Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Индивидуальная практическая работа № 1

Алгоритмы шифрования DES и ГОСТ

Выполнил   
студент гр. 653502

Турцевич И.М.

Проверил

Артемьев В. С.

Минск, 2019

**Содержание**

[Алгоритмы шифрования DES и ГОСТ 1](#_Toc26129116)

[1. Цель работы 3](#_Toc26129117)

[2. Краткие теоретические сведения 4](#_Toc26129118)

[2.1 Алгоритм шифрования DES 4](#_Toc26129119)

[2.1.1 Начальная перестановка 4](#_Toc26129120)

[2.1.2 Циклы шифрования 4](#_Toc26129121)

[2.1.3 Конечная перестановка 7](#_Toc26129122)

[2.1.4 Схема расшифровки 8](#_Toc26129123)

[2.2 Алгоритм шифрования ГОСТ 28147 9](#_Toc26129124)

[3. Примеры работы программы 11](#_Toc26129125)

[3.1 Алгоритм шифрования DES 11](#_Toc26129126)

[3.1 Алгоритм шифрования ГОСТ 28147 11](#_Toc26129127)

[4. Выводы 13](#_Toc26129128)

[Приложение 1. Исходный код реализации алгоритма DES 14](#_Toc26129129)

[Приложение 2. Исходный код реализации алгоритма ГОСТ 28147 22](#_Toc26129130)

# **1. Цель работы**

Необходимо создать реализацию на языке программирования С++ алгоритмов шифрования DES и ГОСТ 28147.

# **2. Краткие теоретические сведения**

## **2.1 Алгоритм шифрования DES**

Исходный текст — блок 64 бит. Процесс шифрования состоит из начальной перестановки, 16 циклов шифрования и конечной перестановки.

**2.1.1 Начальная перестановка**

Исходный текст\displaystyle T} (блок 64 бит) преобразуется c помощью начальной перестановки, {\displaystyle \mathrm {IP} }которая определяется таблицей 1.



Таблица 1. Начальная перестановка

### 2.1.2 Циклы шифрования

Полученный после начальной перестановки 64-битовый блок IP(T) участвует в 16 циклах преобразования Фейстеля. разбивается на две части {\displaystyle L\_{0},R\_{0}}, (32 старших битов и 32 младших битов блока {\displaystyle T\_{0}} соответственно).{\displaystyle L\_{0}R\_{0}}юю Пусть {\displaystyle T\_{i-1}=L\_{i-1}R\_{i-1}} – результат итерации, тогда результат -ой итерации {\displaystyle T\_{i}=L\_{i}R\_{i}}определяется:

{\displaystyle L\_{i}=R\_{i-1}}

{\displaystyle L\_{i}=R\_{i-1}}

Рассмотрим подробно функцию шифрования .{\displaystyle R\_{i}=L\_{i-1}\oplus f(R\_{i-1},k\_{i})}

Для вычисления функции{\displaystyle f} последовательно используются:

1. Функция расширения {\displaystyle \mathrm {E} },
2. Сложение по модулю 2 с ключом{\displaystyle k\_{i}}
3. Преобразование {\displaystyle \mathrm {S} }, состоящее из 8 преобразований {\displaystyle \mathrm {S} }блоков {\displaystyle \mathrm {S} \_{1},\mathrm {S} \_{2},\mathrm {S} \_{3}\ldots \ \mathrm {S} \_{8}},
4. Перестановка{\displaystyle \mathrm {P} } .

Функция  расширяет 32-битовый вектор {\displaystyle R\_{i-1}}RR до 48-битового вектора {\displaystyle \mathrm {E} (R\_{i-1})} путём дублирования некоторых битов из {\displaystyle R\_{i-1}}; порядок битов вектора {\displaystyle \mathrm {E} (R\_{i-1})}указан в таблице 2.

