МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра кафедры САПР

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по дисциплине «Защита компьютерной информации»

Тема: Передача дейтаграмм по TCP между клиентами и сервером

Студент гр. 1335	________________	Максимов Ю.Е.
Преподаватель	________________	Горячев А. В.

Санкт-Петербург

2025

Оглавление

1. Введение

Алгоритм «Кузнечик» (англ. Kuznechik) — это современный симметричный блочный шифр, разработанный для обеспечения криптографической защиты информации в автоматизированных системах различного назначения. Он утверждён в качестве государственного стандарта Российской Федерации и стран СНГ — ГОСТ 34.12-2018.

2. История и стандартизация

«Кузнечик» был разработан Центром защиты информации и специальной связи ФСБ России совместно с АО «ИнфоТеКС» и принят как часть стандарта ГОСТ Р 34.12-2015, а затем утверждён межгосударственным стандартом ГОСТ 34.12-2018 с 1 июня 2019 года. Этот стандарт пришёл на смену ГОСТ 28147-89, обеспечив более высокий уровень безопасности и соответствие современным требованиям.

3. Основные характеристики алгоритма

- Тип шифра: симметричный блочный шифр.
- Размер блока: 128 бит.
- Длина ключа: 256 бит.
- Структура: основан на SP-сети (подстановочно-перестановочной сети), а не на сети Фейстеля, как многие другие блочные шифры.
- Назначение: обеспечение конфиденциальности, аутентичности и целостности информации.

4. Структура и принципы работы

Основу алгоритма составляет SP-сеть, в которой каждый раунд состоит из нелинейного (S) и линейного (L) преобразований, а также наложения итерационного ключа (операция X). В «Кузнечике» используется десять раундов: девять полных и один неполный, в котором выполняется только операция X.

Мастер-ключ длиной 256 бит делится на две части, из которых с помощью специальных преобразований и итерационных констант формируются десять раундовых ключей. Для генерации каждой пары раундовых ключей применяется восемь итераций с использованием линейного преобразования и S-преобразования.

Шифрование блока данных выполняется последовательным применением операций X, S и L с соответствующими раундовыми ключами. Расшифрование происходит в обратном порядке с использованием обратных преобразований S и L. Алгоритм реализован так, чтобы быть эффективным как в программных, так и в аппаратных решениях.

5. Криптостойкость и анализ безопасности

«Кузнечик» спроектирован с учётом современных требований к криптостойкости и предполагается устойчивым ко всем известным видам атак на блочные шифры. Тем не менее, исследователи отмечали, что значения S-блока были сгенерированы не случайно, а по скрытому алгоритму, что вызвало вопросы о потенциальных уязвимостях. На практике атаки на сокращённое число раундов требуют огромных вычислительных ресурсов и не применимы к полной версии алгоритма.

6. Применение и реализация

Стандарт ГОСТ 34.12-2018 рекомендует использовать «Кузнечик» для защиты информации различной степени секретности в государственных, коммерческих и промышленных системах. Алгоритм поддерживается в различных режимах работы (например, режимы простой замены, гаммирования и др.), которые определяются отдельными стандартами. Реализация возможна как на программном, так и на аппаратном уровне, включая компактные и высокоскоростные решения на ПЛИС.

7. Заключение

Алгоритм «Кузнечик» (ГОСТ 34.12-2018) является современным и эффективным средством симметричного блочного шифрования, соответствующим международным стандартам и требованиям к безопасности информации. Его структура обеспечивает высокую степень защиты и гибкость применения в различных сферах.

