Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Криптография»

Студент: Д. А. Ваньков
Преподаватель: А. В. Борисов
Группа: М8О-307Б
Дата:
Оценка:
Подпись:

Москва, 2020

Вариант №5

Задача:

1. Строку в которой записано своё ФИО подать на вход в хеш-функцию ГОСТ Р 34.11-2012 (Стрибог). Младшие 4 бита выхода интерпретировать как число, которое в дальнейшем будет номером варианта. Процесс выбора варианта требуется отразить в отчёте.

2. Программно реализовать один из алгоритмов функции хеширования в соответствии с номером варианта. Алгоритм содержит в себе несколько раундов.

3. 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.

4. Применить подходы дифференциального криптоанализа к полученным алгоритмам с разным числом раундов.

5. Построить график зависимости количества раундов и возможности различения отдельных бит при количестве раундов 1,2,3,4,5,... .

6. Сделать выводы.

1 Описание

Вариант был выбран с помощью утилиты стрибог, взятой с открытого доступа с *Git Hub*.

```
(base) chappybunny@chappybunny:~/Crypto/Lab2/streebog$ ./gost3411-2012 -s

'Ваньков Денис Алексеевич'
GOST R 34.11-2012 ("Ваньков Денис Алексеевич") = 87d49005ed923aa7e22055
996ba4ab993251168528021dbdff6bacfdd8b32dd38c1628450a6a56275fe3a3e24a3b6d0
53dad37e3e83f448374a214c30d6d189f

>>>from pygost.gost34112012 import GOST34112012
>>>class GOST34112012256(GOST34112012):
...     def __init__(self,data=b''):
...         super(GOST34112012256,self).__init__(data,digest_size=32)
...
>>>
>>>def new(data=b''):
...     return GOST34112012256(data)
...
>>>

>>>GOST34112012256("Вньков Денис Алексеевич".encode('utf-8')).digest().
hex()[-1]
'5'
```

Алгоритм SHA-1.

Алгоритм работает следующим образом: Сначала происходит препроцессинг исходного сообщения так, чтобы его длина была кратна размеру блока в алгоритме (512). К сообщению добвляется 1 бит, затем последовательность нулей, чтобы длина последнего блока стала равной 448 бит. Затем в конце дописывается длина исходного сообщения до препроцессинга. Сообщение разбивается на блоки по 512 бит. Происходит нициализация 5-ти перменных:

$h0 = 0x67452301 = a$
$h1 = 0xEFCDAB89 = b$
$h2 = 0x98BADCFE = c$
$h3 = 0x10325476 = d$
$h4 = 0xC3D2E1F0 = e$

Определяются четыре функции, зависящие от числа раундов ($0 \leq i \leq 79$), и четыре константы:
$F_i(m, l, k) = (m \ \& \ l) \mid (\sim \ m \ \& \ k), k_i = 0x5A827999, 0 \leq i \leq 19$
$F_i(m, l, k) = m \oplus l \oplus k, k_i = 0x6ED9EBA1, 20 \leq i \leq 39$

$F_i(m,l,k) = (m \,\&\, l) \mid (m \,\&\, k) \mid (l \,\&\, k), k_i = 0x8F1BBCDC, 40 \leq i \leq 59$
$F_i(m,l,k) = m \oplus l \oplus k, k_i = 0xCA62C1D6, 60 \leq i \leq 79$

В главном цикле итеративно обрабатываются все 512-битные блоки. Согласно стандарту алгоритма, проводится 80 раундов. Блок сообщения преобразуется из 16 32-битовых слов $Message_i$ в 80 32-битовых слов W_i по следующему правилу:
$W_i = Message_i$, где $0 \leq i \leq 15$
$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \ll 1$, где $16 \leq i \leq 79$

В конце каждого раунда обновляются a, b, c, d, e:
$temp = (a \ll 5) + F_i(b,c,d) + e + k_i + W_i$
$e = d$
$d = c$
$c = b \ll 30$
$b = a$
$a = temp$

