



Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

Криптографические методы защиты информации

Лабораторная работа №2

**Генерирование и тестирование псевдослучайных
шифрующих последовательностей**

Выполнил
Глазков Андрей
студент 4 курса
группы 06-852

Казань – 2023

Цели работы

- Разработка программы реализующая генератор М-последовательности на основе полинома обратной связи;
- Тестирование М-последовательности с помощью сериального теста;
- Тестирование М-последовательности с помощью корреляционного теста;
- Реализовать в программе шифрование/дешифрование двоичного файла с помощью М-последовательности.

Ход выполнения работы:

1. Для создания генератора написали `class Register`

```
class Register final {  
  
    private:  
  
        ParserSettings::Settings settings_;  
  
        std::vector<int> polynome_;  
  
        boost::dynamic_bitset<> digits_;  
  
        boost::dynamic_bitset<> MSeq_;  
  
  
    public:  
  
        Register(int argc, char** argv);  
  
        ~Register() = default;  
  
        Register(const Register&) = delete;  
  
        Register(Register&&) = delete;  
  
        Register& operator=(const Register&) = delete;  
  
        Register& operator=(Register&&) = delete;  
  
  
    public:  
  
        void runTask();  
  
  
    private:  
  
        void startTest();  
  
        void readFile();  
  
        void genDigits();  
  
        void parsePoly();  
};
```

```

        void generationMseq();

        void MSeq4File(std::string);

    private:

        void serialTest();

        void pokerTest();

        void corrTest();

    private:

        void crutches();

    };
} //Register

```

2. Реализовали функцию для генерации М-последовательности.

```

void Register::generationMseq() {

    std::set<boost::dynamic_bitset<>> findContain;

    std::stack<bool> s;

    int ret_xor;

    s.push(digits_[0]);

    while (findContain.count(digits_) == 0) {

        findContain.insert(digits_);

        bool ret_xor = 0;

        for (auto c : polynome_){

            ret_xor ^= digits_[c - 1];

        }

        digits_ = digits_ >> 1;

        digits_[digits_.size() - 1] = ret_xor;

        s.push(digits_[0]);

    }

    while (!s.empty()) {

        MSeq_.push_back(s.top());
    }
}

```

```

        s.pop();

    }

}

```

3. Написали реализацию сериального теста:

```

namespace Serialtest {

    template<typename T1, typename T2> using umap = std::unordered_map<T1, T2>;

    static umap<int, umap<double, double>> criticalHi {
        {2, {{0.95, 0.352}, {0.9, 0.584}, {0.8, 1.005}, {0.2, 4.6}, {0.1, 6.251},
        {0.05, 7.815}}},
        {3, {{0.95, 2.167}, {0.9, 2.833}, {0.8, 3.28}, {0.2, 9.8}, {0.1, 12.017},
        {0.05, 14.057}}},
        {4, {{0.95, 7.261}, {0.9, 8.547}, {0.8, 10.31}, {0.2, 19.31}, {0.1, 22.307},
        {0.05, 24.996}}}
    };

    class sTest final{
    private:
        boost::dynamic_bitset<> st_MSeq_;

        int st_k_;

        std::map<int, int> st_Mseries_;
        umap<int, double> st_Mfreq_;
        double st_hi_;
        size_t st_n_;
        double st_n_t_;

    public:
        sTest(const boost::dynamic_bitset<>& MSeq, int k) : st_MSeq_(MSeq),
        st_k_(k) {

            st_n_ = st_MSeq_.size() / st_k_;

            countSeries();

            countNs();

        }
    };
}