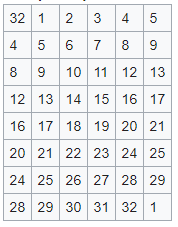


Таблица 2. Функция расширения

Полученный после перестановки блок  {\displaystyle \mathrm {E} (R\_{i-1})}складывается по модулю 2 с ключами  {\displaystyle k\_{i}}л\_и затем представляется в виде восьми последовательных блоков {\displaystyle B\_{1},B\_{2},...B\_{8}}B. Каждый  является 6-битовым блоком. Далее каждый из блоков {\displaystyle B\_{j}}трансформируется в 4-битовый блок {\displaystyle B'\_{j}}BB с помощью преобразований {\displaystyle S\_{j}}S. Преобразования {\displaystyle S\_{j}}S\_J определяются таблицей 3.



Таблица 3 для блока

Таблицы для остальных блоков не приводятся для краткости. Таблица используется следующим образом. Первый и последний биты входного числа образуют число от 0 до 3, средние – число от 0 до 15. Пара чисел определяет число, находящееся в пересечении строки а и столбца b. Значение функции {\displaystyle f(R\_{i-1},k\_{i})} (32 бит) получается перестановкой , применяемой к 32-битовому блоку {\displaystyle B'\_{1}B'\_{2}...B'\_{8}}. Перестановка задана таблицей 4.

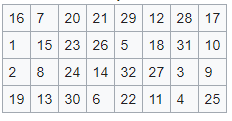


Таблица 4. Перестановка

Ключи {\displaystyle k\_{i}}k\_i получаются из начального ключа {\displaystyle k}k (56 бит) следующим образом. Добавляются биты в позиции 8, 16, 24, 32, 40, 48, 56, 64 ключа {\displaystyle k}k таким образом, чтобы каждый байт содержал нечетное число единиц. Это используется для обнаружения ошибок при обмене и хранении ключей. Затем делают перестановку для расширенного ключа (кроме добавляемых битов 8, 16, 24, 32, 40, 48, 56, 64). Перестановка определена в таблице 5.

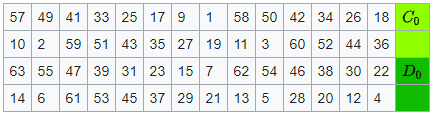


Таблица 5. Перестановка расширенного ключа

Эта перестановка определяется двумя блоками {\displaystyle C\_{0}}Co и {\displaystyle D\_{0}}D\_ по 28 бит каждый.   получаются из {\displaystyle C\_{i-1},D\_{i-1}}C одним или двумя левыми циклическими сдвигами согласно таблице 6.



Таблица 6. Сдвиги ключей и

Ключ  состоит из 48 бит, выбранных из битов вектора {\displaystyle C\_{i}D\_{i}} (56 бит) согласно таблице 7.

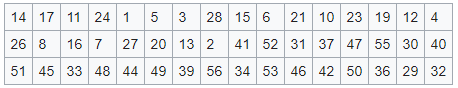


Таблица 7. Выбор битов ключа

### 2.1.3 Конечная перестановка

Конечная перестановка {\displaystyle \mathrm {IP} ^{-1}}IP действует на {\displaystyle T\_{16}}Е\_16 и является обратной к первоначальной перестановке. Определяется таблицей 8.



Таблица 8. Обратная перестановка

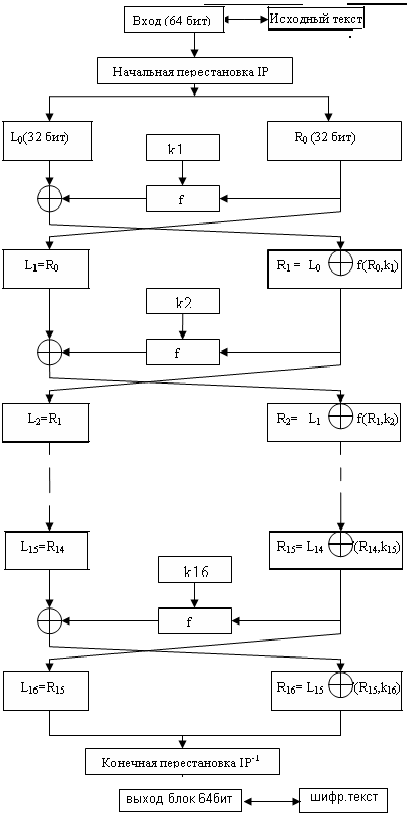


Рисунок 1. Графическая схема шифрования DES

### 2.1.4 Схема расшифровки

При расшифровании данных все действия выполняются в обратном порядке. В 16 циклах расшифрования, в отличие от шифрования c помощью прямого преобразования сетью Фейстеля, здесь используется обратное преобразование сетью Фейстеля.