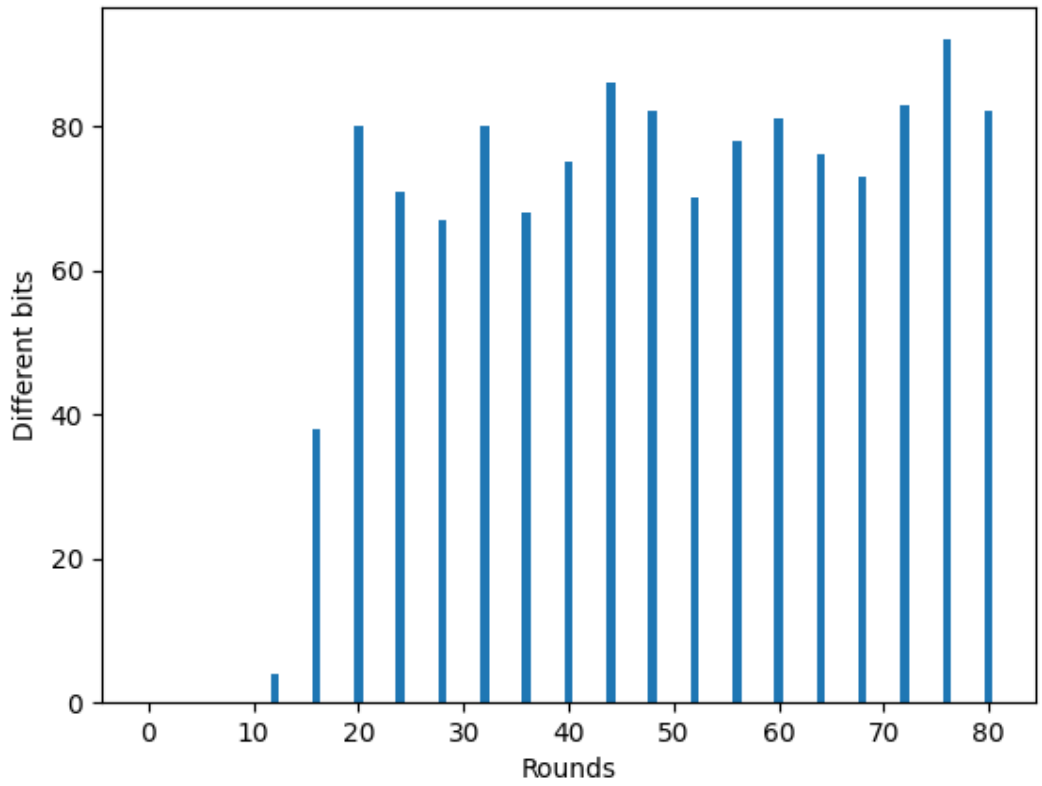
Затем к $h0, h1, h2, h3, h4$ прибавляются a, b, c, d, e соответственно и начинается следующая итерация. Итоговым значением будет объединение пяти 32-битовых слов в одно 160-битное хеш-значение.

