Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Криптография»

Студент: И. Т. Батяновский
Преподаватель: А. В. Борисов
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №3

Задача:

1. Строку в которой записано своё ФИО подать навход 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 алгоритм функции хеширования. 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 алгоритму с разным числом раундов.
5. Построить график зависимости количества раундов и возможности различения отдельных бит при различном количестве раундов.

Вариант 1: ГОСТ Р 34.11-2012 (Стриборг).

1 Расчёт варианта

Для расчета варианта я написал программу на языке *Python*, которая при помощи сторонней библиотеки **pygost** вычисляет хэш переданной ей строки (в моем случае «Батяновский Иван Тарасович»).

Код программы VarGen.py:

```
1 from pygost import gost34112012256
2
3 print(gost34112012256.new("Батяновский ИванТарасович".encode('utf-8')).digest().hex()
      [-1])
```

Результат вычисления:

C:\Anaconda\envs\Cryptography3\python.exe C:/Users/Ivan/PycharmProjects
/Cryptography3/VarGen.py
1

Process finished with exit code 0

Следовательно, мне следует реализовать алгоритм *ГОСТ Р 34.11-2012 (Стриборг)*. Стрибог — это семейство хеш-функций, включающее в себя всего две функции. Функцию с длиной выходного значения в 256 бит и функцию с длиной выходного значения в 512 бит. Они обе реализованы, но для «Дифференциального анализа» я буду использовать функцию с длиной выходного значения в 256 бит.

2 Описание алгоритма

Прежде чем приступать к описанию алгоритма, расскажем о преобразованиях, которые используются при вычислении хеш-функции.

1. **X - преобразование.** На вход функции **X** подаются две последовательности длиной 512 бит каждая, выходом функции является XOR этих последовательностей.
2. **S - преобразование.** Функция **S** является обычной функцией подстановки. Каждый байт из 512-битной входной последовательности заменяется соответствующим байтом из таблицы подстановок π.
   Таблица π является константой и может быть записана в виде массива.
3. **P - преобразование.** Функция перестановки. Для каждой пары байт из входной последовательности происходит замена одного байта другим.
   Таблица перестановок τ также является константой.
4. **L - преобразование.** Представляет собой умножение 64-битного входного вектора на бинарную матрицу **A** размерами 64x64.
   Представляет собой умножение 64-битного входного вектора на бинарную матрицу **A** размерами 64x64.

1 Функция сжатия

Опишем используемую в новом стандарте функцию сжатия g_n в виде алгоритма. Пусть **h, N** и **m** — 512-битные последовательности. Для вычисления функции **g(N, m, h)** необходимо проделать следующие шаги:

1. Вычислить значение K = h $\oplus$ N;
2. Присвоить значение K = S(K);
3. Присвоить значение K = P(K);
4. Присвоить значение K = L(K);
5. Вычислить t = E(K, m);
6. Присвоить значение t = h $\oplus$ t;
7. Вычислить значение G = t $\oplus$ m;
8. Вернуть G в качестве результата вычисления функции g(N, m, h).

Функция E(K, m) выполняет нижеприведенные действия:

1. Вычислить значение state = K $\oplus$ m;
2. Для i = 0 по 10 (количество раундов, которые мы можем изменять) выполнить:
   - Присвоить значение state = S(state);
   - Присвоить значение state = P(state);
   - Присвоить значение state = L(state);
   - Вычислить K = KeySchedule(K, i);
   - Присвоить значение state = state $\oplus$ K;
3. Вернуть state в качестве результата.

Функция KeySchedule(K, i), отвечает за формирование временного ключа K на каждом раунде функции E(K, m). Функция KeySchedule(K, i) производит следующие вычисления:

1. Присвоить значение K = K $\oplus$ C[i];
2. Присвоить значение K = S(K);
3. Присвоить значение K = P(K);
4. Присвоить значение K = L(K);
5. Вернуть K в качестве результата функции.

Здесь C — это набор 512-битных значений. Значение C является постоянным.

2 Вычисление хеш-функции

Теперь опишем процедуру формирования хеш-значения. Для любого входного сообщения M:

1. Присвоить начальные значения внутренних переменных:
   - Для хеш-функции с длиной выхода 512 бит: h=iv=$0x00^{64}$. Для хеш-функции с длиной выхода 256 бит: h=iv=$0x01^{64}$.
   - N = 0^{512}
   - $\Sigma = 0^{512}$
2. Проверить следующее условие: длина сообщения M < 512. Если условие выполняется перейти к пункту 3. В противном случае выполнить последовательность вычислений:
   - m — последние 512 бит сообщения M
   - h = g(N, m, h)
   - N = (N + 512) mod 2^{512}
   - Σ = (Σ + m) mod 2^{512}
   - Обрезать M, убрав последние 512 бит
   - Перейти к шагу 2
3. Произвести дополнение сообщения M до длины в 512 бит по следующему правилу:
   m = $0^{511-|M|}$ ||1||M, |M| - длина сообщения M в битах.
4. Вычислить h = g(N, m, h)
5. Вычислить N = (N + |M|) mod 2^{512}
6. Вычислить Σ = (Σ + m) mod 2^{512}
7. Вычислить h = g(0, h, N)
8. Вычислить h = g(0, h, Σ)
9. Для хеш-функции с длиной выхода в 512 бит возвращаем h в качестве результата. Для функции с длиной выхода 256 бит возвращаем MSB_{256}(h).