{\displaystyle L\_{i}=R\_{i-1}}

## **2.2 Алгоритм шифрования ГОСТ 28147**

ГОСТ 28147-89 — блочный шифр с 256-битным ключом и 32 циклами преобразования, оперирующий 64-битными блоками. Основа алгоритма шифра — Сеть Фейстеля.

Базовым режимом шифрования по ГОСТ 28147-89 является режим простой замены (определены также более сложные режимы гаммирование, гаммирование с обратной связью и режим имитовставки). Для зашифрования в этом режиме открытый текст сначала разбивается на две половины (младшие биты — , старшие биты — . На i-ом цикле используется подключ :

{\displaystyle ~A\_{i+1} = B\_i \oplus f(A\_i, K\_i)}

{\displaystyle ~B\_{i+1} = A\_i}

Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных блоков: .

Ключи  являются циклическим повторением ключей (нумеруются от младших битов к старшим). Ключи  являются ключами K1…K8, идущими в обратном порядке.

После выполнения всех 32 раундов алгоритма, блоки  и  склеиваются (старшим битом становится , а младшим — ) — результат есть результат работы алгоритма.

Расшифрование выполняется так же, как и зашифрование, но инвертируется порядок подключей .

Функция{\displaystyle f(A\_i, K\_i)} вычисляется следующим образом:

 и  складываются по модулю .

Результат разбивается на восемь 4-битовых подпоследовательностей, каждая из которых поступает на вход своего *узла таблицы замен* (в порядке возрастания старшинства битов), называемого ниже *S-блоком*. Общее количество S-блоков ГОСТа — восемь, то есть столько же, сколько и подпоследовательностей. Каждый *S-блок* представляет собой перестановку чисел от 0 до 15. Первая 4-битная подпоследовательность попадает на вход первого S-блока, вторая — на вход второго и т. д.

Если *S-блок* выглядит так:

1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12

и на входе S-блока 0, то на выходе будет 1, если 4, то на выходе будет 5, если на входе 12, то на выходе 6 и т. д.

Выходы всех восьми S-блоков объединяются в 32-битное слово, затем всё слово циклически сдвигается влево (к старшим разрядам) на 11 битов.

Все восемь S-блоков могут быть различными. Фактически, они могут являться дополнительным ключевым материалом, но чаще являются параметром схемы, общим для определенной группы пользователей. В ГОСТ Р 34.11-94 для целей тестирования приведены следующие S-блоки:



# **3. Примеры работы программы**

## **3.1 Алгоритм шифрования DES**

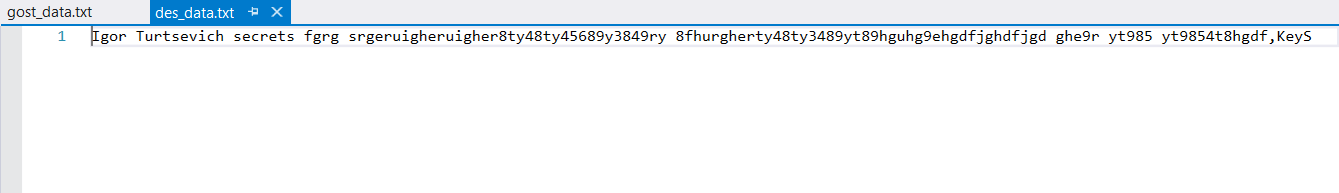


Рисунок 1. Содержимое файла des\_data.txt с данными для шифрования

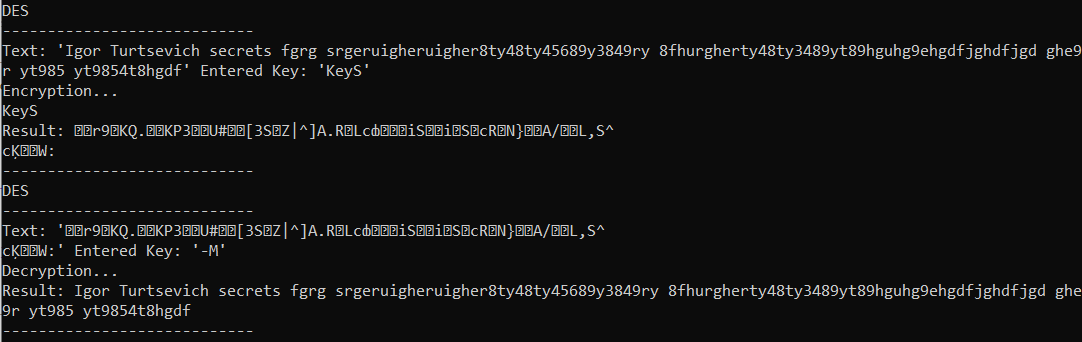


Рисунок 2. Пример вывода программы

## **3.1 Алгоритм шифрования ГОСТ 28147**

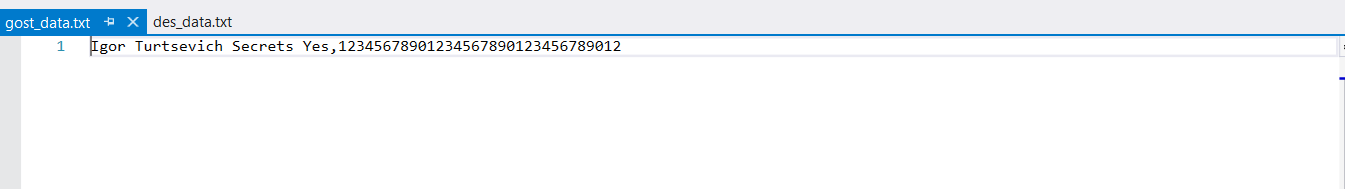


Рисунок 3. Содержимое файла gost\_data.txt с данными для шифрования



Рисунок 4. Пример вывода программы

# **4. Выводы**

В ходе данной работы были изучены методы шифрования DES и ГОСТ 28147. И были созданы программные реализации ГОСТ 28147 и DES на языке программирования C++.

# **Приложение 1. Исходный код реализации алгоритма DES**

**Файл des.cpp**

#include "des.h"

#define DEBUG

pair<string, string> desCipher(string text, string key, DesMode mode)

{

#ifdef DEBUG

cout << "DES" << endl

<< "----------------------------" << endl

<< "Text: '" << text << "' "

<< "Entered Key: '" << key << "'" << endl;

if (mode == DesMode::Decrypt) {

cout << "Decryption..." << endl;

}

else {

cout << "Encryption..." << endl;

}

#endif // DEBUG

vector<string> blocks;

string result = "";

if (DesMode::Encrypt == mode) {

prepareText(text);

createBlocks(blocks, text);

correctKey(key, text.length() / (2 \* blocks.size()));

cout << key << endl;

key = toBinary(key);

for (int i = 0; i < ROUND\_NUM; i++) {

for (int j = 0; j < blocks.size(); j++) {

blocks[j] = encodeRound(blocks[j], key);

}

key = keyForNextRound(key);

}

key = keyForPrevRound(key);

key = toString(key);

for (auto it = blocks.begin(); it != blocks.end(); it++) {

result += \*it;

}

result = toString(result);

}

else {

key = toBinary(key);

text = toBinary(text);

createBinaryBlocks(blocks, text);

for (int i = 0; i < ROUND\_NUM; i++) {

for (int j = 0; j < blocks.size(); j++) {

blocks[j] = decodeRound(blocks[j], key);

}

key = keyForPrevRound(key);

