

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**  
**Федеральное государственное бюджетное образовательное**  
**учреждение высшего образования**  
**«Московский Авиационный Институт»**  
**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии**  
**и прикладная математика»**  
**Кафедра: 806 «Вычислительная математика**  
**и программирование»**

Курсовой проект  
по курсу «Криптография»

Группа: М8О-307Б-21

Студент: Ф. А. Меркулов

Преподаватель: А. В. Борисов

Оценка:

Дата: 10.05.2024

Москва, 2024

## **Оглавление**

<b>1</b>	<b>Тема .....</b>	<b>3</b>
<b>2</b>	<b>Задание.....</b>	<b>3</b>
<b>3</b>	<b>Получение варианта .....</b>	<b>4</b>
<b>4</b>	<b>Реализация алгоритма согласно варианту .....</b>	<b>6</b>
<b>5</b>	<b>Модификация количества раундов алгоритма.....</b>	<b>8</b>
<b>6</b>	<b>Дифференциальный криптоанализ.....</b>	<b>9</b>
<b>7</b>	<b>Выводы .....</b>	<b>15</b>
<b>8</b>	<b>Список источников .....</b>	<b>16</b>

# 1 Тема

Функции хэширования.

## 2 Задание

№0. Строку в которой записано своё ФИО подать на вход в хэш-функцию ГОСТ Р 34.11-2012 (Стрибог). Младшие 4 бита выхода интерпретировать как 16-тиричное число, которое в дальнейшем будет номером варианта.

№1. Программно реализовать один из алгоритмов функции хэширования в соответствии с номером варианта. Алгоритм содержит в себе несколько раундов.

№2. Модифицировать оригинальный алгоритм таким образом, чтобы количество раундов было настраиваемым параметром программы. в этом случае новый алгоритм не будет являться стандартом, но будет интересен для исследования.

№3. Применить подходы дифференциального криптоанализа к полученным алгоритмам с разным числом раундов.

№4. Построить график зависимости количества раундов и возможности различения отдельных бит при количестве раундов 1,2,3,4,5,... .

№5. Сделать выводы.

Примечание №1. Допустимо использовать сторонние реализации для пункта 1, при условии, что они проходят тесты из стандарта и пригодны для дальнейшей модификации.

Примечание №2. Если в алгоритме описывается семейство с разными размерами блоков, то можно выбрать любой из них.

### 3 Получение варианта

У меня возникли сложности с получением варианта обычным способом и в итоге я использовал библиотеку gostcrypto:

```
pip install gostcrypto
```

Collecting gostcrypto  
Downloading gostcrypto-1.2.5-py3-none-any.whl (82 kB)  
82.4/82.4 kB 2.0 MB/s eta 0:00:00  
Installing collected packages: gostcrypto  
Successfully installed gostcrypto-1.2.5

```
[ ] import gostcrypto

hash_string = u'Меркулов Фёдор Алексеевич'.encode('utf8')
hash_obj = gostcrypto.gosthash.new('streebog256', data=hash_string)
hash_result = hash_obj.hexdigest()
print(hash_result)
```

2e92b44e40b2dfd88e513ee15083fb66ac923a9f20c07fdd4d5d4ba95a2f91f8

Значит мой вариант 8 = JH

На всякий случай покажу, что вариант определяется корректно:

1) Как было в примере:  
from pygost import gost34112012256  
gost34112012256.new("Иванов Иван Иванович").digest();  
'\xe6\xe1\xfb5H\x94\x8e\xeah@\xe6Pl\xa4&\xfb\x12za-  
A\xa6\x08\xc2m\xfe\xf9L[\x94N4\xbe'; # => вариант 'e'

Какой вариант получает моя реализация:

```
import gostcrypto

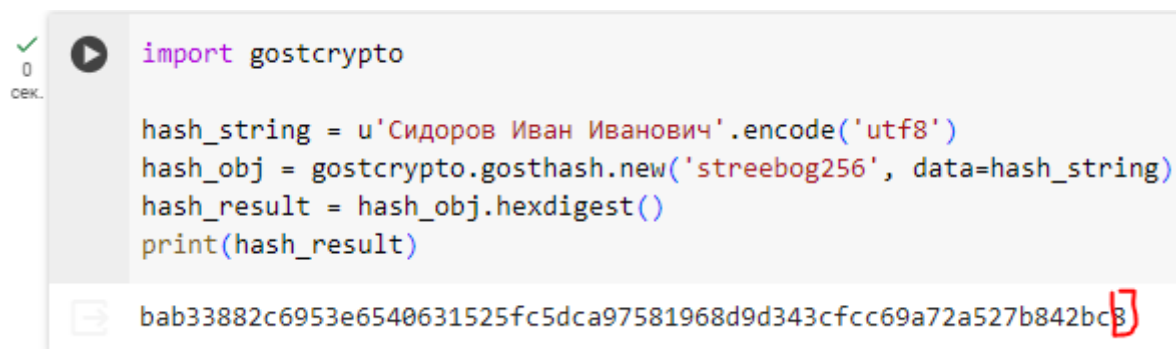
hash_string = u'Иванов Иван Иванович'.encode('utf8')
hash_obj = gostcrypto.gosthash.new('streebog256', data=hash_string)
hash_result = hash_obj.hexdigest()
print(hash_result)
```

e6e1f548948eea6840e6506ca426fb127a612d41a608c26dfef94c5b944e34be

2) Какой был в примере:

gost34112012256.new("Сидоров Иван Иванович").digest();  
'\xba\x38\x82\xc6\x95>e@c\x15%\xfc]\xca\x97X\x19h\xd9\xd3C\xcf\xcc\x7\*R  
{\x84+\xc8' # => вариант '8'