3 Реализация алгоритма

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
class GOST
    {
        // Matrix A for MixColumns (L) function
        private ulong[] A = {
        0x8e20faa72ba0b470, 0x47107ddd9b505a38, 0xad08b0e0c3282d1c, 0xd8045870ef14980e,
        0x6c022c38f90a4c07, 0x3601161cf205268d, 0x1b8e0b0e798c13c8, 0x83478b07b2468764,
        0xa011d380818e8f40, 0x5086e740ce47c920, 0x2843fd2067adea10, 0x14aff010bdd87508,
        0x0ad97808d06cb404, 0x05e23c0468365a02, 0x8c711e02341b2d01, 0x46b60f011a83988e,
        0x90dab52a387ae76f, 0x486dd4151c3dfdb9, 0x24b86a840e90f0d2, 0x125c354207487869,
        0x092e94218d243cba, 0x8a174a9ec8121e5d, 0x4585254f64090fa0, 0xaccc9ca9328a8950,
        0x9d4df05d5f661451, 0xc0a878a0a1330aa6, 0x60543c50de970553, 0x302a1e286fc58ca7,
        0x18150f14b9ec46dd, 0x0c84890ad27623e0, 0x0642ca05693b9f70, 0x0321658cba93c138,
        0x86275df09ce8aaa8, 0x439da0784e745554, 0xafc0503c273aa42a, 0xd960281e9d1d5215,
        0xe230140fc0802984, 0x71180a8960409a42, 0xb60c05ca30204d21, 0x5b068c651810a89e,
        0x456c34887a3805b9, 0xac361a443d1c8cd2, 0x561b0d22900e4669, 0x2b838811480723ba,
        0x9bcf4486248d9f5d, 0xc3e9224312c8c1a0, 0xeffa11af0964ee50, 0xf97d86d98a327728,
        0xe4fa2054a80b329c, 0x727d102a548b194e, 0x39b008152acb8227, 0x9258048415eb419d,
        0x492c024284fbaec0, 0xaa16012142f35760, 0x550b8e9e21f7a530, 0xa48b474f9ef5dc18,
        0x70a6a56e2440598e, 0x3853dc371220a247, 0x1ca76e95091051ad, 0x0edd37c48a08a6d8,
        0x07e095624504536c, 0x8d70c431ac02a736, 0xc83862965601dd1b, 0x641c314b2b8ee083
        };

        // Substitution for SubBytes function
        private byte[] Sbox={
        0xFC, 0xEE, 0xDD, 0x11, 0xCF, 0x6E, 0x31, 0x16, 0xFB, 0xC4, 0xFA, 0xDA, 0x23, 0
            xC5, 0x04, 0x4D,
        0xE9, 0x77, 0xF0, 0xDB, 0x93, 0x2E, 0x99, 0xBA, 0x17, 0x36, 0xF1, 0xBB, 0x14, 0
            xCD, 0x5F, 0xC1,
        0xF9, 0x18, 0x65, 0x5A, 0xE2, 0x5C, 0xEF, 0x21, 0x81, 0x1C, 0x3C, 0x42, 0x8B, 0
            x01, 0x8E, 0x4F,
        0x05, 0x84, 0x02, 0xAE, 0xE3, 0x6A, 0x8F, 0xA0, 0x06, 0x0B, 0xED, 0x98, 0x7F, 0
            xD4, 0xD3, 0x1F,
        0xEB, 0x34, 0x2C, 0x51, 0xEA, 0xC8, 0x48, 0xAB, 0xF2, 0x2A, 0x68, 0xA2, 0xFD, 0
            x3A, 0xCE, 0xCC,
        0xB5, 0x70, 0x0E, 0x56, 0x08, 0x0C, 0x76, 0x12, 0xBF, 0x72, 0x13, 0x47, 0x9C, 0
            xB7, 0x5D, 0x87,
        0x15, 0xA1, 0x96, 0x29, 0x10, 0x7B, 0x9A, 0xC7, 0xF3, 0x91, 0x78, 0x6F, 0x9D, 0
            x9E, 0xB2, 0xB1,
        0x32, 0x75, 0x19, 0x3D, 0xFF, 0x35, 0x8A, 0x7E, 0x6D, 0x54, 0xC6, 0x80, 0xC3, 0
            xBD, 0x0D, 0x57,
        0xDF, 0xF5, 0x24, 0xA9, 0x3E, 0xA8, 0x43, 0xC9, 0xD7, 0x79, 0xD6, 0xF6, 0x7C, 0
            x22, 0xB9, 0x03,
        0xE0, 0x0F, 0xEC, 0xDE, 0x7A, 0x94, 0xB0, 0xBC, 0xDC, 0xE8, 0x28, 0x50, 0x4E, 0
```

