Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №1

**Симметричная криптография.**

**Двойной и тройной DES.**

Выполнил:

Куликов А.Д.

Проверил:

Артемьев В.С.

Минск 2019

Оглавление

[Введение 3](#_Toc22124971)

[Блок-схема алгоритма 4](#_Toc22124972)

[Результаты выполнения программы 5](#_Toc22124973)

[Вывод 6](#_Toc22124974)

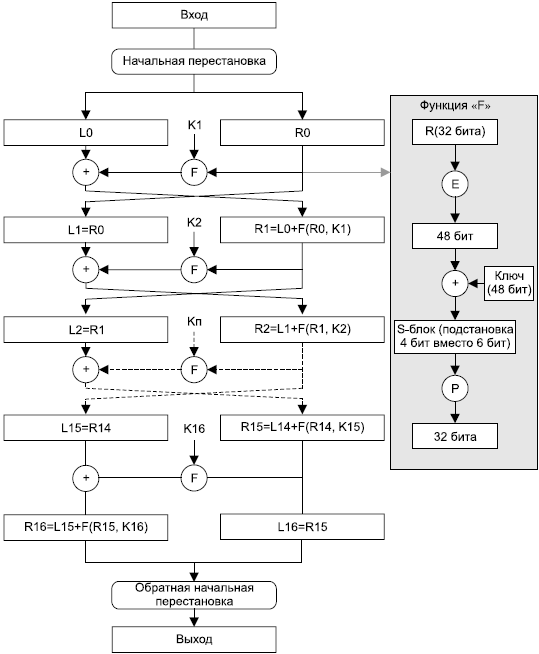
[Исходный код программы 7](#_Toc22124975)

# Введение

*Цель:* Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритмов двойной и тройной DES.

*Постановка задачи:* Изучить теоретические сведения. Используя полученные знания создать программу на языке C++, выполняющую чтение исходных данных из файла и шифрующую (дешифрующую) их с помощью с помощью алгоритмов двойной и тройной DES*.*

# Блок-схема алгоритма



# Результаты выполнения программы



Рисунок Исходные данные



Рисунок Алгоритм DES



Рисунок 3 Двойной DES



Рисунок Тройной DES

# Вывод

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Расшифрование в DES является операцией обратной шифрованию и выполняется путем повторения операций шифрования в обратной последовательности

Чтобы увеличивать криптостойкость DES, появляются несколько вариантов double DES (2DES), triple DES (3DES).

Методы 2DES и 3DES основаны на DES, но увеличивают длину ключей (2DES — 112 бит, 3DES — 168 бит) и поэтому увеличивается криптостойкость.

# Исходный код программы

des.cpp

#include "des.h"

#include "utils.h"

#include "constants.h"

vector<bool> des::extend\_key(const vector<bool>& key\_bits)

{

const int len = key\_bits.size();

vector<bool> key\_ext;

for (auto i = 0, cur = 0; i < len; i++)

{

cur ^= key\_bits[i];

key\_ext.push\_back(key\_bits[i]);

if ((i + 1) % 7 == 0)

{

key\_ext.push\_back(!cur);

cur = 0;

}

}

return key\_ext;

}

vector<vector<bool>> des::generate\_keys(const string& key) const

{

const auto size = 56;

const vector<char> bytes(key.begin(), key.end());

auto bits = get\_bits(bytes);

const vector<bool> key\_bits(bits.begin(), bits.begin() + 64);

//auto key\_ext = extend\_key(key\_bits);

auto key\_ext = permutate(key\_bits, kp1);

vector<bool> c(key\_ext.begin(), key\_ext.begin() + size / 2);

vector<bool> d(key\_ext.begin() + size / 2, key\_ext.end());

vector<vector<bool>> keys;

for (auto i = 0; i < 16; i++)

{

rotate(c.begin(), c.begin() + rt[i], c.end());

rotate(d.begin(), d.begin() + rt[i], d.end());

vector<bool> cur\_key;

cur\_key.insert(cur\_key.end(), c.begin(), c.end());

cur\_key.insert(cur\_key.end(), d.begin(), d.end());

cur\_key = permutate(cur\_key, kp2);

keys.push\_back(cur\_key);

}

return keys;

}

vector<bool> des::feistel(const vector<bool>& r, const vector<bool>& key) const

{

vector<bool> res;

auto r\_new = permutate(r, et);

r\_new = xor\_v(r\_new, key);

const auto len = r\_new.size();

res.reserve(r.size());

for (unsigned int i = 0, j = 0; i < len; i += 6, j++)

{

vector<bool> block(r\_new.begin() + i, r\_new.begin() + (i + 6));

const auto y = (block[0] << 1) + block[5];

const auto x = (block[1] << 3) + (block[2] << 2) + (block[3] << 1) + block[4];

auto s\_box\_val\_bits = get\_bits(s\_box[j][y][x], 4);

res.insert(res.end(), s\_box\_val\_bits.begin(), s\_box\_val\_bits.end());