Какой вариант получает моя реализация:



The screenshot shows a code editor with a green checkmark and a play button icon. The code is as follows:

```
import gostcrypto

hash_string = u'Сидоров Иван Иванович'.encode('utf8')
hash_obj = gostcrypto.gosthash.new('streebog256', data=hash_string)
hash_result = hash_obj.hexdigest()
print(hash_result)
```

Below the code, the output is displayed: `bab33882c6953e6540631525fc5dca97581968d9d343cfcc69a72a527b842bc8`. The last character '8' is highlighted with a red bracket.

Именно из-за этого я считаю, что определил вариант корректно и выполнял работу 8 варианта, то есть исследовал функцию хэширования JH.

## 4 Реализация алгоритма согласно варианту

Алгоритм JH представляет собой семейство криптографических хэш-функций, включающее варианты JH-224, JH-256, JH-384 и JH-512, различающиеся длиной выходного хэша. Важным аспектом JH является его адаптивность и эффективность при аппаратной и программной реализации, что было ключевым фактором при его выборе в качестве финалиста второго тура конкурса SHA-3. Алгоритм использовал переменный параметр  $d$ , который позволял настраивать уровень криптостойкости, делая его гибким инструментом для разработки в различных средах. Последняя версия, JH42, отличается увеличением количества раундов в функции компрессии до 42, что повысило его криптографическую надежность, но мне потребовалось модифицировать оригинальный алгоритм таким образом, чтобы количество раундов было настраиваемым параметром программы. В этом случае новый алгоритм уже не является стандартом, но интересен для исследования.

В спецификации JH используются следующие параметры:

- $Cr(d)$ : Константы раунда, используемые в функции  $Ed$  для  $0 \leq r < 6 \times (d-1)$ . Каждая  $Cr(d)$  является 24-битной константой.
- $d$ : Размерность блока битов. Блок состоит из  $2d$  4-битных элементов.
- $h$ : Число битов в хэш-значении,  $h=1024$ .
- $H(i)$ :  $i$ -е хэш-значение размером  $h$  бит.  $H(0)$  — это начальное хэш-значение;  $H(N)$  — окончательное хэш-значение и усечённое значение для получения сообщения дайджеста.
- $H(i)j$ :  $j$ -й бит  $i$ -го хэш-значения, где  $H(i) = H(i)0 \| H(i)1 \| \dots \| H(i)h-1$ .
- $\ell$ : Длина сообщения  $M$ , в битах.
- $m$ : Число бит в блоке сообщения  $M(i)$ .  $m=512$ .
- $M$ : Сообщение для хэширования.
- $M(i)$ : Блок сообщения  $i$  размером  $m$  бит.
- $M(i)j$ :  $j$ -й бит  $i$ -го блока сообщения, то есть  $M(i) = M(i)0 \| M(i)1 \| \dots \| M(i)m-1$ .
- $N$ : Число блоков в дополненном сообщении.

Хэш-функция JH состоит из пяти этапов: добавление в сообщение  $M$  дополнительных бит (раздел 4.1), разбиение дополненного сообщения на блоки (раздел 4.2), установка начального значения хэша  $H(0)$  (раздел 4.3), вычисление окончательного значения хэша  $H(N)$  (раздел 4.4) и генерация дайджеста сообщения путем усечения  $H(N)$  (раздел 4.5).

### 4.1 Дополнение сообщения

Сообщение  $M$  дополняется так, чтобы его длина стала кратна 512 битам. Предположим, что длина исходного сообщения  $M$  составляет  $\ell$  бит. К

сообщению добавляется бит "1", за которым следуют  $384-1+(\ell \bmod 512)$  нулевых бит (для  $\ell \bmod 512 = 0, 1, 2, \dots, 510, 511$ , количество добавляемых нулевых бит равно 383, 384, 893, ..., 385, 384 соответственно), а затем добавляется 128-битный блок, который равен числу  $\ell$  в двоичном виде с обратным порядком байт. Таким образом, к сообщению  $M$  добавляется как минимум 512 дополнительных бит.

#### 4.2 Разбиение дополненного сообщения

После дополнения сообщение разбивается на блоки по  $N$  512-битных блоков,  $M(1), M(2), \dots, M(N)$ . 512-битный блок сообщения представляется в виде четырех 128-битных слов. Первые 128 бит блока сообщения обозначаются как  $M(i)0$ , следующие 128 бит — как  $M(i)1$ , и так далее до  $M(i)3$ .

#### 4.3 Установка начального значения хэша $H(0)$

Начальное значение хэша  $H(0)$  устанавливается в зависимости от размера сообщения. Первые два байта  $H(-1)$  устанавливаются как размер сообщения, а остальные байты  $H(-1)$  устанавливаются как 0. Затем  $H(0) = F8(H(-1), M(0))$ .