```
            x33, 0x0A, 0x4A,
39          0xA7, 0x97, 0x60, 0x73, 0x1E, 0x00, 0x62, 0x44, 0x1A, 0xB8, 0x38, 0x82, 0x64, 0
                x9F, 0x26, 0x41,
40          0xAD, 0x45, 0x46, 0x92, 0x27, 0x5E, 0x55, 0x2F, 0x8C, 0xA3, 0xA5, 0x7D, 0x69, 0
                xD5, 0x95, 0x3B,
41          0x07, 0x58, 0xB3, 0x40, 0x86, 0xAC, 0x1D, 0xF7, 0x30, 0x37, 0x6B, 0xE4, 0x88, 0
                xD9, 0xE7, 0x89,
42          0xE1, 0x1B, 0x83, 0x49, 0x4C, 0x3F, 0xF8, 0xFE, 0x8D, 0x53, 0xAA, 0x90, 0xCA, 0
                xD8, 0x85, 0x61,
43          0x20, 0x71, 0x67, 0xA4, 0x2D, 0x2B, 0x09, 0x5B, 0xCB, 0x9B, 0x25, 0xD0, 0xBE, 0
                xE5, 0x6C, 0x52,
44          0x59, 0xA6, 0x74, 0xD2, 0xE6, 0xF4, 0xB4, 0xC0, 0xD1, 0x66, 0xAF, 0xC2, 0x39, 0
                x4B, 0x63, 0xB6
45          };
46
47          // Substitution for Transposition (P) function
48          private byte[] Tau={
49          0, 8, 16, 24, 32, 40, 48, 56,
50          1, 9, 17, 25, 33, 41, 49, 57,
51          2, 10, 18, 26, 34, 42, 50, 58,
52          3, 11, 19, 27, 35, 43, 51, 59,
53          4, 12, 20, 28, 36, 44, 52, 60,
54          5, 13, 21, 29, 37, 45, 53, 61,
55          6, 14, 22, 30, 38, 46, 54, 62,
56          7, 15, 23, 31, 39, 47, 55, 63
57          };
58
59          // Constant values for KeySchedule function
60          private byte[][] C = {
61          new byte[64]{
62          0xb1,0x08,0x5b,0xda,0x1e,0xca,0xda,0xe9,0xeb,0xcb,0x2f,0x81,0xc0,0x65,0x7c,0x1f
                ,
63          0x2f,0x6a,0x76,0x43,0x2e,0x45,0xd0,0x16,0x71,0x4e,0xb8,0x8d,0x75,0x85,0xc4,0xfc
                ,
64          0x4b,0x7c,0xe0,0x91,0x92,0x67,0x69,0x01,0xa2,0x42,0x2a,0x08,0xa4,0x60,0xd3,0x15
                ,
65          0x05,0x76,0x74,0x36,0xcc,0x74,0x4d,0x23,0xdd,0x80,0x65,0x59,0xf2,0xa6,0x45,0x07
66          },
67          new byte[64]{
68          0x6f,0xa3,0xb5,0x8a,0xa9,0x9d,0x2f,0x1a,0x4f,0xe3,0x9d,0x46,0x0f,0x70,0xb5,0xd7
                ,
69          0xf3,0xfe,0xea,0x72,0x0a,0x23,0x2b,0x98,0x61,0xd5,0x5e,0x0f,0x16,0xb5,0x01,0x31
                ,
70          0x9a,0xb5,0x17,0x6b,0x12,0xd6,0x99,0x58,0x5c,0xb5,0x61,0xc2,0xdb,0x0a,0xa7,0xca
                ,
71          0x55,0xdd,0xa2,0x1b,0xd7,0xcb,0xcd,0x56,0xe6,0x79,0x04,0x70,0x21,0xb1,0x9b,0xb7
72          },
73          new byte[64]{
74          0xf5,0x74,0xdc,0xac,0x2b,0xce,0x2f,0xc7,0x0a,0x39,0xfc,0x28,0x6a,0x3d,0x84,0x35
```