```

```

~sTest() {}};

void run(double alpha) const {

    std::stringstream s;

    s << "_____ \n";

    s << "_____ SERIAL_TEST _____ \n";

    s << "_____ \n";

    s << "_____ ЭМПИРИЧЕСКИЕ ЧАСТОТЫ _____ \n";

    for (auto&& curr : st_Mseries_) {

        for (int i = st_k_; i != 0; --i) {

            s << (curr.first >> (i - 1) & 1);

        }

        s << " " << curr.second << '\n';

    }

    s << "_____ ЭТАЛОННАЯ ЧАСТОТА _____ \n";

    s << "N_t = " << st_n_t_ << '\n';

    s << "_____ Критерий Hi^2 _____ \n";

    s << "Hi^2 = " << st_hi_ << '\n';

    s << "_____ \n";

    if (alpha == 0) {

        double Hi_min = criticalHi[st_k_][0.1];

        double Hi_max = criticalHi[st_k_][0.9];

        if (Hi_max <= st_hi_ && Hi_min >= st_hi_) {

            s << "_____ SERIAL_TEST_PASSED! _____ \n";

        } else {

            s << "_____ SERIAL_TEST_FAILED! _____ \n";

        }

    } else {

        double Hi_min = criticalHi[st_k_][alpha];

        if (Hi_min >= st_hi_) {

            s << "_____ SERIAL_TEST_PASSED! _____ \n";

        } else {

            s << "_____ SERIAL_TEST_FAILED! _____ \n";

        }

    }

}

```

```

        s << "_____\n\n";

        std::lock_guard<std::mutex> lock(Generator::mtx_);

        std::cout << s.str();

    }

private:

    void countSeries() {

        while (st_MSeq_.size() % st_k_ != 0) {

            st_MSeq_ = st_MSeq_ >> 1;

            st_MSeq_.pop_back();

        }

        for (size_t s = st_MSeq_.size(), e = 0; s != e; s -= st_k_) {

            int tmp = 0;

            for (size_t i = 0; i != st_k_; ++i) {

                tmp = tmp << 1 | st_MSeq_[s - i - 1];

            }

            st_Mseries_[tmp]++;

        }

        for (auto&& curr : st_Mfreq_) {

            st_Mfreq_[curr.first] = (0.0 + curr.second) / ((0.0 +
st_MSeq_.size()) / st_k_);

        }

    }

    void countNs() {

        st_n_t_ = (0.0 + st_n_) / std::pow(2, st_k_);

        for (auto&& curr : st_Mseries_) {

            st_hi_ += (0.0 + std::pow(0.0 + curr.second - st_n_t_, 2)) /
st_n_t_;

        }

    }

}; //class sTest

}; //namespace Serialtest

```

4. Написали реализацию покер теста:

```
namespace Pokertest{

    template<typename T1, typename T2> using umap = std::unordered_map<T1, T2>;

    static umap<double, double> criticalHi = {{0.95, 1.645}, {0.9, 2.2}, {0.8, 3.07},
{0.2, 8.56}, {0.1, 10.64}, {0.05, 12.59}};

    class pTest {

    private:

        int                q_ = 10;

        int                m_;

        double             hi_;

        std::vector<uint8_t> u_;

        u_int              count_[7];

    public:

        pTest() {}

        ~pTest() {}

    private:

        void makeCount(boost::dynamic_bitset<> tmp) {

            while (tmp.size() % 32 != 0) {

                tmp = tmp >> 1;

                tmp.pop_back();

            }

            m_ = tmp.size();

            for (size_t s = tmp.size(), e = 0; s != e; s-=32) {

                u_int curr = 0;

                for (size_t i = 0; i != 32; ++i){

                    curr = curr << 1 | tmp[s - i - 1];

                }

            }

        }

    };

}
```

```

    }

    curr = (0.0 + curr) / (std::pow(2,32) - 1) * q_;

    if (curr != 10) {

        u_.push_back(static_cast<uint8_t>(curr));

    } else {

        u_.push_back(static_cast<uint8_t>(9));

    }

}

while (u_.size() % 5 != 0) {

    u_.pop_back();

}

for (size_t s = 0, e = u_.size(); s != e; s+=5) {

    std::map<int,int> m;

    for (size_t i =0; i != 5; ++i){

        m[u_[s + i]]++;

    }

    if (m.size() == 5) {

        count_[0]++;

    } else if (m.size() == 4) {

        count_[1]++;

    } else if (m.size() == 3) {

        bool visit = false;

        for (auto&& c : m) {

            if (c.second == 3) {

                count_[3]++;

                visit = true;

            }

        }

        if (visit == false) {

            count_[2]++;

        }

    }

}