Более детально, значения начальных параметров  $H(-1)j$  для  $H(0)$  для JH-224, JH-256, JH-384 и JH-512 соответственно устанавливаются как 0x00E0, 0x0100, 0x0180, 0x0200. Пусть  $H(-1)j = 0$  для  $16 \leq j \leq 1023$ . Начальное значение хэша  $H(0)$  вычисляется как:

$$H(0) = F8(H(-1), M(0)).$$

#### 4.4 Вычисление окончательного хэш-значения $H(N)$

Функция сжатия  $F8$  применяется для генерации  $H(N)$  путём последовательного сжатия блоков  $M(1), M(2), \dots, M(N)$ . Окончательное 1024-битное хэш-значение  $H(N)$  вычисляется следующим образом:

для  $i=1$  до  $N$ ,

$$H(i) = F8(H(i-1), M(i)).$$

#### 4.5 Генерация дайджеста сообщения

Дайджест сообщения генерируется путём усечения  $H(N)$ .

## 5 Модификация количества раундов алгоритма

Для того чтобы количество раундов было не константным (42 в данной реализации), а изменяемым параметром нужно было изменить всего пару мест:

```
/*perform 42 rounds*/  
for (i = 0; i < 42; i = i + 7)
```

заменён на:

```
/*perform the set number of rounds*/  
for (i = 0; i < state->rounds; i = i + 7)
```

И соответственно в структуру добавлена переменная, отвечающая за количество раундов:

```
typedef struct  
{  
    int hashbitlen; /*the message digest size*/  
    unsigned long long databitlen; /*the message size in bits*/  
    unsigned long long datasize_in_buffer; /*the size of the message remained  
    in buffer; assumed to be multiple of 8bits except for the last partial block  
    at the end of the message*/  
    word128 x0, x1, x2, x3, x4, x5, x6, x7; /*1024-bit state;*/  
    unsigned char buffer[64]; /*512-bit message block;*/  
    int rounds;  
} hashState;
```



## 6 Дифференциальный криптоанализ

Метод дифференциального криптоанализа применяется для оценки влияния малых изменений во входных данных на результаты хэширования в криптографических системах. В ходе данной работы был проведён ряд экспериментов, целью которых было исследование эффекта различного числа раундов шифрования на изменения выходных данных.

Для экспериментов был модифицирован алгоритм хэширования JH, реализованный на языке C. Программа позволяет задавать количество раундов и количество битов, которые будут изменены, для анализа. В процессе работы программа принимает от пользователя входное сообщение и параметры эксперимента: количество битов и раундов для проверки.

В каждом раунде шифрования использовалась модифицированная функция хэширования JH, вызываемая дважды: с оригинальным сообщением и с сообщением, в котором были изменены случайно выбранные биты. Затем, программа анализирует и сохраняет результаты, вычисляя расстояние Хэмминга - количество изменённых битов между двумя результатами хэширования.

Функции, реализованные для данного исследования:

```
void computeHash(char *text, int roundCount, unsigned char *hashResult){
    hashState state;
    Init(&state, 256, roundCount);
    Update(&state, (unsigned char *)text, strlen(text));
    Final(&state, hashResult);
}

void changingBits(char *text, int size, int bits, char *result){
    int IndexOfBit = 0;
    for (int i = 0; i < size; i++){
        result[i] = text[i];
    }
    for (int i = 0; i < size && IndexOfBit < bits; i++){
        for (int j = 0; j < 8 && IndexOfBit < bits; j++){
            if ((result[i] & (1 << j)) == 0){
                result[i] |= (1 << j);
                IndexOfBit++;
            } else {
                result[i] &= ~(1 << j);
                IndexOfBit++;
            }
        }
    }
}
```

```

int countDifferenceInBits(const char *str1, const char *str2, int size){
    int count = 0;
    for (int i = 0; i < size; i++){
        unsigned char diff = str1[i] ^ str2[i];
        for (int j = 0; j < 8; j++){
            count += diff & (1 << j) ? 1 : 0;
        }
    }
    return count;
}

void compareHashes(char *text, int bitsToChange, int roundCount) {
    unsigned char originalHash[32];
    unsigned char modifiedHash[32];
    char modifiedText[strlen(text)];

    for (int i = 1; i <= bitsToChange; i++){
        for (int j = 1; j <= roundCount; j++){
            computeHash(text, j, originalHash);
            changingBits(text, strlen(text), i, modifiedText);
            computeHash(modifiedText, j, modifiedHash);
            int diffBits = countDifferenceInBits((const char *)originalHash, (const char *)modifiedHash, 32);
            printf("%d,%d,%d\n", i, j, diffBits);
        }
    }
}

int main(){
    int maximumNumberOfRounds, maximumNumberOfBitsToChange;
    char text[(int)1e3];

    printf("Максимальное количество раундов: ");
    scanf("%d", &maximumNumberOfRounds);
    printf("Максимальное количество битов которые необходимо менять: ");
    scanf("%d", &maximumNumberOfBitsToChange);
    printf("Текст для хэширования: ");
    scanf("%[^\n]s", text);

    compareHashes(text, maximumNumberOfBitsToChange, maximumNumberOfRounds);

    return 0;
}

```

Не модифицированная версия JH42 доступна по ссылке:

[https://www3.ntu.edu.sg/home/wuhj/research/jh/jh\\_sse2\\_opt64.h](https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_sse2_opt64.h)