```
        ,
75      0x06,0xf1,0x5e,0x5f,0x52,0x9c,0x1f,0x8b,0xf2,0xea,0x75,0x14,0xb1,0x29,0x7b,0x7b
        ,
76      0xd3,0xe2,0x0f,0xe4,0x90,0x35,0x9e,0xb1,0xc1,0xc9,0x3a,0x37,0x60,0x62,0xdb,0x09
        ,
77      0xc2,0xb6,0xf4,0x43,0x86,0x7a,0xdb,0x31,0x99,0x1e,0x96,0xf5,0x0a,0xba,0x0a,0xb2
78      },
79      new byte[64]{
80      0xef,0x1f,0xdf,0xb3,0xe8,0x15,0x66,0xd2,0xf9,0x48,0xe1,0xa0,0x5d,0x71,0xe4,0xdd
        ,
81      0x48,0x8e,0x85,0x7e,0x33,0x5c,0x3c,0x7d,0x9d,0x72,0x1c,0xad,0x68,0x5e,0x35,0x3f
        ,
82      0xa9,0xd7,0x2c,0x82,0xed,0x03,0xd6,0x75,0xd8,0xb7,0x13,0x33,0x93,0x52,0x03,0xbe
        ,
83      0x34,0x53,0xea,0xa1,0x93,0xe8,0x37,0xf1,0x22,0x0c,0xbe,0xbc,0x84,0xe3,0xd1,0x2e
84      },
85      new byte[64]{
86      0x4b,0xea,0x6b,0xac,0xad,0x47,0x47,0x99,0x9a,0x3f,0x41,0x0c,0x6c,0xa9,0x23,0x63
        ,
87      0x7f,0x15,0x1c,0x1f,0x16,0x86,0x10,0x4a,0x35,0x9e,0x35,0xd7,0x80,0x0f,0xff,0xbd
        ,
88      0xbf,0xcd,0x17,0x47,0x25,0x3a,0xf5,0xa3,0xdf,0xff,0x00,0xb7,0x23,0x27,0x1a,0x16
        ,
89      0x7a,0x56,0xa2,0x7e,0xa9,0xea,0x63,0xf5,0x60,0x17,0x58,0xfd,0x7c,0x6c,0xfe,0x57
90      },
91      new byte[64]{
92      0xae,0x4f,0xae,0xae,0x1d,0x3a,0xd3,0xd9,0x6f,0xa4,0xc3,0x3b,0x7a,0x30,0x39,0xc0
        ,
93      0x2d,0x66,0xc4,0xf9,0x51,0x42,0xa4,0x6c,0x18,0x7f,0x9a,0xb4,0x9a,0xf0,0x8e,0xc6
        ,
94      0xcf,0xfa,0xa6,0xb7,0x1c,0x9a,0xb7,0xb4,0x0a,0xf2,0x1f,0x66,0xc2,0xbe,0xc6,0xb6
        ,
95      0xbf,0x71,0xc5,0x72,0x36,0x90,0x4f,0x35,0xfa,0x68,0x40,0x7a,0x46,0x64,0x7d,0x6e
96      },
97      new byte[64]{
98      0xf4,0xc7,0x0e,0x16,0xee,0xaa,0xc5,0xec,0x51,0xac,0x86,0xfe,0xbf,0x24,0x09,0x54
        ,
99      0x39,0x9e,0xc6,0xc7,0xe6,0xbf,0x87,0xc9,0xd3,0x47,0x3e,0x33,0x19,0x7a,0x93,0xc9
        ,
100     0x09,0x92,0xab,0xc5,0x2d,0x82,0x2c,0x37,0x06,0x47,0x69,0x83,0x28,0x4a,0x05,0x04
        ,
101     0x35,0x17,0x45,0x4c,0xa2,0x3c,0x4a,0xf3,0x88,0x86,0x56,0x4d,0x3a,0x14,0xd4,0x93
102     },
103     new byte[64]{
104     0x9b,0x1f,0x5b,0x42,0x4d,0x93,0xc9,0xa7,0x03,0xe7,0xaa,0x02,0x0c,0x6e,0x41,0x41
        ,
105     0x4e,0xb7,0xf8,0x71,0x9c,0x36,0xde,0x1e,0x89,0xb4,0x44,0x3b,0x4d,0xdb,0xc4,0x9a
        ,
106     0xf4,0x89,0x2b,0xcb,0x92,0x9b,0x06,0x90,0x69,0xd1,0x8d,0x2b,0xd1,0xa5,0xc4,0x2f
```