```



```

        } else if (m.size() == 2) {

            if ((*m.begin()).second == 3 || (*m.begin()).second == 2) {

                count_[4]++;

            } else {

                count_[5]++;

            }

        } else {

            count_[6]++;

        }

    }

}

public:

    void run(const boost::dynamic_bitset<>& MSeq, double alpha) {

        makeCount(MSeq);

        double p[7];

        p[0] = (0.3024 * MSeq.size() / 160);

        p[1] = (0.504 * MSeq.size() / 160);

        p[2] = (0.108 * MSeq.size() / 160);

        p[3] = (0.072 * MSeq.size() / 160);

        p[4] = (0.009 * MSeq.size() / 160);

        p[5] = (0.0045 * MSeq.size() / 160);

        p[6] = (0.0001 * MSeq.size() / 160);

        hi_ = 0.0;

        for (int i = 0; i < 7; ++i) {

            hi_ += std::pow(count_[i] - p[i], 2) / p[i];

        }

        std::stringstream s;

        s << "_____ \n";

```

```

        s << "_____POKER_TEST_____\n";

        s << "_____\n";

        s << " _ЭМПИР_И_ЭТАЛОННЫЕ_ЧАСТОТЫ_ \n";

        for (int i = 0; i < 7; ++i) {

            s << "N_" << i << "=" << count_[i] << "\t--\tP_" << i << "=" <<
p[i] << std::endl;

        }

        s << "_____Критерий_Hi^2_____\n";

        s << "Hi^2 =" << hi_ << '\n';

        s << "_____\n";

        if (alpha == 0.0) {

            double Hi_max = criticalHi[0.9];

            double Hi_min = criticalHi[0.1];

            if (Hi_max <= hi_ && Hi_min >= hi_) {

                s << "_____POKER_TEST_PASSED!_____\n";

            } else {

                s << "_____POKER_TEST_FAILED!_____\n";

            }

        } else {

            double Hi_min = criticalHi[alpha];

            if (Hi_min >= hi_) {

                s << "_____POKER_TEST_PASSED!_____\n";

            } else {

                s << "_____POKER_TEST_FAILED!_____\n";

            }

        }

        s << "_____\n\n";

        std::lock_guard<std::mutex> lock(Generator::mtx_);

        std::cout << s.str();

    }

}; //class pTest

} //namespace Pokertest

```

5. Написали реализацию для корреляционного теста:

```
namespace Corrtest {

    template<typename T1, typename T2> using umap = std::unordered_map<T1, T2>;

    class cTest final{

        friend class Generator::Register;

    private:

        std::vector<int> k_{1,2,8,9};

        std::map<int, double> r_;

    public:

        cTest(){}

        ~cTest(){}

    private:

        void autoCorr(const boost::dynamic_bitset<>& MSeq) {

            for (auto curr : k_) {

                double m_i = 0.0;

                for (size_t j = 0; j < MSeq.size() - curr; ++j) {

                    m_i += MSeq[j];

                }

                m_i *= 1.0 / (MSeq.size() - curr);

                double m_i_k = 0.0;

                for (size_t j = 0; j < MSeq.size(); ++j) {

                    m_i_k += MSeq[j];

                }

                m_i_k *= 1.0 / (MSeq.size() - curr);

                double d_i = 0.0;

                for (size_t j = 0; j < MSeq.size() - curr; ++j) {

                    d_i += std::pow(MSeq[j] - m_i, 2);

                }

                d_i *= 1.0 / (MSeq.size() - curr - 1);

                double d_i_k = 0.0;

                for (size_t j = 0; j < MSeq.size(); ++j) {
```

```

        d_i_k += std::pow(MSeq[j] - m_i_k, 2);
    }

    d_i_k *= 1.0 / (MSeq.size() - curr - 1);

    for (size_t j = 0; j < MSeq.size() - curr; ++j){
        r_[curr] += (MSeq[j] - m_i) * (MSeq[j + curr] - m_i_k);
    }

    r_[curr] /= (MSeq.size() - curr);
    r_[curr] /= std::sqrt(d_i * d_i_k);
    r_[curr] = std::abs(r_[curr]);
}

}

public:

    void run(const boost::dynamic_bitset<>& MSeq) {

        autoCorr(MSeq);

        double n = MSeq.size();

        double r_cr = 1.0 / (n-1) + 2.0 / (n - 2) * std::sqrt((n * (n - 3)) / (n
+ 1));

        std::stringstream s;

        s << "_____ \n";
        s << "_____CORR_TEST_____ \n";
        s << "_____ \n";

        bool pass = true;

        s << "_АВТОКОРРЕЛЯЦИОННАЯ_ФУНКЦИЯ_ \n";

        for (auto&& curr : r_) {

            if (curr.second > r_cr) {

                pass = false;

            }

            s << "R[" << curr.first << "]: " << curr.second << '\n';

        }

        s << "_____ \n";

        s << "_____КРИТИЧЕСКОЕ_ЗНАЧЕНИЕ_____ \n";

        s << "R[k]_cr: " << r_cr << '\n';

        s << "_____ \n";