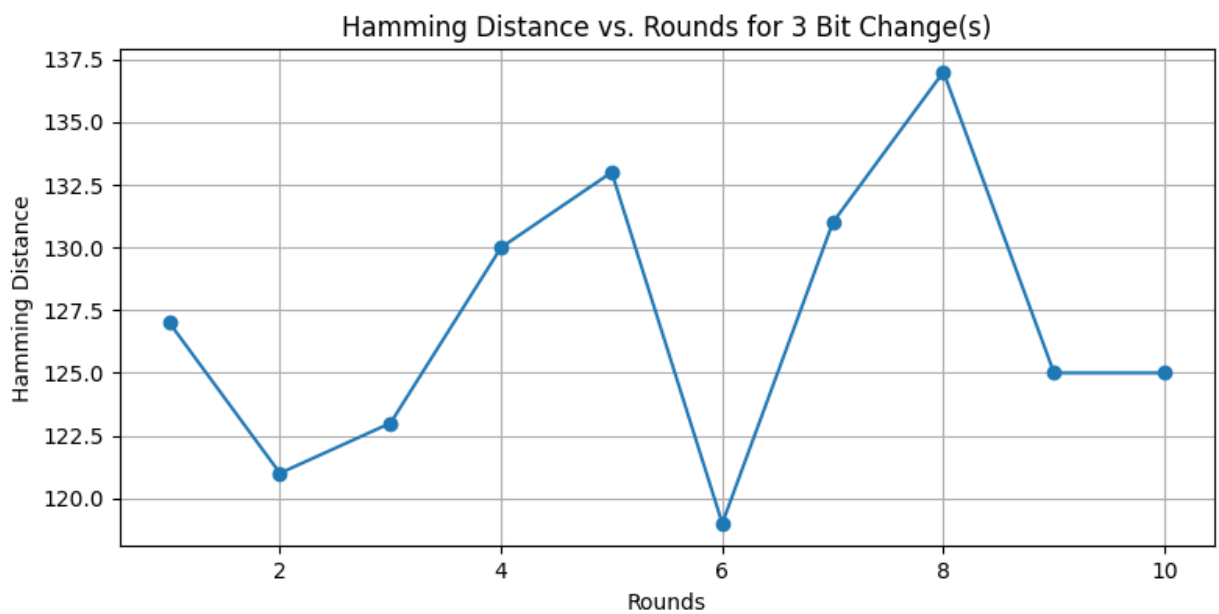
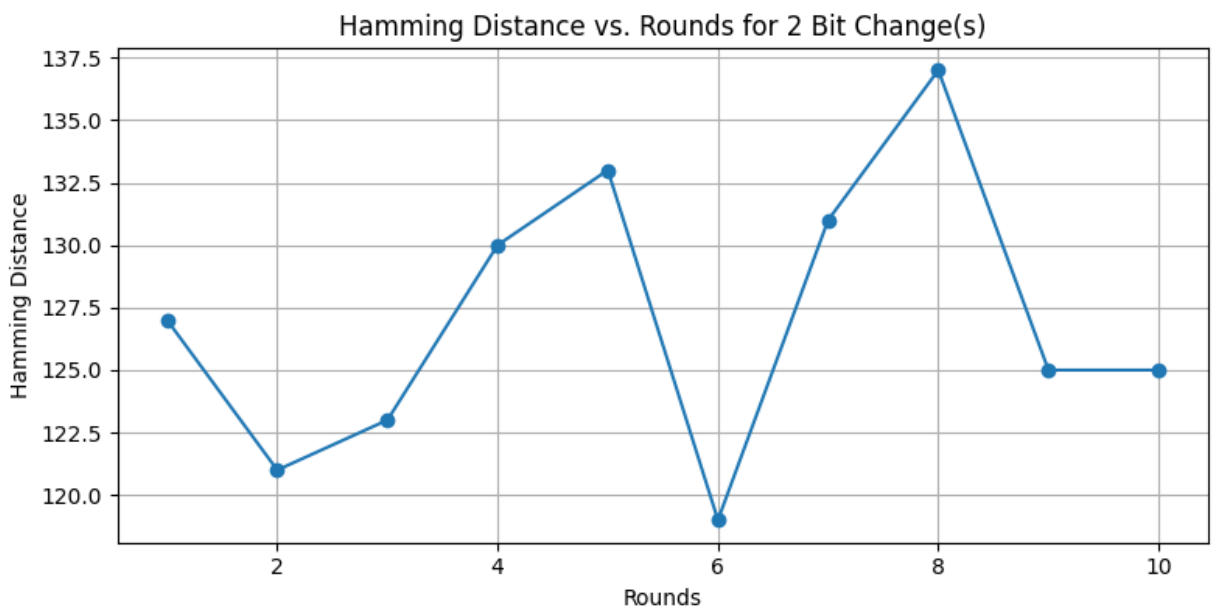
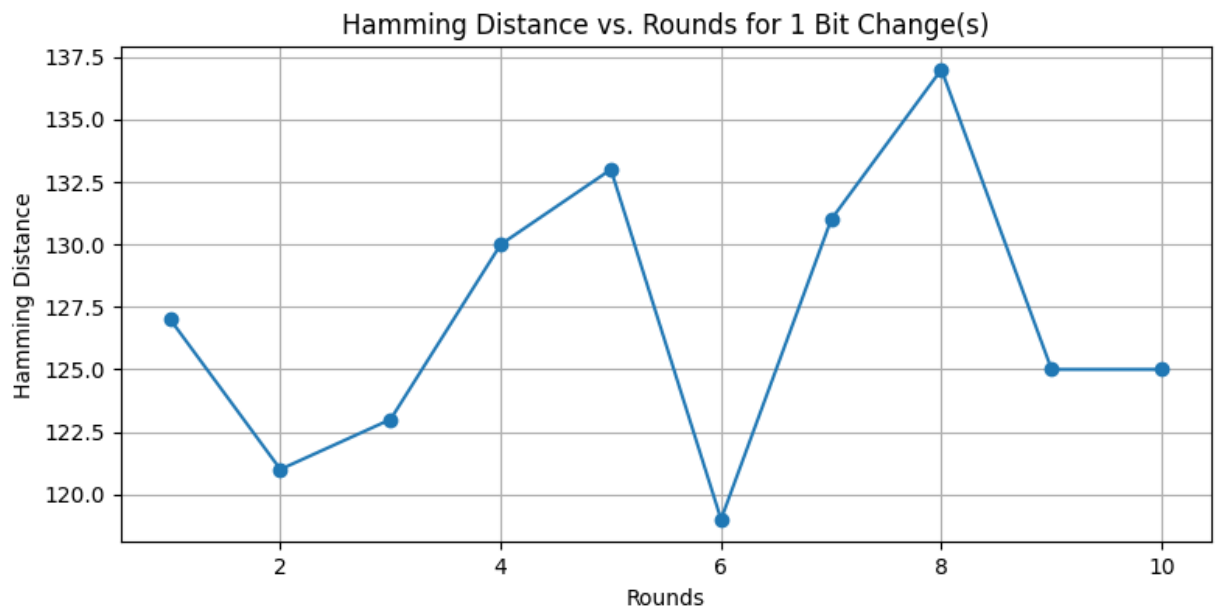
Входные данные:

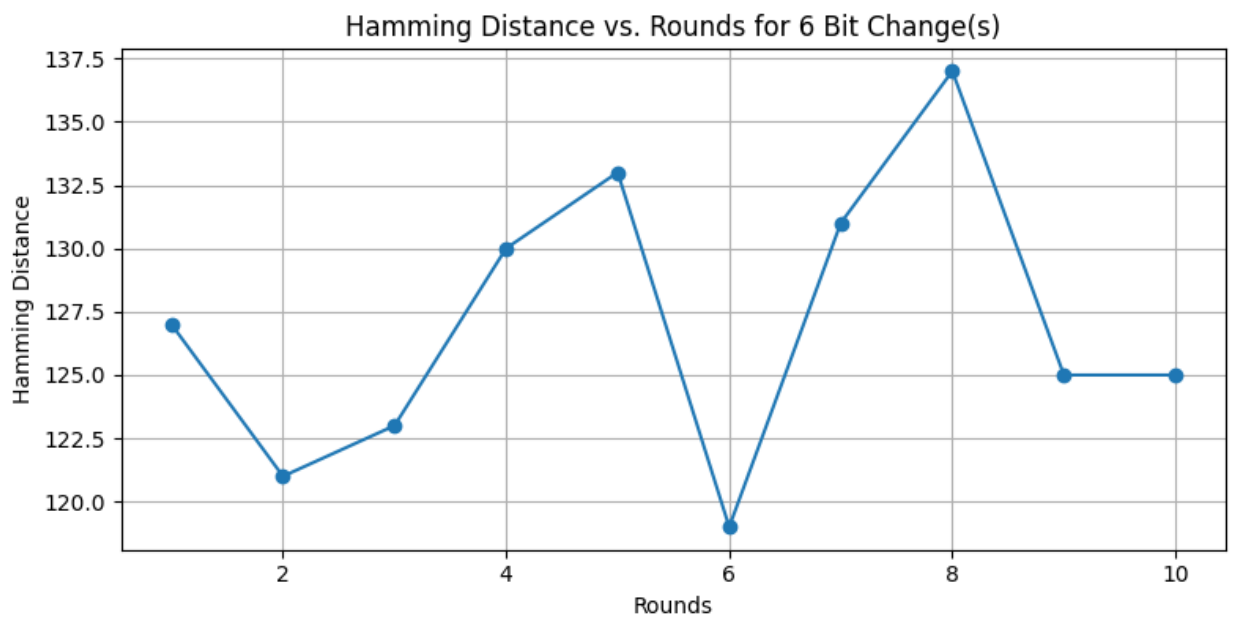
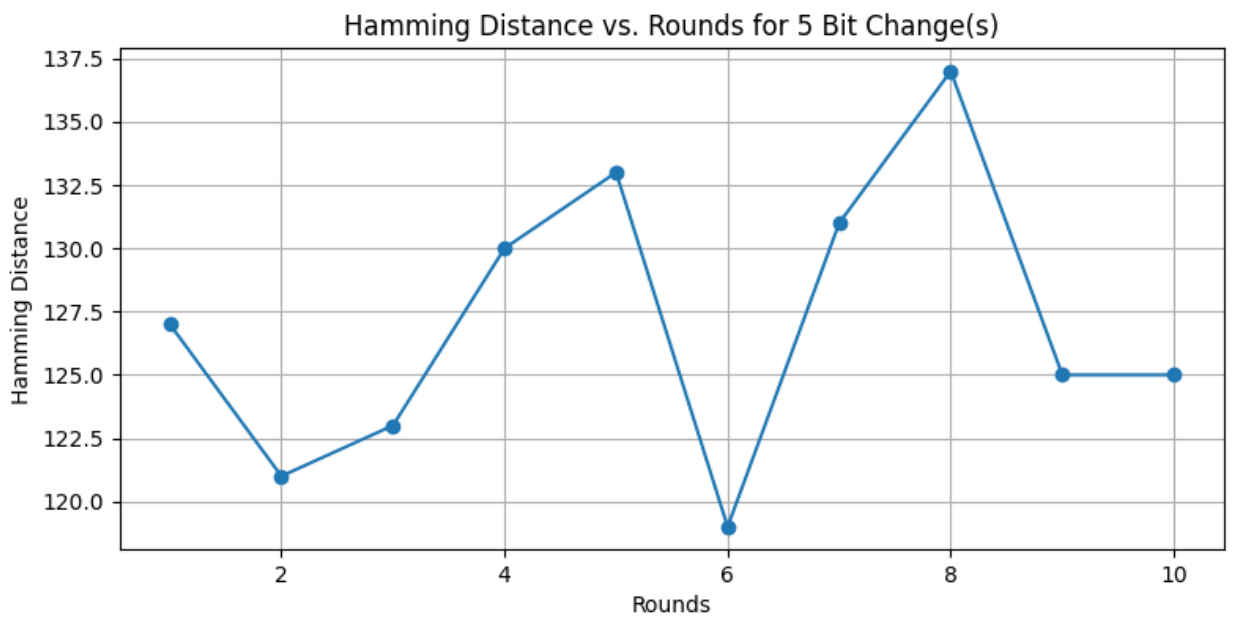
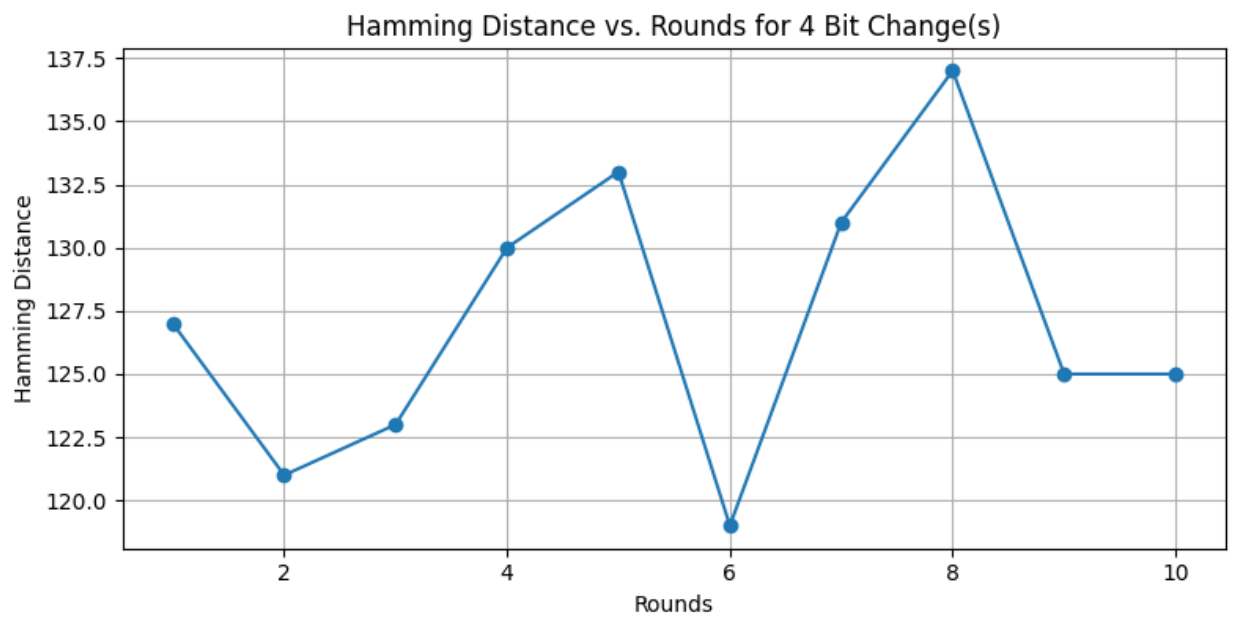
Максимальное количество раундов: 10