8. Код

Kuznechik.h

```
#include <vector>
```

```
#include <cassert>

#include <iostream>

#include <climits>

#include <fstream>

#include <string>

#include <cstring>

#include <omp.h>


void encrypt_file( const char* input_file_name, const char*
output_file_name, const char* key_1, const char* key_2);

void encrypt_file( const char* input_file_name, const char*
output_file_name, const char* hexadecimal_key);


void decrypt_file( const char* input_file_name, const char*
output_file_name, const char* key_1, const char* key_2);

void decrypt_file( const char* input_file_name, const char*
output_file_name, const char* hexadecimal_key);


const char* const hex_symbol_table = "0123456789abcdef";


std::string char_to_hex_string( char c);

std::string hex_to_string ( const std::string input_string);

std::string string_to_hex ( const std::string input_string);
```

```
struct block
{
    public:
        static const int size = 16;

        block();
        block( std::vector<unsigned char> input_string);
        block( std::string imput_string);

        unsigned char operator[] ( const int index) const;
        friend block operator ^ ( const block& a, const block& b);
        friend std::ostream& operator << ( std::ostream& os, const block& b);
        const std::vector<unsigned char>& get_data() const { return data; }

        void mod();
        void print();
    private:
        std::vector<unsigned char> data;
};

struct key_pair
{
    block key_1;
    block key_2;

    key_pair( block key_1, block key_2) : key_1( key_1), key_2( key_2) {}

    key_pair() = default;

};


std::ostream& operator << ( std::ostream& os, const block& b);


//////////////////////////


class kuznechik

{

    private:

        const unsigned char substitution_table[UCHAR_MAX + 1] =

        {

            (unsigned char)0xFC, (unsigned char)0xEE, (unsigned char)0xDD,
(unsigned char)0x11, (unsigned char)0xCF, (unsigned char)0x6E, (unsigned
char)0x31, (unsigned char)0x16,

            (unsigned char)0xFB, (unsigned char)0xC4, (unsigned char)0xFA,
(unsigned char)0xDA, (unsigned char)0x23, (unsigned char)0xC5, (unsigned
char)0x04, (unsigned char)0x4D,

            (unsigned char)0xE9, (unsigned char)0x77, (unsigned char)0xF0,
(unsigned char)0xDB, (unsigned char)0x93, (unsigned char)0x2E, (unsigned
char)0x99, (unsigned char)0xBA,
```

(unsigned char)0x17, (unsigned char)0x36, (unsigned char)0xF1, (unsigned char)0xBB, (unsigned char)0x14, (unsigned char)0xCD, (unsigned char)0x5F, (unsigned char)0xC1,

(unsigned char)0xF9, (unsigned char)0x18, (unsigned char)0x65, (unsigned char)0x5A, (unsigned char)0xE2, (unsigned char)0x5C, (unsigned char)0xEF, (unsigned char)0x21,

(unsigned char)0x81, (unsigned char)0x1C, (unsigned char)0x3C, (unsigned char)0x42, (unsigned char)0x8B, (unsigned char)0x01, (unsigned char)0x8E, (unsigned char)0x4F,

(unsigned char)0x05, (unsigned char)0x84, (unsigned char)0x02, (unsigned char)0xAE, (unsigned char)0xE3, (unsigned char)0x6A, (unsigned char)0x8F, (unsigned char)0xA0,

(unsigned char)0x06, (unsigned char)0x0B, (unsigned char)0xED, (unsigned char)0x98, (unsigned char)0x7F, (unsigned char)0xD4, (unsigned char)0xD3, (unsigned char)0x1F,

(unsigned char)0xEB, (unsigned char)0x34, (unsigned char)0x2C, (unsigned char)0x51, (unsigned char)0xEA, (unsigned char)0xC8, (unsigned char)0x48, (unsigned char)0xAB,

(unsigned char)0xF2, (unsigned char)0x2A, (unsigned char)0x68, (unsigned char)0xA2, (unsigned char)0xFD, (unsigned char)0x3A, (unsigned char)0xCE, (unsigned char)0xCC,

(unsigned char)0xB5, (unsigned char)0x70, (unsigned char)0x0E, (unsigned char)0x56, (unsigned char)0x08, (unsigned char)0x0C, (unsigned char)0x76, (unsigned char)0x12,

(unsigned char)0xBF, (unsigned char)0x72, (unsigned char)0x13, (unsigned char)0x47, (unsigned char)0x9C, (unsigned char)0xB7, (unsigned char)0x5D, (unsigned char)0x87,

(unsigned char)0x15, (unsigned char)0xA1, (unsigned char)0x96, (unsigned char)0x29, (unsigned char)0x10, (unsigned char)0x7B, (unsigned char)0x9A, (unsigned char)0xC7, (unsigned char)0xF3, (unsigned char)0x91, (unsigned char)0x78, (unsigned char)0x6F, (unsigned char)0x9D, (unsigned char)0x9E, (unsigned char)0xB2, (unsigned char)0xB1,

(unsigned char)0x32, (unsigned char)0x75, (unsigned char)0x19, (unsigned char)0x3D, (unsigned char)0xFF, (unsigned char)0x35, (unsigned char)0x8A, (unsigned char)0x7E,