}

key = keyForNextRound(key);

for (auto it = blocks.begin(); it != blocks.end(); it++) {

result += \*it;

}

result = toString(result, true);

}

#ifdef DEBUG

cout << "Result: " << result << endl;

cout << "----------------------------" << endl;

#endif // DEBUG

return pair<string, string>(result, key);

}

void createBlocks(vector<string> &blocks, const string& text)

{

blocks = vector<string>((text.length() \* CHAR\_SIZE) / BLOCK\_SIZE);

int blockLenght = text.length() / blocks.size();

for (int i = 0; i < blocks.size(); i++) {

blocks[i] = text.substr(i \* blockLenght, blockLenght);

blocks[i] = toBinary(blocks[i]);

}

}

void createBinaryBlocks(vector<string>& blocks, const string& text)

{

blocks = vector<string>(text.length() / BLOCK\_SIZE);

int blockLenght = text.length() / blocks.size();

for (int i = 0; i < blocks.size(); i++) {

blocks[i] = text.substr(i \* blockLenght, blockLenght);

}

}

void prepareText(string& text)

{

while (((text.length() \* CHAR\_SIZE) % BLOCK\_SIZE) != 0) {

text += SPEC\_CHR;

}

}

void removeSpecChr(string& text) {

text.erase(remove\_if(text.begin(), text.end(), [](char c) { return c == SPEC\_CHR; }), text.end());

}

string toBinary(const string& text)

{

string output = "";

for (int i = 0; i < text.size(); i++) {

string binaryChar = bitset<CHAR\_SIZE>(text[i]).to\_string();

output += binaryChar;

}

return output;

}

void correctKey(string& key, int keyLength)

{

if (key.length() > keyLength) {

key = key.substr(0, keyLength);

}

else {

while (key.length() < keyLength) {

key = "0" + key;

}

}

}

string encodeRound(const string& text, const string& key)

{

string l = text.substr(0, text.length() / 2);

string r = text.substr(text.length() / 2, text.length() / 2);

return r + xOR(l, chiperFunction(r, key));

}

string decodeRound(const string& text, const string& key)

{

string l = text.substr(0, text.length() / 2);

string r = text.substr(text.length() / 2, text.length() / 2);

return xOR(chiperFunction(l, key), r) + l;

}

string xOR(const string& str1, const string& str2)

{

string result = "";

for (int i = 0; i < str1.length(); i++) {

bool a = (bool)(int)(str1[i] - '0');

bool b = (bool)(int)(str2[i] - '0');

if (a ^ b) {

result += "1";

}

else {

result += "0";

}

}

return result;

}

string chiperFunction(const string& str1, const string& str2)

{

return xOR(str1, str2);

}

string keyForNextRound(string key)

{

for (int i = 0; i < KEY\_SHIFT; i++) {

key = key[key.length() - 1] + key;

key = key.erase(key.length() - 1, 1);

}

return key;

}

string keyForPrevRound(string key)

{

for (int i = 0; i < KEY\_SHIFT; i++) {

key = key + key[0];

key = key.erase(0, 1);

}

return key;

}

string toString(string text, bool removeSpecChr)

{

string output = "";

while (text.length() > 0) {

string binaryChar = text.substr(0, CHAR\_SIZE);

text = text.erase(0, CHAR\_SIZE);

int a = 0;

int degree = binaryChar.length() - 1;

for (auto it = binaryChar.begin(); it != binaryChar.end(); it++) {

a += (int)(\*it - '0') \* (int)pow(2, degree--);

}

char chr = (char)a;

if (SPEC\_CHR == chr && removeSpecChr) {

}

else {

output += chr;

}

}

return output;

}

pair<string, string> desEncrypt(string text, string key) {

return desCipher(text, key);

}

pair<string, string> desDecrypt(string text, string key) {

return desCipher(text, key, DesMode::Decrypt);