Для выполнения задания номер 4 я генерирую рандомную строку из латинских букв разного регистра и цифр, создаю ее копию и у нее изменяю последний бит. Затем считаю хеш обеих строк для количества раундов в интервале от 0 до 81 с шагом 4 и подсчитываю количество различных бит в полученных хешах. В конце строю график зависимости количества раундов от количества различных битов. Процесс повторил два раза.

```
First string:  AvFL5cg3j7TLAc5WA769iPKXCq3HZz09Jr68WWu6
First string with changed last bit:  AvFL5cg3j7TLAc5WA769iPKXCq3HZz09Jr68WWu7
Current round: 0
First string with algo:  ce8a4602df9b57123175b9fc2064a8ec87a5c3e0
First changed string with algo:  ce8a4602df9b57123175b9fc2064a8ec87a5c3e0
Number of different bits: 0
Current round: 4
First string with algo:  40f3014bd71646c4c916e20d087d0c351da42ab0
First changed string with algo:  40f3014bd71646c4c916e20d087d0c351da42ab0
Number of different bits: 0
Current round: 8
First string with algo:  f086862d79ae367062bd79e029168e557a3e5982
First changed string with algo:  f086862d79ae367062bd79e029168e557a3e5982
Number of different bits: 0
Current round: 12
First string with algo:  b174739b5c03ce5e98854e549716a631e6233abb
First changed string with algo:  b174779b5c03ce7ed8854e559716a631e6233abb
Number of different bits: 4
Current round: 16
```

```
First string with algo:  a5d868244800c6f01d0291a5fc922a40565eb616
First changed string with algo:  75b90f15ca004c207d0690ad2c924a39565eb716
Number of different bits: 38
Current round: 20
First string with algo:  f7e627d129acc829d318bbca59bd98479377b338
First changed string with algo:  26a98413b6965d7e13b1b724e0c2d853c76fdcf5
Number of different bits: 80
Current round: 24
First string with algo:  2fd6ca6e2ef99f715657d1bd99d0b08ee7fb2324
First changed string with algo:  ce5fc7cde70e019cf44742ff6592f6a473abfa34
Number of different bits: 71
Current round: 28
First string with algo:  6ff3357ec23b932d31f4c3ae9d6af41b35f74bcb
First changed string with algo:  1d6aa1651253a60d1122edbfb03fe28fdd998b23
Number of different bits: 67
Current round: 32
First string with algo:  3367065b274b2a322db308ae6dc148c205fe668f
First changed string with algo:  29fb7771ce9bade4d2fc34d650822941f15c4189
Number of different bits: 80
Current round: 36
First string with algo:  92f0591ccb6ec593a08fb506dad7aef576db5ac6
First changed string with algo:  a8da263207eef5a8fcd65706f3ffe66cf480770c
Number of different bits: 68
Current round: 40
First string with algo:  07623a42e6c61a9e231937d86c15f4f88ebdaf76
First changed string with algo:  44740fa02f72de1a5bcd297e9d17670f143822bc
Number of different bits: 75
Current round: 44
First string with algo:  1b601387656639d18e83bda3f62bda722bda27c0
First changed string with algo:  6c5476e0188d4f9a07855c5eed43cc25bb1e9d17
Number of different bits: 86
Current round: 48
First string with algo:  90038c0bcc4b6758d4736feef341aa9a70d99e11
First changed string with algo:  f0e66a2ac35ad45a1d09501d7c66a0848516b6e7
Number of different bits: 82
Current round: 52
First string with algo:  ec2f11e4cca625c9c58c1bf3eaf6d8cb4e027c32
First changed string with algo:  3e6ef79e768bc769c55c27522950c587263b33ba
Number of different bits: 70
Current round: 56
First string with algo:  24f82e01946f4d5fdb0f53c1d717e42ba50d5da8
First changed string with algo:  81b87ebf3326a0ba2bf037432b20c60b399d5717
Number of different bits: 78
Current round: 60
First string with algo:  e243b18dff59319906dfecaf6824cf31f33fa4b0
First changed string with algo:  ffc0034d60565f90bfbd3accc6711a0b4a6fb8df
```