(unsigned char)0x6D, (unsigned char)0x54, (unsigned char)0xC6, (unsigned char)0x80, (unsigned char)0xC3, (unsigned char)0xBD, (unsigned char)0x0D, (unsigned char)0x57,

(unsigned char)0xDF, (unsigned char)0xF5, (unsigned char)0x24, (unsigned char)0xA9, (unsigned char)0x3E, (unsigned char)0xA8, (unsigned char)0x43, (unsigned char)0xC9,

(unsigned char)0xD7, (unsigned char)0x79, (unsigned char)0xD6, (unsigned char)0xF6, (unsigned char)0x7C, (unsigned char)0x22, (unsigned char)0xB9, (unsigned char)0x03,

(unsigned char)0xE0, (unsigned char)0x0F, (unsigned char)0xEC, (unsigned char)0xDE, (unsigned char)0x7A, (unsigned char)0x94, (unsigned char)0xB0, (unsigned char)0xBC,

(unsigned char)0xDC, (unsigned char)0xE8, (unsigned char)0x28, (unsigned char)0x50, (unsigned char)0x4E, (unsigned char)0x33, (unsigned char)0x0A, (unsigned char)0x4A,

(unsigned char)0xA7, (unsigned char)0x97, (unsigned char)0x60, (unsigned char)0x73, (unsigned char)0x1E, (unsigned char)0x00, (unsigned char)0x62, (unsigned char)0x44,

(unsigned char)0x1A, (unsigned char)0xB8, (unsigned char)0x38, (unsigned char)0x82, (unsigned char)0x64, (unsigned char)0x9F, (unsigned char)0x26, (unsigned char)0x41,