```

```

        if (pass){

            s << "_____CORR_TEST_PASSED!_____\n";

        } else {

            s << "_____CORR_TEST_FAILED!_____\n";

        }

        s << "_____\n\n";

        std::lock_guard<std::mutex> lock(Generator::mtx_);

        std::cout << s.str();

    }

}; //class cTest

} //namespace Corrtest

```

6. Написали функции необходимые для шифрования файла:

```

namespace Cypher {

    std::vector<u_char> readFileBin(std::string filename) {

        std::ifstream f(filename, std::ios::binary);

        if (!f.is_open()) {

            throw std::runtime_error("Error ifstream " + filename);

        }

        std::vector<u_char> retVal;

        retVal.reserve(4097);

        std::string s;

        while (std::getline(f, s)) {

            std::for_each(s.begin(), s.end(), [&retVal](u_char
curr){retVal.push_back(curr);});

        }

        f.close();

        retVal.shrink_to_fit();

        return retVal;

    }

    std::vector<u_char> perfomingXOR(std::vector<u_char>& src, std::vector<u_char>&
key, std::string filename) {

        std::vector<u_char> ret(src.size());

```

```

        for (size_t s = 0, e = src.size(); s != e; ++s) {

            ret[s] = src[s] ^ key[s];

        }

        std::ofstream f(filename, std::ios::binary);

        if (!f.is_open()) {

            throw std::runtime_error("Error ofstream " + filename);

        }

        std::for_each(ret.begin(), ret.end(), [&f](u_char curr){f << curr;});

        f.close();

        return ret;

    }

void encrypt(boost::dynamic_bitset<> MSeq_, std::string filename) {

    try {

        std::vector<u_char> src = readFileBin(filename);

        std::vector<u_char> key;

        key.reserve(src.size());

        if (MSeq_.size() < 8) {

            throw std::runtime_error("BAD MSeq");

        }

        while (MSeq_.size() % 8 != 0) {

            MSeq_.pop_back();

        }

        for (size_t s = MSeq_.size(), e = 0; s != e; s-=8) {

            u_char curr = 0;

            for (size_t i = 0; i != 8; ++i){

                curr = curr << 1 | MSeq_[s - i - 1];

            }

            key.push_back(static_cast<uint8_t>(curr));

        }

        size_t i = 0;

```

```

        while (src.size() > key.size()){

            key.push_back(key[i]);

        }

        std::vector<u_char> encrypted = perfomingXOR(src, key, "encrypted.txt");

        perfomingXOR(encrypted, key, "decrypted.txt");

    } catch (const std::exception& e) {

        std::cerr << e.what();

    }

}

}

```

7. Провели тестирование согласно заданию 1. (Сгенерировали М-последовательность на основании ключа 1111 и полинома x^4+x^1+1 и вывели результат на экран)

```

→ Lab2 git:(main) x ./lab2 --file=task1 --poly=x4+x1+1
1111010110010001

```

8. Провели тестирование согласно заданию 2. (Для этого использовали М-последовательность на основании ключа 1001100100100111100011110 и полинома $x^{23}+x^9+1$)

```
+ Lab2 git:(main) x ./lab2 --file=key.txt --poly=x23+x9+1 --task=2

-----POKER_TEST-----
_ЭМПИР_И_ЭТАЛОННЫЕ_ЧАСТОТЫ_
N_0=238 --      P_0=247.739
N_1=428 --      P_1=412.899
N_2=78  --      P_2=88.4783
N_3=62  --      P_3=58.9856
N_4=9   --      P_4=7.37319
N_5=4   --      P_5=3.6866
N_6=0   --      P_6=0.0819244
-----Критерий_Hi^2-----
Hi^2 = 2.79766

-----POKER_TEST_PASSED!-----

-----SERIAL_TEST-----
_ЭМПИРИЧЕСКИЕ_ЧАСТОТЫ_
00 16424
01 16292
10 16398
11 16425
_ЭТАЛОННАЯ_ЧАСТОТА_
N_t = 16384.8
_Критерий_Hi^2_
Hi^2 = 0.72865

-----SERIAL_TEST_PASSED!-----

-----CORR_TEST-----
_АВТОКОРРЕЛЯЦИОННАЯ_ФУНКЦИЯ_
R[1]: 1.52577e-05
R[2]: 2.2887e-05
R[8]: 2.28872e-05
R[9]: 3.05168e-05

_КРИТИЧЕСКОЕ_ЗНАЧЕНИЕ_
R[k]_cr: 0.00553175

-----CORR_TEST_PASSED!-----
```