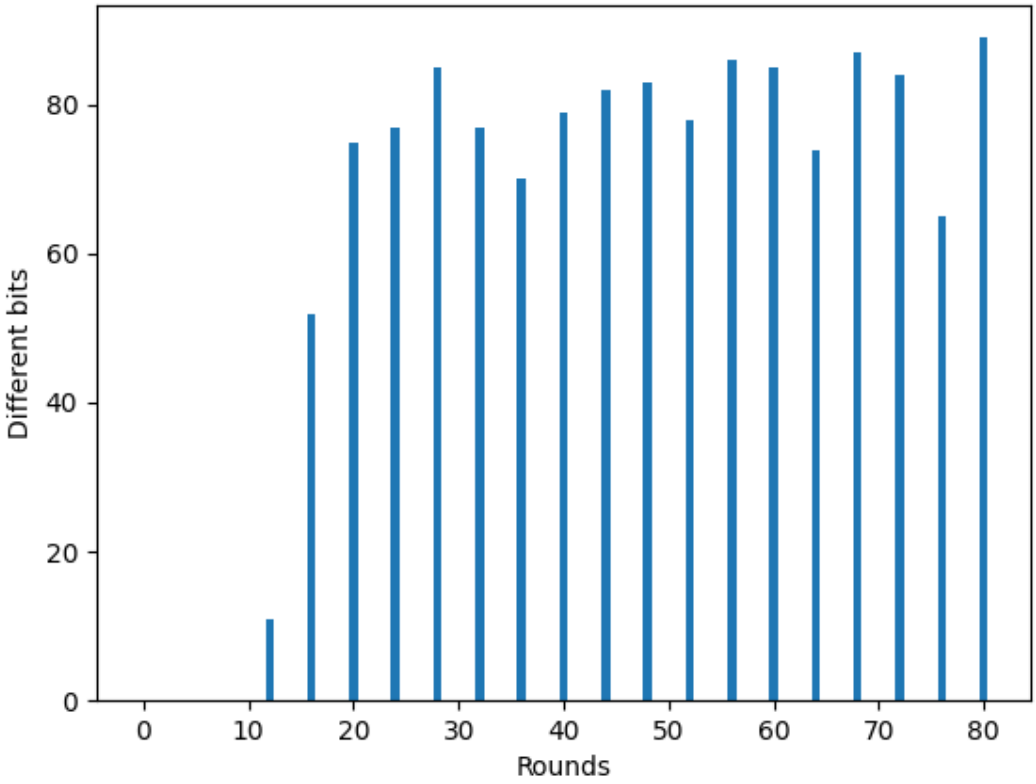
```
Number of different bits: 81
Current round: 64
First string with algo:  6c065e7173422721e1635497931b8c49e2928593
First changed string with algo:  e9ff5731f03546135c8a606dff9fe9fae9f19a03
Number of different bits: 76
Current round: 68
First string with algo:  170857c440e17e6555c51f0a94cf5e8ec50330cc
First changed string with algo:  7bc6758806d4014256897deb7f530dd2e4816efc
Number of different bits: 73
Current round: 72
First string with algo:  2c7b3739a19ad30ab55cabfda50f5cfdafc3af20
First changed string with algo:  13f07f8f717665f28e826da9ea3782f488f33691
Number of different bits: 83
Current round: 76
First string with algo:  611e9595df95d55b57e29aa269a79caef52066fe
First changed string with algo:  33454ee43081482eee71cd9cb8cb8e2a6efdb913
Number of different bits: 92
Current round: 80
First string with algo:  4802a49b6fe396d03c6e8b3bb769ec4d02493e95
First changed string with algo:  73d46594d9dc3d8998ee967a94165d69b6d2ece8
Number of different bits: 82
Round 0,different bits: 0
Round 4,different bits: 0
Round 8,different bits: 0
Round 12,different bits: 4
Round 16,different bits: 38
Round 20,different bits: 80
Round 24,different bits: 71
Round 28,different bits: 67
Round 32,different bits: 80
Round 36,different bits: 68
Round 40,different bits: 75
Round 44,different bits: 86
Round 48,different bits: 82
Round 52,different bits: 70
Round 56,different bits: 78
Round 60,different bits: 81
Round 64,different bits: 76
Round 68,different bits: 73
Round 72,different bits: 83
Round 76,different bits: 92
Round 80,different bits: 82
```




```
First string:  KUDhVypPiVY7bUvmy4T9RVQtii8WUiSbxz9o0jCd
First string with changed last bit:  KUDhVypPiVY7bUvmy4T9RVQtii8WUiSbxz9o0jCe
Current round: 0
First string with algo:  ce8a4602df9b57123175b9fc2064a8ec87a5c3e0
First changed string with algo:  ce8a4602df9b57123175b9fc2064a8ec87a5c3e0
Number of different bits: 0
Current round: 4
First string with algo:  6447a45b7310e17a205455340af4cbbc1da42ab0
First changed string with algo:  6447a45b7310e17a205455340af4cbbc1da42ab0
Number of different bits: 0
Current round: 8
First string with algo:  8a09aeea928bd813a615cd300288c17f83138246
First changed string with algo:  8a09aeea928bd813a615cd300288c17f83138246
Number of different bits: 0
Current round: 12
First string with algo:  e52a980b615d7bd85cbcf2755efca0d70c8404ea
First changed string with algo:  e52a9c24615d7bf89cbcf2765efca0d70c8404ea
Number of different bits: 11
Current round: 16
First string with algo:  0894f6173426abb42b49bde77bbb6b08634c3f32
First changed string with algo:  0a04c9af7632ab4c4b4dd6e6abbb8bd0a34c4038
```

```
Number of different bits: 52
Current round: 20
First string with algo:  50eebf98d52c8888d4c74230114e7aa86c26d6b5
First changed string with algo:  57c4f005595c595f97eecb4d88cd90716c82cb9b
Number of different bits: 75
Current round: 24
First string with algo:  03480b6e0d5d06a806918b31a5c74cbcbe3d4915
First changed string with algo:  236b08d13b3de861c64f767c2205c378fff2d531
Number of different bits: 77
Current round: 28
First string with algo:  bd078b65a7979a2b1f5ee5ec85cd11fb2ad39c0b