(unsigned char)0xAD, (unsigned char)0x45, (unsigned char)0x46, (unsigned char)0x92, (unsigned char)0x27, (unsigned char)0x5E, (unsigned char)0x55, (unsigned char)0x2F,

```
        (unsigned char)0x8C, (unsigned char)0xA3, (unsigned char)0xA5,
(unsigned char)0x7D, (unsigned char)0x69, (unsigned char)0xD5, (unsigned
char)0x95, (unsigned char)0x3B,

        (unsigned char)0x07, (unsigned char)0x58, (unsigned char)0xB3,
(unsigned char)0x40, (unsigned char)0x86, (unsigned char)0xAC, (unsigned
char)0x1D, (unsigned char)0xF7,

        (unsigned char)0x30, (unsigned char)0x37, (unsigned char)0x6B,
(unsigned char)0xE4, (unsigned char)0x88, (unsigned char)0xD9, (unsigned
char)0xE7, (unsigned char)0x89,

        (unsigned char)0xE1, (unsigned char)0x1B, (unsigned char)0x83,
(unsigned char)0x49, (unsigned char)0x4C, (unsigned char)0x3F, (unsigned
char)0xF8, (unsigned char)0xFE,

        (unsigned char)0x8D, (unsigned char)0x53, (unsigned char)0xAA,
(unsigned char)0x90, (unsigned char)0xCA, (unsigned char)0xD8, (unsigned
char)0x85, (unsigned char)0x61,

        (unsigned char)0x20, (unsigned char)0x71, (unsigned char)0x67,
(unsigned char)0xA4, (unsigned char)0x2D, (unsigned char)0x2B, (unsigned
char)0x09, (unsigned char)0x5B,

        (unsigned char)0xCB, (unsigned char)0x9B, (unsigned char)0x25,
(unsigned char)0xD0, (unsigned char)0xBE, (unsigned char)0xE5, (unsigned
char)0x6C, (unsigned char)0x52,

        (unsigned char)0x59, (unsigned char)0xA6, (unsigned char)0x74,
(unsigned char)0xD2, (unsigned char)0xE6, (unsigned char)0xF4, (unsigned
char)0xB4, (unsigned char)0xC0,

        (unsigned char)0xD1, (unsigned char)0x66, (unsigned char)0xAF,
(unsigned char)0xC2, (unsigned char)0x39, (unsigned char)0x4B, (unsigned
char)0x63, (unsigned char)0xB6

    };
```

```
        const unsigned char substitution_table_reversed[UCHAR_MAX + 1] =
        {
            (unsigned char)0xA5, (unsigned char)0x2D, (unsigned char)0x32,
(unsigned char)0x8F, (unsigned char)0x0E, (unsigned char)0x30, (unsigned
char)0x38, (unsigned char)0xC0,
            (unsigned char)0x54, (unsigned char)0xE6, (unsigned char)0x9E,
(unsigned char)0x39, (unsigned char)0x55, (unsigned char)0x7E, (unsigned
char)0x52, (unsigned char)0x91,
            (unsigned char)0x64, (unsigned char)0x03, (unsigned char)0x57,
(unsigned char)0x5A, (unsigned char)0x1C, (unsigned char)0x60, (unsigned
char)0x07, (unsigned char)0x18,
            (unsigned char)0x21, (unsigned char)0x72, (unsigned char)0xA8,
(unsigned char)0xD1, (unsigned char)0x29, (unsigned char)0xC6, (unsigned
char)0xA4, (unsigned char)0x3F,
            (unsigned char)0xE0, (unsigned char)0x27, (unsigned char)0x8D,
(unsigned char)0x0C, (unsigned char)0x82, (unsigned char)0xEA, (unsigned
char)0xAE, (unsigned char)0xB4,
            (unsigned char)0x9A, (unsigned char)0x63, (unsigned char)0x49,
(unsigned char)0xE5, (unsigned char)0x42, (unsigned char)0xE4, (unsigned
char)0x15, (unsigned char)0xB7,
            (unsigned char)0xC8, (unsigned char)0x06, (unsigned char)0x70,
(unsigned char)0x9D, (unsigned char)0x41, (unsigned char)0x75, (unsigned
char)0x19, (unsigned char)0xC9,
            (unsigned char)0xAA, (unsigned char)0xFC, (unsigned char)0x4D,
(unsigned char)0xBF, (unsigned char)0x2A, (unsigned char)0x73, (unsigned
char)0x84, (unsigned char)0xD5,
            (unsigned char)0xC3, (unsigned char)0xAF, (unsigned char)0x2B,
(unsigned char)0x86, (unsigned char)0xA7, (unsigned char)0xB1, (unsigned
char)0xB2, (unsigned char)0x5B,
            (unsigned char)0x46, (unsigned char)0xD3, (unsigned char)0x9F,
(unsigned char)0xFD, (unsigned char)0xD4, (unsigned char)0x0F, (unsigned
char)0x9C, (unsigned char)0x2F,

            (unsigned char)0x9B, (unsigned char)0x43, (unsigned char)0xEF,
(unsigned char)0xD9, (unsigned char)0x79, (unsigned char)0xB6, (unsigned
char)0x53, (unsigned char)0x7F,

            (unsigned char)0xC1, (unsigned char)0xF0, (unsigned char)0x23,
(unsigned char)0xE7, (unsigned char)0x25, (unsigned char)0x5E, (unsigned
char)0xB5, (unsigned char)0x1E,

            (unsigned char)0xA2, (unsigned char)0xDF, (unsigned char)0xA6,
(unsigned char)0xFE, (unsigned char)0xAC, (unsigned char)0x22, (unsigned
char)0xF9, (unsigned char)0xE2,

            (unsigned char)0x4A, (unsigned char)0xBC, (unsigned char)0x35,
(unsigned char)0xCA, (unsigned char)0xEE, (unsigned char)0x78, (unsigned
char)0x05, (unsigned char)0x6B,

            (unsigned char)0x51, (unsigned char)0xE1, (unsigned char)0x59,
(unsigned char)0xA3, (unsigned char)0xF2, (unsigned char)0x71, (unsigned
char)0x56, (unsigned char)0x11,

            (unsigned char)0x6A, (unsigned char)0x89, (unsigned char)0x94,
(unsigned char)0x65, (unsigned char)0x8C, (unsigned char)0xBB, (unsigned
char)0x77, (unsigned char)0x3C,

            (unsigned char)0x7B, (unsigned char)0x28, (unsigned char)0xAB,
(unsigned char)0xD2, (unsigned char)0x31, (unsigned char)0xDE, (unsigned
char)0xC4, (unsigned char)0x5F,

            (unsigned char)0xCC, (unsigned char)0xCF, (unsigned char)0x76,
(unsigned char)0x2C, (unsigned char)0xB8, (unsigned char)0xD8, (unsigned
char)0x2E, (unsigned char)0x36,

            (unsigned char)0xDB, (unsigned char)0x69, (unsigned char)0xB3,
(unsigned char)0x14, (unsigned char)0x95, (unsigned char)0xBE, (unsigned
char)0x62, (unsigned char)0xA1,
```