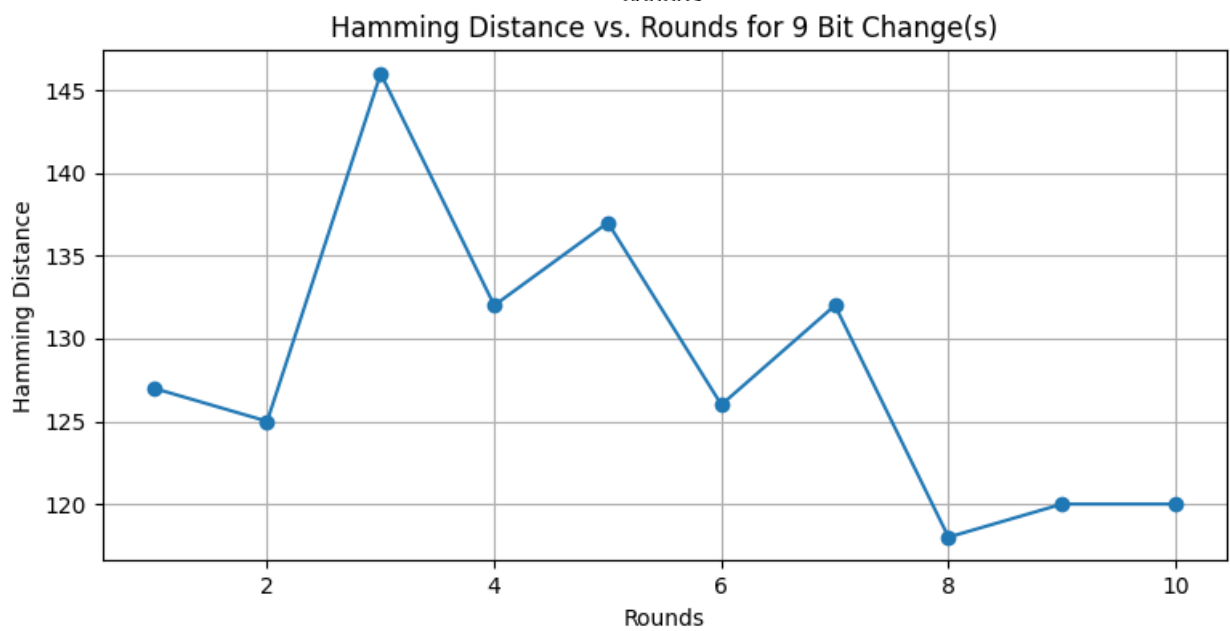
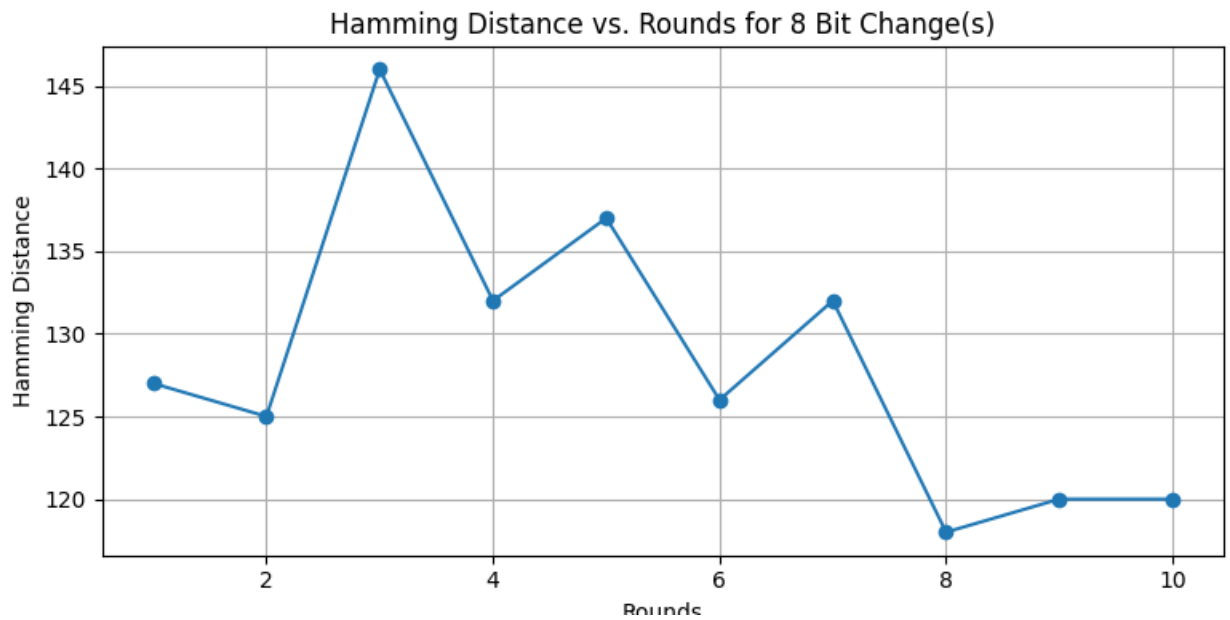
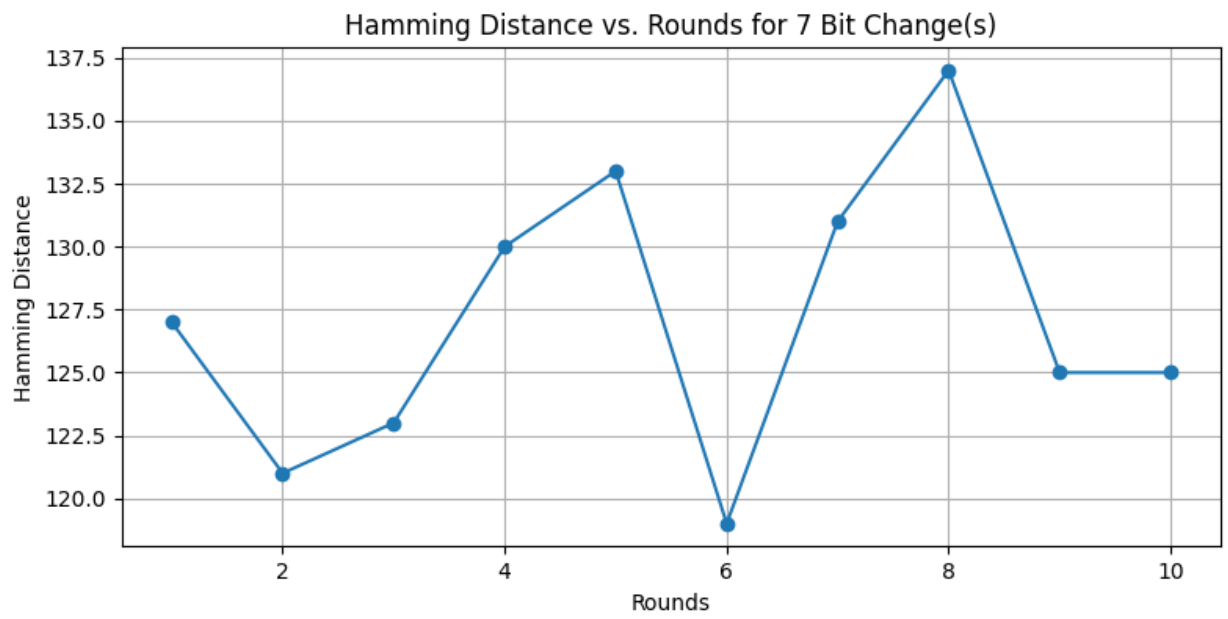
Максимальное количество битов которые необходимо менять: 10