9. Провели тестирование согласно заданию 3.


```

→ Lab2 git:(main) x ./lab2 --file=key.txt --poly=x23+x9+1 --task=3 && ls -l
total 260
-rw-r--r-- 1 ayglazkov ayglazkov 1080 Mar 22 19:01 Makefile
-rw-r--r-- 1 ayglazkov ayglazkov 739 Mar 22 00:47 binfile
-rw-r--r-- 1 ayglazkov ayglazkov 739 Mar 22 19:18 decrypted.txt
-rw-r--r-- 1 ayglazkov ayglazkov 739 Mar 22 19:18 encrypted.txt
drwxr-xr-x 1 ayglazkov ayglazkov 78 Mar 21 23:50 inc
-rw-r--r-- 1 ayglazkov ayglazkov 25 Mar 22 00:11 key.txt
-rwxr-xr-x 1 ayglazkov ayglazkov 239416 Mar 22 19:09 lab2
drwxr-xr-x 1 ayglazkov ayglazkov 144 Mar 22 19:09 obj
drwxr-xr-x 1 ayglazkov ayglazkov 158 Mar 21 03:49 src
-rw-r--r-- 1 ayglazkov ayglazkov 4 Mar 21 05:22 task1

```

```

→ Lab2 git:(main) x cat binfile

```

Тест русского алфавита, а почему бы и не отработать. Я считаю. In Britain in the last few weeks, we have had several special television and radio programmes to mark the centenary of the disaster, and the film Titanic starring Leonardo DiCaprio and Kate Winslet - a film which you either love or hate - has been re-released in 3D, so that the iceberg looks even more scary. Some people have even dressed up in Edwardian dress and gone on a special cruise to the place where Titanic went down. On board the cruise ship, they have enjoyed some of the food that was on the menu in Titanic's restaurant. I am sure that they had a great time, but I do not think I would have enjoyed the trip.

```

→ Lab2 git:(main) x cat encrypted.txt

```

```

0000-L 0 00000 0 0 0 000s 0 0\00r 0A( 0 00005L 0 0N 005 000 00 | 000S: 00 00 00 id]zq 0 00D 000q= 0 000V 0H% 0 0# 0ZW 0
0tp 07 00 00 0kdi
C 0u 0 0 *C
3(iWo 0 00T 0 0
5CD 0 0 0
0 00r 0 0ZhUf 00 0!n 0z@G 0 [ 0 0=)as! 0b4 0 00 005 003 000000 0 00s.$ 00-` 00Y 0 00 000 00U 0
00B_c8 00 00 G 0/90o 00_ 0 0 0 0

000 00pýc$0 00o% 0 000000000 1 0kC< 0/? 000
mQ 0 000000 000 0\ 00M]T 0 00 0 0 00000 0I 002. 00 0/E 0000> 0 0 d' *000^H7 0
00 00 0 0EN 0v 00 0 000 00000000
0 0T 0 0 000 00
0~,t 0000000Kft 0 0\00hi 0x|!% 0 00 00 0 00 00

```

```

→ Lab2 git:(main) x cat decrypted.txt

```

Тест русского алфавита, а почему бы и не отработать. Я считаю. In Britain in the last few weeks, we have had several special television and radio programmes to mark the centenary of the disaster, and the film Titanic starring Leonardo DiCaprio and Kate Winslet - a film which you either love or hate - has been re-released in 3D, so that the iceberg looks even more scary. Some people have even dressed up in Edwardian dress and gone on a special cruise to the place where Titanic went down. On board the cruise ship, they have enjoyed some of the food that was on the menu in Titanic's restaurant. I am sure that they had a great time, but I do not think I would have enjoyed the trip.