(unsigned char)0x3B, (unsigned char)0x16, (unsigned char)0x66, (unsigned char)0xE9, (unsigned char)0x5C, (unsigned char)0x6C, (unsigned char)0x6D, (unsigned char)0xAD,

(unsigned char)0x37, (unsigned char)0x61, (unsigned char)0x4B, (unsigned char)0xB9, (unsigned char)0xE3, (unsigned char)0xBA, (unsigned char)0xF1, (unsigned char)0xA0,

(unsigned char)0x85, (unsigned char)0x83, (unsigned char)0xDA, (unsigned char)0x47, (unsigned char)0xC5, (unsigned char)0xB0, (unsigned char)0x33, (unsigned char)0xFA,

(unsigned char)0x96, (unsigned char)0x6F, (unsigned char)0x6E, (unsigned char)0xC2, (unsigned char)0xF6, (unsigned char)0x50, (unsigned char)0xFF, (unsigned char)0x5D,

(unsigned char)0xA9, (unsigned char)0x8E, (unsigned char)0x17, (unsigned char)0x1B, (unsigned char)0x97, (unsigned char)0x7D, (unsigned char)0xEC, (unsigned char)0x58,

(unsigned char)0xF7, (unsigned char)0x1F, (unsigned char)0xFB, (unsigned char)0x7C, (unsigned char)0x09, (unsigned char)0x0D, (unsigned char)0x7A, (unsigned char)0x67,

(unsigned char)0x45, (unsigned char)0x87, (unsigned char)0xDC, (unsigned char)0xE8, (unsigned char)0x4F, (unsigned char)0x1D, (unsigned char)0x4E, (unsigned char)0x04,

(unsigned char)0xEB, (unsigned char)0xF8, (unsigned char)0xF3, (unsigned char)0x3E, (unsigned char)0x3D, (unsigned char)0xBD, (unsigned char)0x8A, (unsigned char)0x88,

(unsigned char)0xDD, (unsigned char)0xCD, (unsigned char)0x0B, (unsigned char)0x13, (unsigned char)0x98, (unsigned char)0x02, (unsigned char)0x93, (unsigned char)0x80,

(unsigned char)0x90, (unsigned char)0xD0, (unsigned char)0x24, (unsigned char)0x34, (unsigned char)0xCB, (unsigned char)0xED, (unsigned char)0xF4, (unsigned char)0xCE,

```
            (unsigned char)0x99, (unsigned char)0x10, (unsigned char)0x44,
(unsigned char)0x40, (unsigned char)0x92, (unsigned char)0x3A, (unsigned
char)0x01, (unsigned char)0x26,

            (unsigned char)0x12, (unsigned char)0x1A, (unsigned char)0x48,
(unsigned char)0x68, (unsigned char)0xF5, (unsigned char)0x81, (unsigned
char)0x8B, (unsigned char)0xC7,

            (unsigned char)0xD6, (unsigned char)0x20, (unsigned char)0x0A,
(unsigned char)0x08, (unsigned char)0x00, (unsigned char)0x4C, (unsigned
char)0xD7, (unsigned char)0x74

        };


        const unsigned char mask[block::size] =

        {

           (unsigned char)1, (unsigned char)148, (unsigned char)32, (unsigned
char)133, (unsigned char)16, (unsigned char)194, (unsigned char)192,
(unsigned char)1,

           (unsigned char)251, (unsigned char)1, (unsigned char)192, (unsigned
char)194, (unsigned char)16, (unsigned char)133, (unsigned char)32,
(unsigned char)148

        };


        const int number_of_iteration_keys = 10;


        std::vector<block> data;

        std::vector<block> iteration_constants;

        std::vector<block> iteration_keys;
        void read_file_to_data_buffer( const char* file_name, bool is_hex =
false);
        void calculate_iteration_constants();

        void generate_iteraion_keys( block key_1, block key_2);

        unsigned char get_mask_value( int index) const;
        unsigned char get_substituted_value( int index) const;
        unsigned char get_reversed_substituted_value( int index) const;

        block get_iteration_constant( int index) const;
        block get_iteration_key( int index) const;
        void set_iteration_key( int index, const block value);

        static unsigned char GF_mul( unsigned char a, unsigned char b);

        block L( const block input_block);
        block R( const block input_block);
        block S( const block input_block);

        block L_reversed( const block input_block);
        block R_reversed( const block input_block);
        block S_reversed( const block input_block);


        key_pair F( const key_pair input_key_pair, const block
iteraion_constant);


        block encrypt_block( const block input_block);

        block decrypt_block( const block input_block);


        void write_to_file( const char* output_file, bool use_hex = false);


    public:

        kuznechik( const char* file_name, const block key_1, const block
key_2);

        kuznechik( const char* file_name, const char* hexadecimal_key);


        void encrypt_data( const char* output_file_name, bool use_hex =
false);

        void decrypt_data( const char* output_file_name, bool use_hex =
false);

};
```