```
        ,
0x36,0xac,0xc2,0x35,0x59,0x51,0xa8,0xd9,0xa4,0x7f,0x0d,0xd4,0xbf,0x02,0xe7,0x1e
},
new byte[64]{
0x37,0x8f,0x5a,0x54,0x16,0x31,0x22,0x9b,0x94,0x4c,0x9a,0xd8,0xec,0x16,0x5f,0xde
    ,
0x3a,0x7d,0x3a,0x1b,0x25,0x89,0x42,0x24,0x3c,0xd9,0x55,0xb7,0xe0,0x0d,0x09,0x84
    ,
0x80,0x0a,0x44,0x0b,0xdb,0xb2,0xce,0xb1,0x7b,0x2b,0x8a,0x9a,0xa6,0x07,0x9c,0x54
    ,
0x0e,0x38,0xdc,0x92,0xcb,0x1f,0x2a,0x60,0x72,0x61,0x44,0x51,0x83,0x23,0x5a,0xdb
},
new byte[64]{
0xab,0xbe,0xde,0xa6,0x80,0x05,0x6f,0x52,0x38,0x2a,0xe5,0x48,0xb2,0xe4,0xf3,0xf3
    ,
0x89,0x41,0xe7,0x1c,0xff,0x8a,0x78,0xdb,0x1f,0xff,0xe1,0x8a,0x1b,0x33,0x61,0x03
    ,
0x9f,0xe7,0x67,0x02,0xaf,0x69,0x33,0x4b,0x7a,0x1e,0x6c,0x30,0x3b,0x76,0x52,0xf4
    ,
0x36,0x98,0xfa,0xd1,0x15,0x3b,0xb6,0xc3,0x74,0xb4,0xc7,0xfb,0x98,0x45,0x9c,0xed
},
new byte[64]{
0x7b,0xcd,0x9e,0xd0,0xef,0xc8,0x89,0xfb,0x30,0x02,0xc6,0xcd,0x63,0x5a,0xfe,0x94
    ,
0xd8,0xfa,0x6b,0xbb,0xeb,0xab,0x07,0x61,0x20,0x01,0x80,0x21,0x14,0x84,0x66,0x79
    ,
0x8a,0x1d,0x71,0xef,0xea,0x48,0xb9,0xca,0xef,0xba,0xcd,0x1d,0x7d,0x47,0x6e,0x98
    ,
0xde,0xa2,0x59,0x4a,0xc0,0x6f,0xd8,0x5d,0x6b,0xca,0xa4,0xcd,0x81,0xf3,0x2d,0x1b
},
new byte[64]{
0x37,0x8e,0xe7,0x67,0xf1,0x16,0x31,0xba,0xd2,0x13,0x80,0xb0,0x04,0x49,0xb1,0x7a
    ,
0xcd,0xa4,0x3c,0x32,0xbc,0xdf,0x1d,0x77,0xf8,0x20,0x12,0xd4,0x30,0x21,0x9f,0x9b
    ,
0x5d,0x80,0xef,0x9d,0x18,0x91,0xcc,0x86,0xe7,0x1d,0xa4,0xaa,0x88,0xe1,0x28,0x52
    ,
0xfa,0xf4,0x17,0xd5,0xd9,0xb2,0x1b,0x99,0x48,0xbc,0x92,0x4a,0xf1,0x1b,0xd7,0x20
}
};

private byte[] iv =new byte[64];

private byte[] N =new byte[64];

private byte[] Sigma = new byte[64];

public int outLen = 0;

```

```
143         public GOST(int outputLenght)
144         {
145             if (outputLenght == 512)
146             {
147                 for (int i = 0; i < 64; i++)
148                 {
149                     N[i] = 0x00;
150                     Sigma[i] = 0x00;
151                     iv[i] = 0x00;
152                 }
153                 outLen = 512;
154             }
155             else if (outputLenght == 256)
156             {
157                 for (int i = 0; i < 64; i++)
158                 {
159                     N[i] = 0x00;
160                     Sigma[i] = 0x00;
161                     iv[i] = 0x01;
162                 }
163                 outLen = 256;
164             }
165         }
166 
167         private byte[] AddModulo512(byte[] a, byte[] b)
168         {
169             byte[] temp = new byte[64];
170             int i = 0, t = 0;
171             byte[] tempA = new byte[64];
172             byte[] tempB = new byte[64];
173             Array.Copy(a, 0, tempA, 64 - a.Length, a.Length);
174             Array.Copy(b, 0, tempB, 64 - b.Length, b.Length);
175             for (i = 63; i >= 0; i--)
176             {
177                 t = tempA[i] + tempB[i] + (t >> 8);
178                 temp[i] = (byte)(t & 0xFF);
179             }
180             return temp;
181         }
182 
183         private byte[] AddXor512(byte[] a, byte[] b)
184         {
185             byte[] c = new byte[64];
186             for (int i = 0; i < 64; i++)
187                 c[i] = (byte)(a[i] ^ b[i]);
188             return c;
189         }
190 
191         private byte[] S(byte[] state)
```