10. Провели тестирование согласно заданию 4.

```
→ Lab2 git:(main) x ./lab2 --file=key.txt --poly=x23+x9+1 --task=4

-----_SERIAL_TEST_-----
-----_ЭМПИРИЧЕСКИЕ_ЧАСТОТЫ_-----
00 824
01 954
10 726
11 452
-----_ЭТАЛОННАЯ_ЧАСТОТА_-----
N_t = 739
-----_Критерий_Hi^2_-----
Hi^2 = 184.016

-----_SERIAL_TEST_FAILED!_-----

-----_POKER_TEST_-----
-----_ЭМПИР_И_ЭТАЛОННЫЕ_ЧАСТОТЫ_-----
N_0=0  --      P_0=11.1737
N_1=2  --      P_1=18.6228
N_2=11 --      P_2=3.9906
N_3=9  --      P_3=2.6604
N_4=6  --      P_4=0.33255
N_5=7  --      P_5=0.166275
N_6=1  --      P_6=0.003695
-----_Критерий_Hi^2_-----
Hi^2 =699.516

-----_POKER_TEST_FAILED!_-----

-----_CORR_TEST_-----
-----_АВТОКОРРЕЛЯЦИОННАЯ_ФУНКЦИЯ_-----
R[1]: 0.0175965
R[2]: 0.0706628
R[8]: 0.229136
R[9]: 0.00889638

-----_КРИТИЧЕСКОЕ_ЗНАЧЕНИЕ_-----
R[k]_cr: 0.0261805

-----_CORR_TEST_FAILED!_-----
```

Открытый текст не прошел все тесты. Это связано с тем, что текст осознанный и использует только английский и русский алфавиты, соответственно используется лишь определенный набор битов. Также

это связано с тем, что существует разница частот употребления букв в тесте.

11. Провели тестирование согласно заданию 5.

```
→ Lab2 git:(main) x ./lab2 --file=key.txt --poly=x23+x9+1 --task=5

-----POKER_TEST-----
_ЭМПИР_И_ЭТАЛОННЫЕ_ЧАСТОТЫ_
N_0=6  --      P_0=11.0981
N_1=21  --      P_1=18.4968
N_2=5   --      P_2=3.9636
N_3=4   --      P_3=2.6424
N_4=0   --      P_4=0.3303
N_5=0   --      P_5=0.16515
N_6=0   --      P_6=0.00367
-----Критерий_Hi^2-----
Hi^2 =4.14826
-----POKER_TEST_PASSED!-----

-----SERIAL_TEST-----
_ЭМПИРИЧЕСКИЕ_ЧАСТОТЫ_
00 741
01 746
10 700
11 749
_ЭТАЛОННАЯ_ЧАСТОТА_
N_t = 734
-----Критерий_Hi^2-----
Hi^2 = 2.14441
-----SERIAL_TEST_PASSED!-----

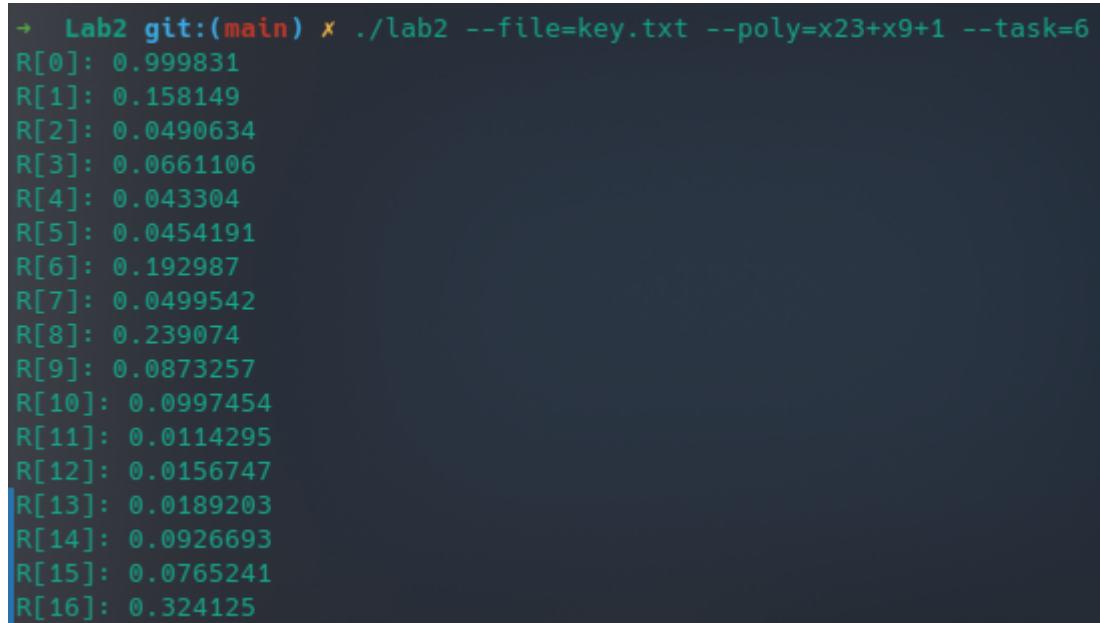
-----CORR_TEST-----
_АВТОКОРРЕЛЯЦИОННАЯ_ФУНКЦИЯ_
R[1]: 0.000844037
R[2]: 0.00545799
R[8]: 0.00987414
R[9]: 0.0100453
-----КРИТИЧЕСКОЕ_ЗНАЧЕНИЕ-----
R[k]_cr: 0.0262701
-----CORR_TEST_PASSED!-----
```