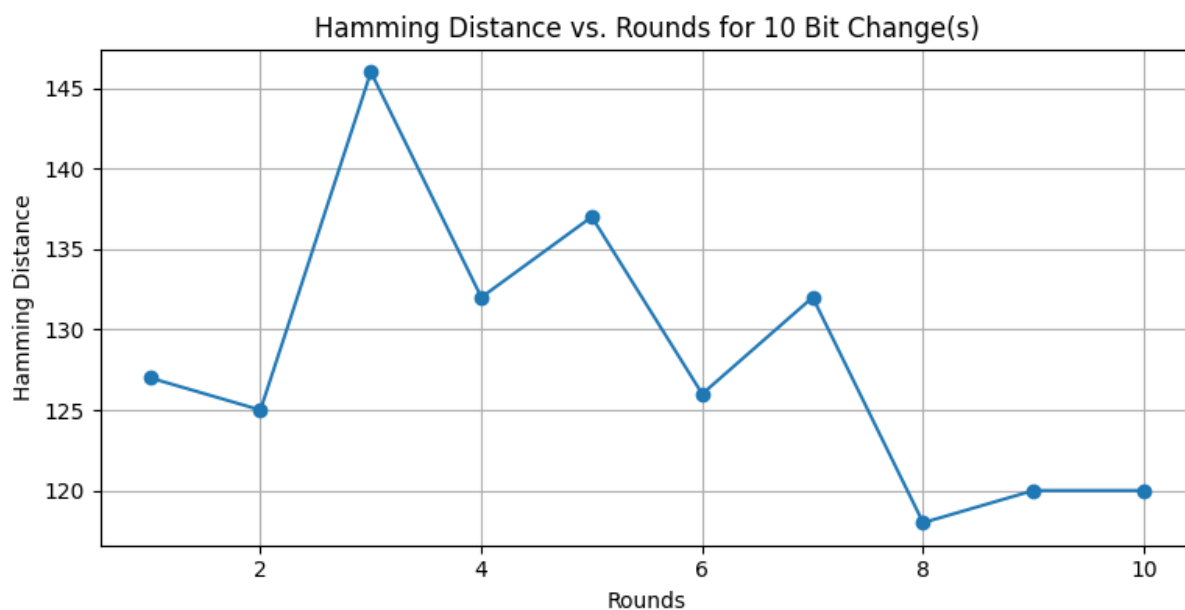
Текст для хэширования: Hello World!