```
192         {
193             byte[] result = new byte[64];
194             for (int i = 0; i < 64; i++)
195                 result[i] = Sbox[state[i]];
196             return result;
197         }
198 
199         private byte[] P(byte[] state)
200         {
201             byte[] result = new byte[64];
202             for (int i = 0; i < 64; i++)
203             {
204                 result[i] = state[Tau[i]];
205             }
206             return result;
207         }
208 
209         private byte[] L(byte[] state)
210         {
211             byte[] result = new byte[64];
212             for (int i = 0; i < 8; i++)
213             {
214                 ulong t = 0;
215                 byte[] tempArray = new byte[8];
216                 Array.Copy(state, i * 8, tempArray, 0, 8);
217                 tempArray = tempArray.Reverse().ToArray();
218                 BitArray tempBits1 = new BitArray(tempArray);
219                 bool[] tempBits=new bool[64];
220                 tempBits1.CopyTo(tempBits, 0);
221                 tempBits=tempBits.Reverse().ToArray();
222                 for (int j = 0; j < 64; j++)
223                 {
224                     if (tempBits[j] != false)
225                         t = t ^ A[j];
226                 }
227                 byte[] ResPart = BitConverter.GetBytes(t).Reverse().ToArray();
228                 Array.Copy(ResPart, 0, result, i * 8, 8);
229             }
230             return result;
231         }
232 
233         private byte[] KeySchedule(byte[] K, int i)
234         {
235             K=AddXor512(K, C[i]);
236             K = S(K);
237             K = P(K);
238             K = L(K);
239             return K;
240         }
```

```
241
242         private byte[] E(byte[] K, byte[] m)
243         {
244             byte[] state = AddXor512(K, m);
245             for (int i = 0; i < 4; i++)
246             {
247                 state=S(state);
248                 state = P(state);
249                 state = L(state);
250                 K=KeySchedule(K, i);
251                 state = AddXor512(state, K);
252             }
253             return state;
254         }
255
256         private byte[] G_n(byte[] N, byte[] h, byte[] m)
257         {
258             byte[] K = AddXor512(h, N);
259             K=S(K);
260             K=P(K);
261             K=L(K);
262             byte[] t= E(K, m);
263             t=AddXor512(t, h);
264             byte[] newh = AddXor512(t, m);
265             return newh;
266         }
267
268         public byte[] GetHash(byte[] message)
269         {
270             byte[] paddedMes=new byte[64];
271             int len = message.Length * 8;
272             byte[] h = new byte[64];
273             Array.Copy(iv, h, 64);
274             byte[] N_0 ={
275             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
                 ,0x00,
276             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
                 ,0x00,
277             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
                 ,0x00,
278             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
                 ,0x00
279             };
280             if (outLen == 512)
281             {
282                 for (int i = 0; i < 64; i++)
283                 {
284                     N[i] = 0x00;
285                     Sigma[i] = 0x00;
```

```
                iv[i] = 0x00;
            }
        }
        else if (outLen == 256)
        {
            for (int i = 0; i < 64; i++)
            {
                N[i] = 0x00;
                Sigma[i] = 0x00;
                iv[i] = 0x01;
            }
        }
        byte[] N_512 = BitConverter.GetBytes(512);
        int inc = 0;
        while (len >= 512)
        {
            inc++;
            byte[] tempMes = new byte[64];
            Array.Copy(message, message.Length - inc*64, tempMes, 0, 64);
            h=G_n(N, h, tempMes);
            N = AddModulo512(N, N_512.Reverse().ToArray());
            Sigma=AddModulo512(Sigma, tempMes);
            len -= 512;
        }
        byte[] message1 = new byte[message.Length - inc * 64];
        Array.Copy(message, 0, message1, 0, message.Length - inc * 64);
        if (message1.Length < 64)
        {
            for (int i = 0; i < (64 - message1.Length - 1); i++)
            {
                paddedMes[i] = 0;
            }
            paddedMes[64 - message1.Length - 1] = 0x01;
            Array.Copy(message1, 0, paddedMes, 64 - message1.Length, message1.Length
                );
        }
        h=G_n(N, h, paddedMes);
        byte[] MesLen = BitConverter.GetBytes(message1.Length * 8);
        N = AddModulo512(N, MesLen.Reverse().ToArray());
        Sigma = AddModulo512(Sigma, paddedMes);
        h = G_n(N_0, h, N);
        h = G_n(N_0, h, Sigma);
        if (outLen == 512)
            return h;
        else
        {
            byte[] h256 = new byte[32];
            Array.Copy(h, 0, h256, 0, 32);
            return h256;
```