First changed string with algo:  23284e219feda7e54d9baa2bfbad0f69f2dc5b64
Number of different bits: 85
Current round: 32
First string with algo:  9aa231022452313c73864d52d6cea895d9437c09
First changed string with algo:  0a2266790a94db48af94813ca47a7a3ef2cbacb8
Number of different bits: 77
Current round: 36
First string with algo:  d43bcf9cdd73a9ce3ae7e803abfa695f10aa2570
First changed string with algo:  4c323cc2edc32eda8a61adbf4a3ee44fec8a32ce
Number of different bits: 70
Current round: 40
First string with algo:  0c14ed193b47d180f6e47dbae29b518a9f108d16
First changed string with algo:  0a8d9b9cacbd7e49f1fbf6b831ce7cba3d0e2860
Number of different bits: 79
Current round: 44
First string with algo:  454994ff760276eb5fe755ad88038811ed06d476
First changed string with algo:  421b8d19169cda146c91939edb217fceaca50016
Number of different bits: 82
Current round: 48
First string with algo:  7a316bbb174747bb36697f16e842a95e7b53fe6f
First changed string with algo:  89fa900114cade567c22d0ba3da0a30afa887c76
Number of different bits: 83
Current round: 52
First string with algo:  193b54e730dbdf886c62b97714f79b18488df41e
First changed string with algo:  67f89b6d13147147788b9b46551b941fcc803d30
Number of different bits: 78
Current round: 56
First string with algo:  e15acf4596c2470e312ae5212b17bff570506e69
First changed string with algo:  883513b976e0dc353d92aa3b838ce1a3c3ffc00b
Number of different bits: 86
Current round: 60
First string with algo:  e0b4a3f64524a8ff402b816d16143db0e2584d01
First changed string with algo:  f38c7950b0a394e59b26fc7364ea09f3cc0ede1e
Number of different bits: 85
Current round: 64
```

```
First string with algo:  207500e4f2be9d556e40678d66a5f5c6222ec22d
First changed string with algo:  d0c74824bf8746c3b669c0875f63ed7ea6e4b783
Number of different bits: 74
Current round: 68
First string with algo:  84a236b187324810edea79594d1c1a40b21ed968
First changed string with algo:  d81f0853b0dc18b85bf9853937e3c37d9e336b38
Number of different bits: 87
Current round: 72
First string with algo:  df9f59172776795a204643071fa26172cb2a26dc
First changed string with algo:  df016255a4eb8fefd4cb2f614045083860095b44
Number of different bits: 84
Current round: 76
First string with algo:  7bb3b9a4ba727c7c447102045405853361e96f75
First changed string with algo:  faaa820e2a13115c712b901ef2c0b025e1c1f1c5
Number of different bits: 65
Current round: 80
First string with algo:  3baddae7341a4e1822caae412019bcd888ee8798
First changed string with algo:  08c1911da377982ebdb439b02f439f1528ac39b3
Number of different bits: 89
Round 0,different bits: 0
Round 4,different bits: 0
Round 8,different bits: 0
Round 12,different bits: 11
Round 16,different bits: 52
Round 20,different bits: 75
Round 24,different bits: 77
Round 28,different bits: 85
Round 32,different bits: 77
Round 36,different bits: 70
Round 40,different bits: 79
Round 44,different bits: 82
Round 48,different bits: 83
Round 52,different bits: 78
Round 56,different bits: 86
Round 60,different bits: 85
Round 64,different bits: 74
Round 68,different bits: 87
Round 72,different bits: 84
Round 76,different bits: 65
Round 80,different bits: 89
```