kuznechik.cpp

```
#include "kuznechik.h"
```

```
void encrypt_file( const char* input_file_name, const char*
output_file_name, const char* key_1, const char* key_2)
{
   kuznechik encryptor( input_file_name, block( key_1), block( key_2));
   encryptor.encrypt_data( output_file_name);
}

void encrypt_file( const char* input_file_name, const char*
output_file_name, const char* hexadecimal_key)
{
   kuznechik encryptor( input_file_name, hexadecimal_key);
   encryptor.encrypt_data( output_file_name);
}

void decrypt_file( const char* input_file_name, const char*
output_file_name, const char* key_1, const char* key_2)
{
   kuznechik encryptor( input_file_name, block( key_1), block( key_2));
   encryptor.decrypt_data( output_file_name);
}

void decrypt_file( const char* input_file_name, const char*
output_file_name, const char* hexadecimal_key)
{
   kuznechik encryptor( input_file_name, hexadecimal_key);
   encryptor.decrypt_data( output_file_name);
}
```

```
std::string hex_to_string ( const std::string input_string)
{
    std::string output_string;

    for ( int i = 0; i < input_string.length(); i += 2)
    {
        const char* p = std::lower_bound( hex_symbol_table,
hex_symbol_table + 16, input_string[i]);
        const char* q = std::lower_bound( hex_symbol_table,
hex_symbol_table + 16, input_string[i + 1]);
        output_string.push_back((( p - hex_symbol_table) << 4) | ( q -
hex_symbol_table));
    }
    return output_string;
}

std::string string_to_hex ( const std::string input_string)
{
    std::string output_string;
    for ( int i = 0; i < input_string.length(); i ++)
        output_string += char_to_hex_string( input_string[i]);
    return output_string;
}

std::string char_to_hex_string( char c)
{
    std::string hex = "0123456789abcdef";
    std::string hex_str;
    hex_str += hex[int(int( c)/16)];
    hex_str += hex[int( c) % 16];
    return hex_str;
}

void kuznechik::encrypt_data( const char* output_file_name, bool use_hex)
{
    double start;
    double end;
    start = omp_get_wtime();
    #pragma omp parallel
    {
        #pragma omp for
         for ( int i = 0; i < data.size(); i++)
          data[i] = encrypt_block( data[i]);
    }
    end = omp_get_wtime();
    std::cout << "Encryption time: "  << end - start << std::endl;
    write_to_file( output_file_name, use_hex);
}

void kuznechik::decrypt_data( const char* output_file_name, bool use_hex)
{
//    omp_set_num_threads(OMP_NUM_THREADS);
    double start;
    double end;
    start = omp_get_wtime();
    #pragma omp parallel
    {
        #pragma omp for
        for ( int i = 0; i < data.size(); i++)
            data[i] = decrypt_block( data[i]);
    }
    end = omp_get_wtime();
    std::cout << "Decryption time: "  << end - start << std::endl;
    write_to_file( output_file_name, use_hex);
}

kuznechik::kuznechik( const char* file_name, const block key_1, const
block key_2)
{
    iteration_keys.resize( number_of_iteration_keys);
    read_file_to_data_buffer( file_name);
    calculate_iteration_constants();
    generate_iteraion_keys( key_1, key_2);
}

kuznechik::kuznechik( const char* file_name, const char* hexadecimal_key)
{
    assert( strlen( hexadecimal_key) == 64 && "Wrong key");
    iteration_keys.resize( number_of_iteration_keys);
    read_file_to_data_buffer( file_name, true);
    calculate_iteration_constants();
    std::string ascii_key_pair = hex_to_string( hexadecimal_key);
    generate_iteraion_keys( ascii_key_pair.substr( 0, 15),
ascii_key_pair.substr( 16));
}

void kuznechik::read_file_to_data_buffer( const char* file_name, bool
is_hex)
{
    std::ifstream input_file_stream( file_name);
    assert( input_file_stream && "Can't find file");
    std::string file_content( ( std::istreambuf_iterator<char>(
input_file_stream)), std::istreambuf_iterator<char>());

    if (is_hex == true)
        file_content = hex_to_string( file_content);

    int length_of_the_trailing_string = file_content.length() % block::size;

    for ( int i = 0; i + block::size <= file_content.length(); i += block::size)
        data.push_back( block( file_content.substr( i, block::size)));