```
334             }
335         }
336     }
337 
338 namespace Compile
339 {
340     class Program
341     {
342         static void Main(string[] args)
343         {
344         GOST G = new GOST(256);
345         GOST G512 = new GOST(512);
346         byte[] message={
347                             0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0
                                x30,0x39,0x38,0x37,
348                             0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0
                                x34,0x33,0x32,0x31,
349                             0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0
                                x38,0x37,0x36,0x35,
350                             0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0
                                x32,0x31,0x30};
351         byte[] res = G.GetHash(message);
352         byte[] res2 = G512.GetHash(message);
353         string h256 = BitConverter.ToString(res);
354         string h512 = BitConverter.ToString(res2);
355         Console.WriteLine(h256);
356         Console.WriteLine();
357         Console.WriteLine(h512);
358         Console.ReadLine();
359         }
360     }
361 }
```

Вычислим хеш-функцию для массива данных(message).

Для этого используем:

```
1  private void ComputeHash()
2          {
3              GOST G = new GOST(256);
4              GOST G512 = new GOST(512);
5              byte[] message={
6                                  0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0
                                       x31,0x30,0x39,0x38,0x37,
7                                  0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0
                                       x35,0x34,0x33,0x32,0x31,
8                                  0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0
                                       x39,0x38,0x37,0x36,0x35,
9                                  0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0
                                       x33,0x32,0x31,0x30};
10             byte[] res = G.GetHash(message);
11             byte[] res2 = G512.GetHash(message);
12             string h256 = BitConverter.ToString(res);
13             string h512 = BitConverter.ToString(res2);
14         }
```

В результате получим значения:

h256 = 00-55-7B-E5-E5-84-FD-52-A4-49-B1-6B-02-51-D0-5D-27-F9-4A-B7-6C-BA-A6
-DA-89-0B-59-D8-EF-1E-15-9D

h512 = 48-6F-64-C1-91-78-79-41-7F-EF-08-2B-33-81-A4-E2-11-C3-24-F0-74-65-4C
-38-82-3A-7B-76-F8-30-AD-00-FA-1F-BA-E4-2B-12-85-C0-35-2F-22-75-24-BC-9A-B1
-62-54-28-8D-D6-86-3D-CC-D5-B9-F5-4A-1A-D0-54-1B

Программа для генерации рандомных данных:

```
from random import seed
from random import randint

# mes = [0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0
    x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0
    x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0
    x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30]
# b = "This is message, length=32 bytes".encode().hex()
deltaX = 0x01
seed(1)
with open("output.txt", "w") as f:
    for i in range(50):
        message = []
        f.write("message = {")
        for j in range(63):
            message.append(str(hex(randint(0x32, 0x63))))
            if j == 62:
                f.write(message[len(message) - 1])
            else:
                f.write(message[len(message) - 1] + ', ')
        f.write("};\n")
```

Программа для проверок данных (подсчет различных бит в хешах и XOR):

```
def xor(x, dx):
    ret = []
    for i in range(len(x) - len(dx)):
        dx.append(0x00)
    dx.reverse()
    for tdx, tx in zip(dx, x):
        ret.append(hex(tx ^ tdx))
    return ret


def bitcounter(aS, bs):
    counter = 0
    # print(aS[0], bs[0])
    for a, b in zip(aS, bs):
        # print(a, b)
        counter += len([1 for raz in bin(a ^ b)[2:] if raz == "1"])
    return counter


xOrig = [0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0
    x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0
    x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0
    x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30]

xhat = [0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0
```

```
    x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0
    x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0
    x35,0x34,0x33,0x32,0x31,0x30,0x39,0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x31]

print(bitcounter(xOrig, xhat))
```

Генерация графика на основе полученных данных:

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [114, 123, 126, 126, 126, 127, 129, 128,
    128, 129])
plt.ylabel('Количество различныхбитвхэшах')
plt.xlabel('Количество раундов')
plt.show()
```

4 Анализ алгоритма

Для анализа работы своего алгоритма я воспользовался методом, который называется «Дифференциальный криптоанализ». Он позволяет оценить качество алгоритма при различном количестве раундов.

При анализе я рассмотрел как изменение количества раундов алгоритма влияет на дифференциальную разность выходного хэша, а именно посчитал количество бит выходного хэша, которое различается для двух текстов с разницей в 1 бит.

Для простоты эксперимента я выбрал дифференциал $\Delta X = 1$ и проводил анализ для количества раундов, варьирующегося от 1 до 10 раундов.

