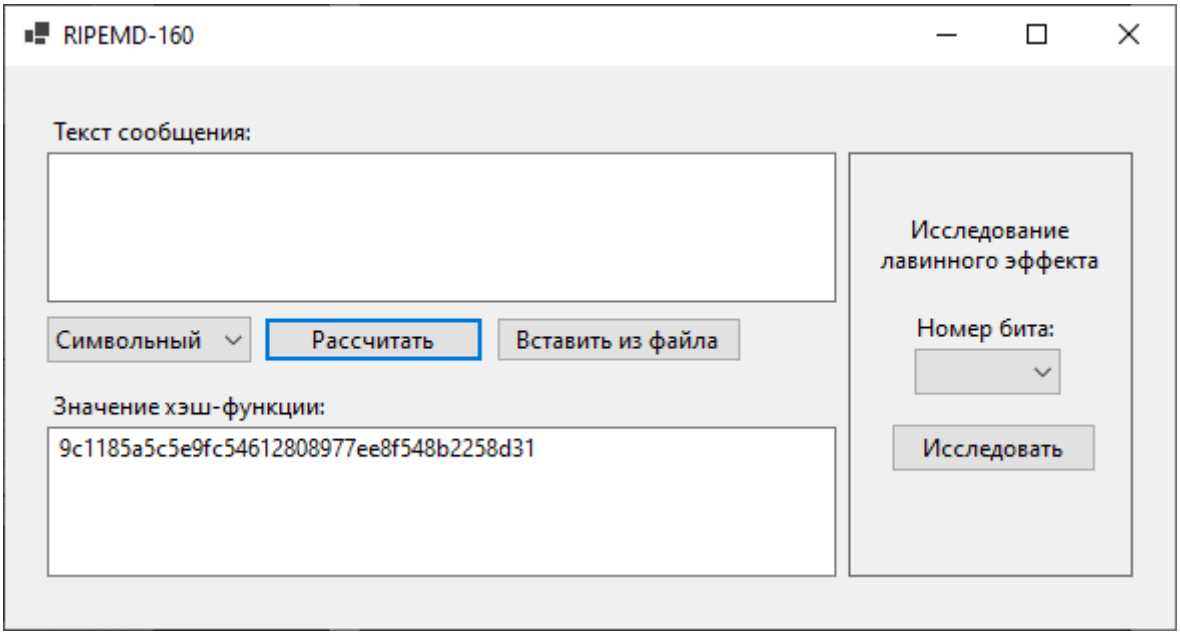
Изменяя бит под номером 23:



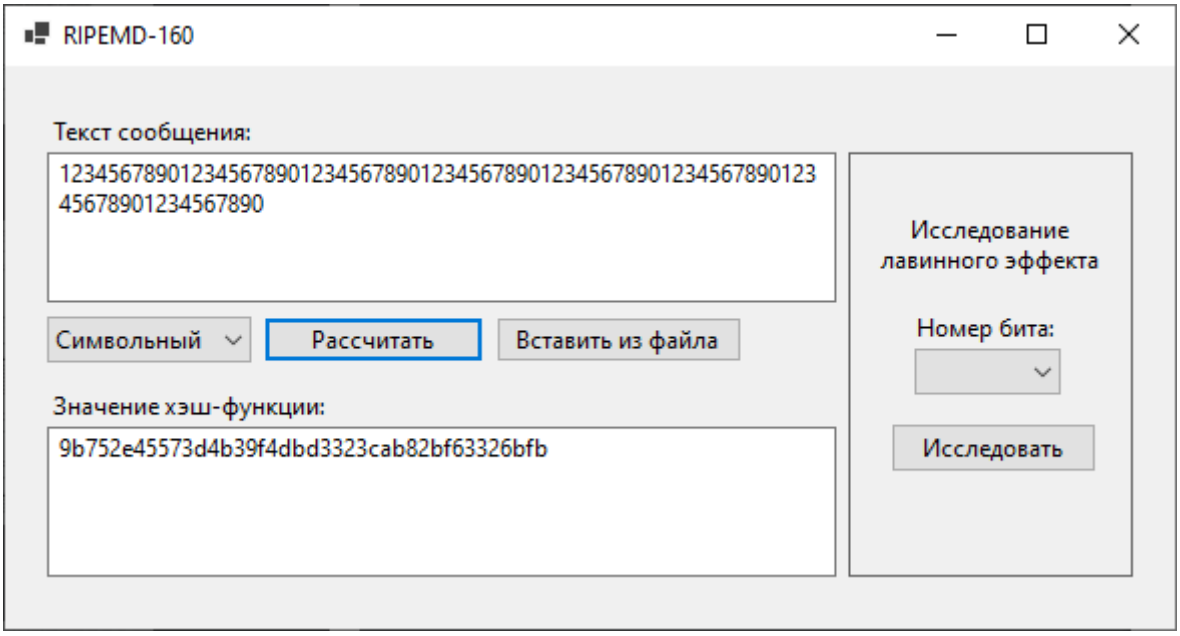
Исследование показало общую закономерность: при изменении одного бита в исходном сообщении, в хэше, начиная с 7 раунда, происходит изменение от 69 до 95 битов. Учитывая, что общая длина сообщения в двоичном представлении составляет 160 битов, процентное соотношение различий в битах составляет от 40% до 60%. Это подчеркивает эффективность и надежность алгоритма хеширования, который обеспечивает высокую степень чувствительности к любым изменениям в исходных данных.

6. Тестирование

Вычислим значения хэш-функций для оставшихся двух тестовых примеров из методических указаний. Для пустого сообщения:



И для сообщения с числовой последовательностью:



Результаты в точности совпадают.

7. Вывод

В ходе проведения лабораторной работы, мы освоили важный метод криптографии - алгоритм хеширования RIPEMD-160, а также исследовали его на лавинный эффект.

Алгоритм RIPEMD-160 является криптографической хеш-функцией, генерирующей 160-разрядное хеш-значение для произвольного входного сообщения. Это делает его идеальным выбором для обеспечения целостности данных в критически важных приложениях. Наша реализация включает все ключевые этапы алгоритма, включая добавление дополнительных бит, преобразование к little-endian, добавление исходной длины сообщения,

определение констант и используемых функций, основной цикл и преобразование частей хэша в результат.

Также мы провели исследование лавинного эффекта на сообщении, состоящем из одного блока. Это важное свойство для криптографических хеш-функций, которое гарантирует, что при малейшем изменении аргумента значение функции сильно изменяется. Мы вычислили значения хэш-функций исходного и модифицированного сообщений в каждом раунде алгоритма и подсчитали число различий в битах. Для наглядности построили графики зависимости числа бит, изменившихся в хэше, от раунда вычисления хэш-функции. Это позволило нам оценить криптостойкость алгоритма хеширования.