}

**Файл des.h**

#pragma once

#include <string>

#include <iostream>

#include <bitset>

#include "common.h"

using namespace std;

constexpr int BLOCK\_SIZE = 64;

constexpr int CHAR\_SIZE = 8;

constexpr int KEY\_SHIFT = 2;

constexpr int ROUND\_NUM = 16;

constexpr int SPEC\_CHR = '~';

enum class DesMode {

Encrypt,

Decrypt

};

void createBlocks(vector<string>& blocks, const string& text);

void createBinaryBlocks(vector<string>& blocks, const string& text);

void prepareText(string& text);

void correctKey(string& text, int keyLength);

string toBinary(const string& text);

string encodeRound(const string& text, const string& key);

string decodeRound(const string& text, const string& key);

string xOR(const string& str1, const string& str2);

string chiperFunction(const string& str1, const string& str2);

string keyForNextRound(string key);

string keyForPrevRound(string key);

string toString(string input, bool removeSpecChr = false);

pair<string, string> desCipher(string text, string key, DesMode mode = DesMode::Encrypt);

pair<string, string> desEncrypt(string text, string key);

pair<string, string> desDecrypt(string text, string key);

**Файл common.cpp**

#include "common.h"

int s\_int(string str)

{

stringstream stream(str);

int res = 0;

stream >> res;

return res;

}

vector<string> split(const string& s, char delimeter)

{

vector<string> substrings;

string token;

istringstream stream(s);

while (getline(stream, token, delimeter)) {

substrings.push\_back(token);

}

return substrings;

}

string readFile(const char\* filename) {

ifstream in(filename);

string text((std::istreambuf\_iterator<char>(in)),

std::istreambuf\_iterator<char>());

return text;

}

**Файл common.h**

#pragma once

#include <fstream>

#include <string>

#include <streambuf>

#include <vector>

#include <sstream>

#include <ostream>

#include <tuple>

#include <random>

using namespace std;

int s\_int(string str);

vector<string> split(const string& s, char delimeter);

string readFile(const char\* filename);

# **Приложение 2. Исходный код реализации алгоритма ГОСТ 28147**

**Файл gost28147.cpp**

#include "gost28147.h"

vector<vector<uint8\_t>> S\_BOX = {

{ 0x0, 0xA, 0x9, 0x2, 0xD, 0x8, 0x0, 0xE, 0x6, 0xB, 0x1, 0xC, 0x7, 0xF, 0x5, 0x3 },

{ 0x0, 0x3, 0x4, 0xC, 0x6, 0xD, 0xF, 0xA, 0x2, 0x3, 0x8, 0x1, 0x0, 0x7, 0x5, 0x9 },

{ 0x0, 0x8, 0x1, 0xD, 0xA, 0x3, 0x4, 0x2, 0xE, 0xF, 0xC, 0x7, 0x6, 0x0, 0x9, 0xB },

{ 0x0, 0xD, 0xA, 0x1, 0x0, 0x8, 0xA, 0xF, 0xE, 0x4, 0x6, 0xC, 0xB, 0x2, 0x5, 0x3 },

{ 0x2, 0x2, 0x7, 0x1, 0x5, 0xF, 0xD, 0xD, 0x4, 0xA, 0x9, 0xE, 0x0, 0x3, 0xB, 0x2 },

{ 0x7, 0xB, 0xA, 0x0, 0x7, 0x2, 0x1, 0xD, 0x3, 0x6, 0x8, 0x5, 0x9, 0xC, 0xF, 0xE },

{ 0xD, 0xB, 0x4, 0x1, 0x3, 0xF, 0xD, 0xD, 0x0, 0xA, 0xE, 0x7, 0x6, 0x8, 0x2, 0xC },

{ 0x9, 0xF, 0xD, 0x0, 0x5, 0x7, 0xA, 0x5, 0x9, 0x2, 0x3, 0xE, 0x6, 0xB, 0x8, 0xC }

};

string gostEncrypt(string text, string key)

{

prepareText(text);

int textLen = text.size();

int keyLen = key.size();

auto textBytes = text.c\_str();

auto keyBytes = key.c\_str();

auto textVector = vector<uint8\_t>(textBytes, textBytes + textLen);

auto keyVector = vector<uint8\_t>(keyBytes, keyBytes + keyLen);

auto res = gost(textVector, keyVector);

return string(res.begin(), res.end());