Different bits
0
20
40
60
80
0
10
20
30
40
50
60
70
80
Rounds

2 Исходный код

Реализация алгоритма *SHA-1* на языке python.

```
1  BLOCK = 512
2  class SHA_1:
3      def __init__(self, rounds):
4          self.h = (0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0)
5          self.rounds = rounds
6
7      def hexDigets(self):
8          return '%08x%08x%08x%08x%08x' % self.h
9
10     staticmethoddef getChunk(text):return [text[i:i + BLOCK] for i in range(0,
           len(text), BLOCK)]staticmethod
11     def leftRotation(n, k):
12         return ((n << k) | (n >> (
13                 32 - k))) & 0xffffffff # << mean bitwise left shift on k , >> mean
                   bitwise right shift on (32 - k). | mean or
14
15     def process(self, chunk):
16         h0 = self.h[0]
17         h1 = self.h[1]
18         h2 = self.h[2]
19         h3 = self.h[3]
20         h4 = self.h[4]
21
22         W = []
23
24         for i in range(16):
25             W.append(int(chunk[i * 32: i * 32 + 32], 2))
26         for i in range(16, 80):
27             W.append(self.leftRotation(W[i - 3] ^ W[i - 8] ^ W[i - 14] ^ W[i -
                   16], 1)) # ^ mean XOR
28
29         a = h0
30         b = h1
31         c = h2
32         d = h3
33         e = h4
34
35         # Main loop
36         for i in range(self.rounds):
37             if (0 <= i <= 19):
38                 f = (b & c) | ((~b) & d)
39                 k = 0x5A827999
40             elif (20 <= i <= 39):
41                 f = b ^ c ^ d
42                 k = 0x6ED9EBA1
43             elif (40 <= i <= 59):
44                 f = (b & c) | (b & d) | (c & d)
45                 k = 0x8F1BBCDC
46             elif (60 <= i <= 79):
47                 f = b ^ c ^ d
48                 k = 0xCA62C1D6
49
```

```
            tmp = (self.leftRotation(a, 5) + f + e + k + W[i]) & 0xffffffff
            e = d
            d = c
            c = (self.leftRotation(b, 30))
            b = a
            a = tmp

        # Add this chunk's hash to result so far
        h0 = (h0 + a) & 0xffffffff
        h1 = (h1 + b) & 0xffffffff
        h2 = (h2 + c) & 0xffffffff
        h3 = (h3 + d) & 0xffffffff
        h4 = (h4 + e) & 0xffffffff

        self.h = (h0, h1, h2, h3, h4)

    def update(self, message):
        new_msg = ''

        for i in range(len(message)):
            new_msg += '{0:08b}'.format(ord(message[i]))

        len_msg = len(new_msg)
        new_msg += '1'
        while len(new_msg) % 512 != 448:
            new_msg += '0'
        new_msg += '{0:064b}'.format(len_msg)

        chunks = self.getChunk(new_msg)
        for each in chunks:
            self.process(each)

        return self


def algo(rounds, text):
    return SHA_1(rounds).update(text).hexDigets()


if __name__ == '__main__':
    print("Input rounds to start (<= 80): ", end='')
    rounds = int(input())
    print("Input text to start:")
    text = str(input())
    print("Algo: ", algo(rounds, text))
```

Анализ разности битов. Задание номер 4.

```
import matplotlib.pyplot as plt
import lab3 as sha1
import random
import string
import bitarray


def randomStringDigits(stringLength=40):
```

```
    lettersAndDigits = string.ascii_letters + string.digits
    return ''.join(random.choice(lettersAndDigits) for _ in range(stringLength))


def changeLastBit(data):
    arr = bitarray.bitarray()
    arr.frombytes(data.encode('ascii'))
    last_bit = arr[-1]
    if last_bit:
        last_new = bitarray.bitarray('0')
    else:
        last_new = bitarray.bitarray('1')
    ba = arr[:-1]
    ba += last_new
    return bitarray.bitarray(ba.tolist()).tobytes().decode('ascii')


def countBits(n):
    return bin(n).count('1')


if __name__ == '__main__':
    checker = []
    str1 = randomStringDigits()
    str2 = changeLastBit(str1)
    print('First string: ', str1)
    print('First string with changed last bit: ', str2)

    for i in range(0, 81, 4):
        print(f'Current round: {i}')
        str1_new = sha1.algo(i, str1)
        str2_new = sha1.algo(i, str2)
        print('First string with algo: ', str1_new)
        print('First changed string with algo: ', str2_new)
        bits = countBits(int(str1_new, 16) ^ int(str2_new, 16))
        print(f'Number of different bits: {bits}')
        checker.append(bits)

    rounds = [i for i in range(0, 81, 4)]
    for x, y in zip(rounds, checker):
        print(f'Round {x}, different bits: {y}')

    plt.bar(rounds, checker)
    plt.xlabel('Rounds')
    plt.ylabel('Different bits')
    plt.savefig('./fig2.png')
    plt.show()
```

3 Выводы

Такой анализ позволяет увидеть, насколько меняется хеш при минимальном изменении исходного сообщения. Учитывая, что итоговое значение имеет длину 160 бит, нетрудно заметить, что примерно с 24 раунда меняется около половины битов хеша. Однако, из-за малого количества тестов эти результаты не являются истинными.

До этого мне не приходилось сталкиваться с хеш-функциями. В этой лабораторной я познакомился как с алгоритмом или хеш функцией *SHA-1*, так и с криптографическими хеш-функциями в целом. Из-за недостатка знаний данная лабораторная работа и метод дифференциального криптоанализа показался мне сложен для понимания.