Шифротекст после поточного шифрования прошел корреляционный тест, покер тест и серийный тест, потому что

поточное шифрование использует ключ, который генерирует случайные данные для шифрования открытого текста.

Это означает, что каждый байт зашифрованного сообщения зависит только от соответствующего байта открытого текста и ключа, что делает его случайным и непредсказуемым. Как результат, в зашифрованном сообщении нет никакой структуры или повторяющихся паттернов, которые могут быть обнаружены с помощью корреляционного теста или других статистических тестов. Поэтому шифротекст после поточного шифрования проходит корреляционный тест, покер тест и сериальный тест без каких-либо проблем.

12. Провели тестирование согласно заданию 6.



```
→ Lab2 git:(main) x ./lab2 --file=key.txt --poly=x23+x9+1 --task=6
R[0]: 0.999831
R[1]: 0.158149
R[2]: 0.0490634
R[3]: 0.0661106
R[4]: 0.043304
R[5]: 0.0454191
R[6]: 0.192987
R[7]: 0.0499542
R[8]: 0.239074
R[9]: 0.0873257
R[10]: 0.0997454
R[11]: 0.0114295
R[12]: 0.0156747
R[13]: 0.0189203
R[14]: 0.0926693
R[15]: 0.0765241
R[16]: 0.324125
```

При уменьшении размера ключевой последовательности до $L=8$ бит, автокорреляционная функция $R[k]$ будет иметь высокий пик на $k = 0$ и ещё несколько пиков на $k = 8, 16, 24$ и т. д.

Это происходит из-за того, что поточный шифр использует ключевую последовательность для генерации псевдослучайной последовательности бит, которые затем используются для шифрования.

Когда ключевая последовательность имеет длину L , псевдослучайная последовательность также будет иметь периодичность L . Это приводит к тому, что сигнал, полученный из псевдослучайной последовательности, начнёт повторяться через каждые L бит, что отражается на графике автокорреляции.

Это означает, что уменьшение размера ключевой последовательности до $L=8$ бит сильно снижает стойкость шифра, так как можно обнаружить повторяющиеся участки в зашифрованном сообщении на всем диапазоне k .

Вывод

В ходе лабораторной работы был реализован поточный шифр с использованием линейной обратной связи, а так же были реализованы корреляционный, сериальный и покер тесты для проверки статистических свойств используемой M последовательности.

Во время работы было проверено, что открытый текст не проходит реализованные тесты, а шифротекст проходит.

Также на практике убедились в важности размера ключевой последовательности. Если уменьшить L до 8 бит, то это снижает криптостойкость шифра и появляется возможность обнаружить повторяющиеся участки в зашифрованном сообщении.

Поэтому необходимо использовать достаточно большой размер ключевой последовательности, чтобы обеспечить надежную защиту данных. Также важно выбирать ключи случайным образом и не использовать их повторно для зашифрования разных сообщений.