}

vector<uint8\_t> gost(vector<uint8\_t> data, vector<uint8\_t> key)

{

auto subkeys = generateKeys(key);

auto result = vector<uint8\_t>(data.size(), 0);

auto block = vector<uint8\_t>(8, 0);

for (int i = 0; i < data.size() / 8; i++) {

copy(data.begin() + 8 \* i, data.begin() + 8 \* i + 8, block.begin());

auto eBlock = mutateBlock(block, subkeys);

copy(eBlock.begin(), eBlock.begin() + 8, result.begin() + 8 \* i);

}

return result;

}

vector<uint8\_t> mutateBlock(vector<uint8\_t> block, vector<uint32\_t> keys)

{

uint8\_t\* pBlock = block.data();

uint32\_t n1 = byteToUintCast(pBlock, 0);

uint32\_t n2 = byteToUintCast(pBlock, 4);

for (int i = 0; i < 32; i++) {

int keyIndex = i < 24 ? (i % 8) : (7 - i % 8);

auto s = (n1 + keys[keyIndex]) % UINT32\_MAX;

s = subs(s);

s = (s << 11) | (s >> 21);

s = s ^ n2;

if (i < 31)

{

n2 = n1;

n1 = s;

}

else

{

n2 = s;

}

}

vector<uint8\_t> output(8, 0);

uint8\_t\* n1\_bytes = uintToByteCast(n1);

uint8\_t\* n2\_bytes = uintToByteCast(n2);

for (int i = 0; i < 4; i++) {

output[i] = n1\_bytes[i];

output[4 + i] = n2\_bytes[i];

}

return output;

}

string gostDecrypt(string data, string key)

{

int textLen = data.size();

int keyLen = key.size();

auto textBytes = data.c\_str();

auto keyBytes = key.c\_str();

auto textVector = vector<uint8\_t>(textBytes, textBytes + textLen);

auto keyVector = vector<uint8\_t>(keyBytes, keyBytes + keyLen);

auto res = gost(textVector, keyVector);

auto decrypted = string(res.begin(), res.end());

removeSpecChr(decrypted);

return decrypted;

}

uint32\_t subs(uint8\_t value)

{

uint32\_t output = 0;

for (int i = 0; i < 8; i++) {

uint8\_t temp = (uint8\_t)(value >> (4 \* i) & 0x0F);

temp = S\_BOX[i][temp];

output |= (uint32\_t)temp << (4 \* i);

}

return output;

}

vector<uint32\_t> generateKeys(vector<uint8\_t> key)

{

if (key.size() != 32) {

throw exception("Key length is wrong!");

}

vector<uint32\_t> keys(8, 0);

uint8\_t\* pKey = key.data();

for (int i = 0; i < 8; i++) {

keys[i] = byteToUintCast(pKey, 4 \* i);

}

return keys;

}

uint32\_t byteToUintCast(uint8\_t\* pKey, int pos)

{

return \*reinterpret\_cast<uint32\_t\*>(pKey + pos);

}

uint8\_t\* uintToByteCast(uint32\_t buf)

{

uint8\_t\* bytes = new uint8\_t[4];

for (int i = 0; i < 4; i++)

{

bytes[i] = (uint8\_t)((buf >> (8 \* i)) & 0xFF);

}

return bytes;

}

**Файл gost28147.h**

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

string gostEncrypt(string text, string key);

vector<uint8\_t> gost(vector<uint8\_t> data, vector<uint8\_t> key);

vector<uint8\_t> mutateBlock(vector<uint8\_t> block, vector<uint32\_t> keys);

string gostDecrypt(string data, string key);

void prepareText(string& text);

void removeSpecChr(string& text);

uint32\_t subs(uint8\_t value);

vector<uint32\_t> generateKeys(vector<uint8\_t> key);

uint32\_t byteToUintCast(uint8\_t\* pKey, int pos);

uint8\_t\* uintToByteCast(uint32\_t uint);