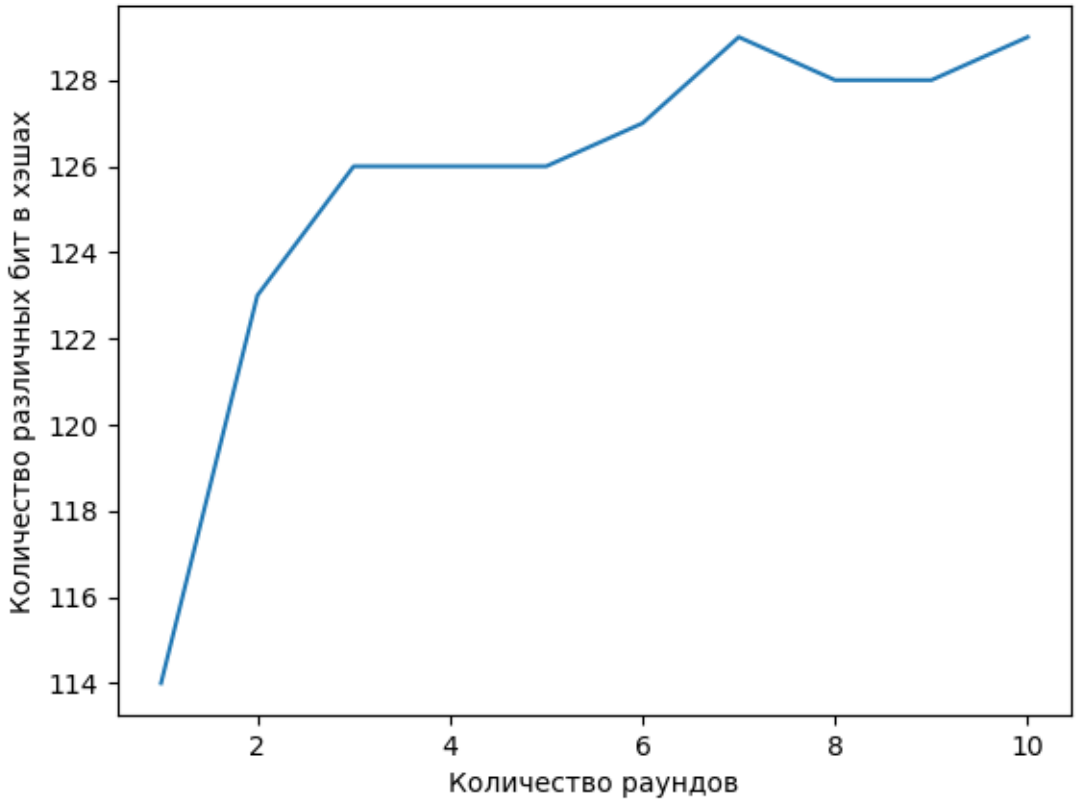
Для этого я генерировал случайным способом различные тексты X_i, и для каждого текста X_i произвел следующую последовательность действий:

1. Сгенерировал 2-ой текст $\overline{X_i} = X_i \oplus \Delta X$, отличающийся от исходного текста X_i лишь 1 битом.

2. Для каждого анализируемого количества раундов $\forall j : \ 1 \leq j \leq 10$:

   - Вычисляем значение хэшэй $Y_i^j = H^j(X_i)$ и $\overline{Y_i^j} = H^j(\overline{X_i})$ при помощи нашей хэш-функции H^j с настраиваемым количеством раундов j.
   - Вычисляем их дифференциал $\Delta Y_i^j = Y_i^j \oplus \overline{Y_i^j}$.
   - Подсчитываем количество битов $c_i^j = \sum_{k=0}^{|\Delta Y_i^j|-1} \Delta Y_i^j[k]$, которые различны у двух хэшей Y_i^j и $\overline{Y_i^j}$.

После подсчитаем среднее количество битов $c^j = \frac{\sum_{i=0}^{n-1}(c_i^j)}{n}$, которое оказалось различным для каждого анализируемого количества раундов $\forall j : 1 \leq j \leq 10$ алгоритма H^j.


Количество различных бит в хэшах
128
126
124
122
120
118
116
114
2
4
6
8
10
Количество раундов

5 Выводы

Выполнив третью лабораторную работу по курсу «Криптография», я получил новые знания о таком понятии, как *Криптографическая хэш-функция*. Мне пришлось узнать и разобраться в том, как они реализуются и даже довелось реализовать один и алгоритмов хэширования - *34.11-2012 (Стриборг)*. Также мне довелось познакомится с новым и несколько пугающим на первый взгляд явлением - дифференциальным криптоанализом и даже применить его простейшие элементы для анализа своего алгоритма.

Ещё пришлось приобрести для себя знания работы с шестнадцеричной системой, узнать новые свойства операции *Исключающее «или»*(а именно одно очень важное $(a \oplus b) \oplus b = a$ (реверсивность))

Список литературы

[1] *Open-source реализации отечественных криптоГОСТов*
URL: https://habr.com/ru/post/273055/

[2] *Основной алгоритм и реализация*
URL: hthttps://habr.com/ru/post/188152/

[3] *Поиск уточнений*
URL: https://www.streebog.net/ru/

[4] *Дифференциальный криптоанализ для самых маленьких*
URL: https://habr.com/ru/post/215527/

[5] *Весь исходный код*
URL: https://github.com/Ivan-Batyanovsky/Cryptography/tree/master/Lab3.