    if (length_of_the_trailing_string != 0)
    {
       std::string trailing_content = file_content.substr( file_content.length() -
length_of_the_trailing_string);
        for ( int i = 0; i < block::size - length_of_the_trailing_string; i++)
```

```
            trailing_content.push_back( ' ');
        data.push_back( block( trailing_content));
    }
}



void kuznechik::calculate_iteration_constants()
{
    //                        123456789012345     15 zeros
    std::string zero_string( "000000000000000");
    for( int i = 0; i < 32; i++)
    {
        std::string iterational_string;
        iterational_string.push_back( (char)i);
        iterational_string += zero_string;
        iteration_constants.push_back( L( block( iterational_string)));
    }
}

unsigned char kuznechik::get_mask_value( int index) const
{
    assert( index >= 0 && index < block::size && "Wrong index value");
    return mask[index];
}

unsigned char kuznechik::get_substituted_value( int index) const
{
    assert( index >= 0 && index <= UCHAR_MAX && "Wrong index
value");
    return substitution_table[index];
}

unsigned char kuznechik::get_reversed_substituted_value( int index) const
{
    assert( index >= 0 && index <= UCHAR_MAX && "Wrong index
value");
    return substitution_table_reversed[index];
}

block kuznechik::get_iteration_constant( int index) const
{
    assert( index >= 0 && index < 2 * block::size && "Wrong index value");
    return iteration_constants.at( index);
}

block kuznechik::get_iteration_key( int index) const
{
    assert( index >= 0 && index < number_of_iteration_keys && "Wrong
index value");
    return iteration_keys[index];
}

void kuznechik::set_iteration_key( int index, const block value)
{
    assert( index >= 0 && index < number_of_iteration_keys && "Wrong
index value");
    iteration_keys[index] = value;
}

block kuznechik::S( const block input_block)
{
        std::vector<unsigned char> transformed_data;
        for ( int i = 0 ; i < block::size; i++)
                transformed_data.push_back( get_substituted_value(
input_block[i]));
        return block( transformed_data);
}

block kuznechik::S_reversed( const block input_block)
{
        std::vector<unsigned char> transformed_data;

        for ( int i = 0 ; i < block::size; i++)
                transformed_data.push_back( get_reversed_substituted_value(
input_block[i]));
        return block( transformed_data);
}

unsigned char kuznechik::GF_mul( unsigned char a, unsigned char b)
{
        unsigned char c = 0;
        for ( int i = 0; i < 8; i++)
    {
                if ( ( b & 1) == 1)
                        c ^= a;
                unsigned char hi_bit = (char)( a & 0x80);
                a <<= 1;
                if( hi_bit == 0)
                        a ^= 0xC3;
                b >>= 1;
        }
        return c;
}

 block kuznechik::R( const block input_block)
 {
    std::vector<unsigned char> transformed_data( block::size);
        unsigned char trailing_symbol = 0;

        for ( int i = block::size - 1; i >= 0; i--)
    {
                if( i == 0)
                        transformed_data[block::size] = input_block[i];
                else
                        transformed_data[i-1] = input_block[i];
                trailing_symbol ^= GF_mul( input_block[i],
get_mask_value(i));
        }
        transformed_data[block::size - 1] = trailing_symbol;
        return block( transformed_data);
}

block kuznechik::R_reversed( const block input_block)
{
    std::vector<unsigned char> transformed_data( block::size);
    unsigned char leading_symbol = input_block[block::size - 1];

    for ( int i = 1; i < block::size; i++)
    {
        transformed_data[i] = input_block[i - 1];
        leading_symbol ^= GF_mul( transformed_data[i], get_mask_value(i));
    }
    transformed_data[0] = leading_symbol;
    return block( transformed_data);
}

block kuznechik::L( const block input_block)
{
        block transformed_block = input_block;
    for ( int i = 0; i < block::size; i++)
        transformed_block = R( transformed_block);
    return transformed_block;
}
```

```
