**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

2. История и стандартизация

3. Основные характеристики алгоритма

4. Структура и принципы работы

5. Криптостойкость и анализ безопасности

6. Применение и реализация

7. Заключение  
8. Код

9. Список литературы

# 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.