Выходные данные – это множество троек: сколько раундов было применено, сколько битов изменено изначально, расстояние Хэмминга между хэшами оригинальной строки и модифицированной, поэтому лучше я представлю эту информацию графически для простоты понимания.









Итоги экспериментов демонстрируют, что даже незначительные изменения в одном бите входного текста могут привести к значительным изменениям в хэш-значениях (расстояние Хэмминга не опускается ниже 100). Это подтверждает высокую чувствительность алгоритма к входным данным. Однако, зависимость от количества изменённых битов не является значительной, как и зависимость от количества раундов, что указывает на эффективность алгоритма в защите от атак данного типа.

## **7 Выводы**

Исследование показало, что хэш-функция ЛН сохраняет свою стойкость к дифференциальному криптоанализу при различном количестве раундов, демонстрируя её высокую надёжность и возможность адаптации к разным настройкам раундов, заданным разработчиками.

## 8 Список источников

[https://www3.ntu.edu.sg/home/wuhj/research/jh/jh\\_sse2\\_opt64.h](https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_sse2_opt64.h)

<https://habr.com/ru/articles/215527/>

<https://ru.wikipedia.org/wiki/JH>

[https://www3.ntu.edu.sg/home/wuhj/research/jh/jh\\_round3.pdf](https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf)

<https://github.com/WhatTheMUCK/CryptoGraphy/tree/main/cp>