block kuznechik::L_reversed( const block input_block)
{
    block transformed_block = input_block;
    for ( int i = 0; i < block::size; i++)
        transformed_block = R_reversed( transformed_block);
    return transformed_block;
}

key_pair kuznechik::F( const key_pair input_key_pair, const block
iteraion_constant)
{
    block returned_key_1;
    block returned_key_2 = input_key_pair.key_1;
    returned_key_1 = L( S( input_key_pair.key_2 ^ iteraion_constant)) ^
returned_key_2;
    return key_pair( returned_key_1, returned_key_2);
}

void kuznechik::generate_iteraion_keys( block key_1, block key_2)
{
    iteration_keys[0] = key_1;
    iteration_keys[1] = key_2;
    key_pair key_pair_1_2( key_1, key_2);
    key_pair key_pair_3_4;
    for( int i = 0; i < 4; i++)
    {
        key_pair_3_4 = F( key_pair_1_2, get_iteration_constant(0 + 8 * i));
        key_pair_1_2 = F( key_pair_3_4, get_iteration_constant(1 + 8 * i));
        key_pair_3_4 = F( key_pair_1_2, get_iteration_constant(2 + 8 * i));
        key_pair_1_2 = F( key_pair_3_4, get_iteration_constant(3 + 8 * i));
        key_pair_3_4 = F( key_pair_1_2, get_iteration_constant(4 + 8 * i));
        key_pair_1_2 = F( key_pair_3_4, get_iteration_constant(5 + 8 * i));
        key_pair_3_4 = F( key_pair_1_2, get_iteration_constant(6 + 8 * i));
        key_pair_1_2 = F( key_pair_3_4, get_iteration_constant(7 + 8 * i));
        set_iteration_key( 2 * i + 2, key_pair_1_2.key_1);
        set_iteration_key( 2 * i + 3, key_pair_1_2.key_2);
    }
}

block kuznechik::encrypt_block( const block input_block)
{
    block returned_block = input_block;
    for( int i = 0; i < 9; i++)
    {
        returned_block = get_iteration_key( i) ^ returned_block;
        returned_block = S( returned_block);
        returned_block = L( returned_block);
    }
    returned_block = returned_block ^ get_iteration_key( 9);
    return returned_block;
}

block kuznechik::decrypt_block( const block input_block)
{
    block returned_block = input_block ^ get_iteration_key( 9);
    for( int i = 8; i >= 0; i--)
    {
        returned_block = L_reversed( returned_block);
        returned_block = S_reversed( returned_block);
        returned_block = get_iteration_key(i) ^ returned_block;
    }
    return returned_block;
}

void kuznechik::write_to_file( const char* output_file, bool use_hex)
{
    std::ofstream output_stream;
    output_stream.open( output_file);
    assert( output_stream.is_open() && "Can't open file");
    for ( block i : data)
        if ( use_hex == true)
            output_stream << hex_to_string( std::string( i.get_data().begin(),
i.get_data().end()));
        else
           output_stream << std::string( i.get_data().begin(), i.get_data().end());
}
////////////////////////

block::block( std::vector<unsigned char> input_string) : data( input_string) {
assert ( input_string.size() == size); };
```

```
block::block() { data.resize( size); }

block::block( std::string input_string)
{
   assert( input_string.length() == size && "Wrong length of the block");
   for ( int i = 0; i < size; i++)
      data.push_back( input_string[i]);
}

unsigned char block::operator[] ( const int index) const
{
   assert ( index < size && "given index causes overflow");
   return data[index];
}

block operator ^ ( const block& a, const block& b)
{
   block result;
   for( int i = 0; i < result.size; i++)
            result.data[i] = b[i]^a[i];
   return result;
}

std::ostream& operator << ( std::ostream& os, const block& b)
{
   return os  << b.data << std::endl;
}
```

```
void block::print()
{
    for( int i : data)
        std::cout << (unsigned char)i;
    std::cout << std::endl;
}
```

9. Список литературы

1. ГОСТ 34.12—2018 «Информационная технология. Криптографическая защита информации. Блочные шифры».

2. Кузнечик (шифр) — Википедия.

3. Извращения с импортозамещением. Работаем с алгоритмом блочного шифрования «Кузнечик» из ГОСТ 34.12—2015 — xakep.ru.

4. Способы реализации алгоритма «Кузнечик» на программируемых логических интегральных схемах — Радиопромышленность.

5. Скачать ГОСТ 34.12-2018 Информационная технология — meganorm.ru.