}

res = permutate(res, p);

return res;

}

vector<bool> des::encrypt\_block(const vector<bool>& block, const vector<vector<bool>>& keys, const bool& decrypt) const

{

vector<bool> res;

auto src = permutate(block, ip);

const auto size = src.size();

vector<bool> l(src.begin(), src.begin() + size / 2);

vector<bool> r(src.begin() + size / 2, src.end());

auto start = 0, end = 16, inc = 1;

if (decrypt) start = 15, end = -1, inc = -1;

while (start != end)

{

auto new\_l = r;

auto new\_r = feistel(r, keys[start]);

new\_r = xor\_v(new\_r, l);

r = vector<bool>(new\_r);

l = vector<bool>(new\_l);

start += inc;

}

res.insert(res.end(), r.begin(), r.end());

res.insert(res.end(), l.begin(), l.end());

res = permutate(res, fp);

return res;

}

des::des()

{

keys\_ = generate\_keys(default\_key\_);

}

string des::encrypt(const string& message, const bool& decrypt) const

{

const vector<char> bytes(message.begin(), message.end());

const auto bits = get\_bits(bytes);

auto blocks = get\_blocks(bits);

vector<vector<bool>> encrypted\_blocks;

encrypted\_blocks.reserve(blocks.size());

for (const auto& block : blocks) encrypted\_blocks.push\_back(encrypt\_block(block, keys\_, decrypt));

return restore\_message(encrypted\_blocks);

}

des::des(const string& base\_key)

{

keys\_ = generate\_keys(base\_key);

}

string des::encrypt(const string& message) const

{

return encrypt(message, false);

}

string des::decrypt(const string& message) const

{

return encrypt(message, true);

}

utils.cpp

#include <algorithm>

#include <vector>

#include <string>

#include "utils.h"

#include <iostream>

using namespace std;

int to\_int(const vector<bool>& bits)

{

unsigned int res = 0;

const unsigned int len = bits.size();

for (unsigned int i = 0; i < len; i++) res += bits[i] << (len - i - 1);

return res;

}

vector<bool> permutate(const vector<bool>& block, const vector<int>& table)

{

const auto len = table.size();

vector<bool> res;

res.reserve(len);

for (unsigned int i = 0; i < len; i++) res.push\_back(block[table[i]]);

return res;

}

vector<bool> xor\_v(vector<bool> a, vector<bool> b)

{

const auto len = a.size();

vector<bool> res;

for (unsigned int i = 0; i < len; i++) res.push\_back(a[i] ^ b[i]);

return res;

}

vector<bool> get\_bits(const vector<char>& text)

{

vector<bool> res;

for (auto c : text)

{

vector<bool> tmp;

tmp.reserve(8);

for (auto i = 0; i < 8; i++)

tmp.push\_back((c >> i) & 1);

reverse(tmp.begin(), tmp.end());

res.insert(res.end(), tmp.begin(), tmp.end());

}

return res;

}

vector<bool> get\_bits(const int val, const int size)

{

vector<bool> res;

res.reserve(size);

for (auto i = 0; i < size; i++) res.push\_back((val >> i) & 1);

reverse(res.begin(), res.end());

return res;

}

vector<vector<bool>> get\_blocks(vector<bool> bits)

{

vector<vector<bool>> res;

vector<bool> batch;

const int n = bits.size();

for (auto i = 0; i < n; i++)

{

batch.push\_back(bits[i]);

if ((i + 1) % 64 == 0)

{

res.push\_back(batch);

batch.clear();

}

}

if (!batch.empty())

{

while (batch.size() < 64) batch.push\_back(false);

res.push\_back(batch);

}

return res;

}

char restore\_byte(const vector<bool>& bits)

{

auto c = 0;

const auto len = bits.size();

for (unsigned int i = 0; i < len; i++) c += bits[i] << (len - i - 1);

return static\_cast<char>(c);

}

bool is\_zero\_vector(const vector<bool>& v)

{

for (auto x : v) if (x) return false;

return true;

}

string restore\_block(const vector<bool>& bits)

{

string res;

const auto len = bits.size();

for (unsigned int i = 0; i < len; i += 8)

{

vector<bool> byte(bits.begin() + i, bits.begin() + (i + 8));

if (is\_zero\_vector(byte)) continue;

res += restore\_byte(byte);

}

return res;

}

string restore\_message(const vector<vector<bool>>& blocks)

{

string res;

for (const auto& block : blocks) res += restore\_block(block);

return res;

}

void print(const vector<bool>& block)

{

for (auto c : block) cout << c;

cout << endl;

}

void print(const vector<vector<bool>>& message)

{

for (const auto& block : message) print(block);

}