Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ

к лабораторной работе №2

на тему

**Идентификация и аутентификация пользователей. Протокол Kerberos.**

Студент Т. П. Власенко

Преподаватель Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_6127s8b6t8wy)

[2 Блок-схема](#_r4y5108cx8yu) алгоритма [4](#_r4y5108cx8yu)

[3 Результат выполнения лабораторной работы](#_jnfz1qbvv5j4) 7

[Выводы](#_ldsbh3e5c0xh) 8

[Приложение А (обязательное) Листинг программного кода](#_54xlhz7mfhe) 9

# 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является разработка программного средство реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

# 2 БЛОК-СХЕМА АЛГОРИТМА

На рисунках 1, 2 продемонстрированы блок-схемы для шифра DES.

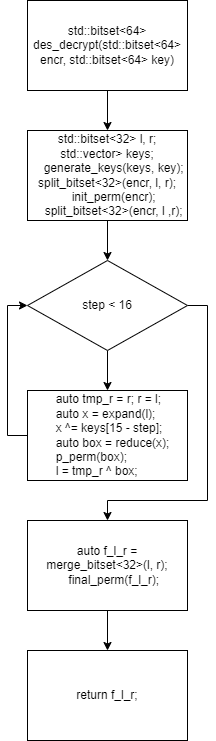


Рисунок 1 – Блок-схема функции *decrypt*

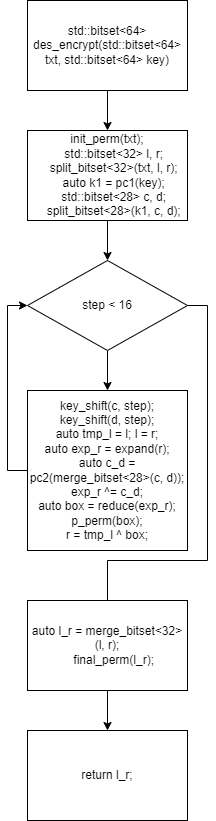


Рисунок 2 – Блок-схема функции *encrypt*

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

На рисунках 3, 4, 5 продемонстрированы примеры работы сервера *KDC*, сервера *SS* и клиента *C* на всех шагах рабочего этапа протокола.

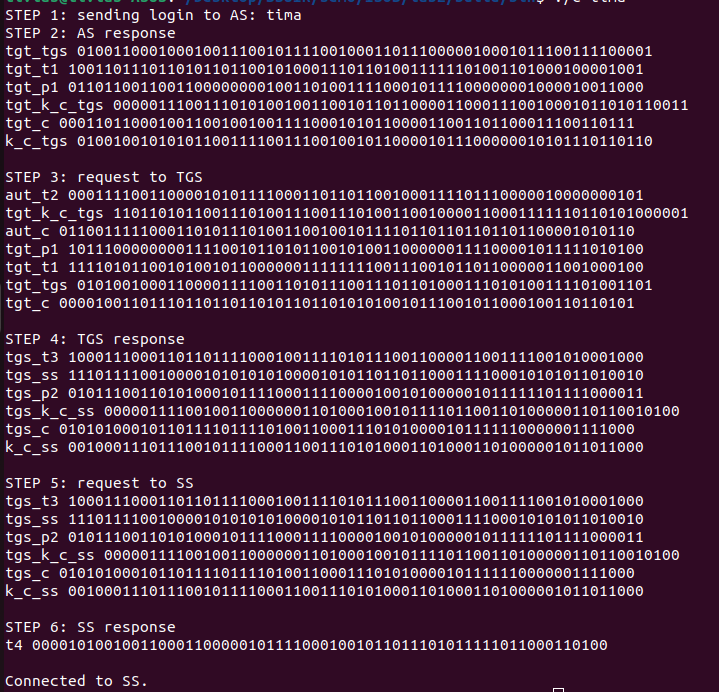


Рисунок 3 – Пример работы клиента *C*

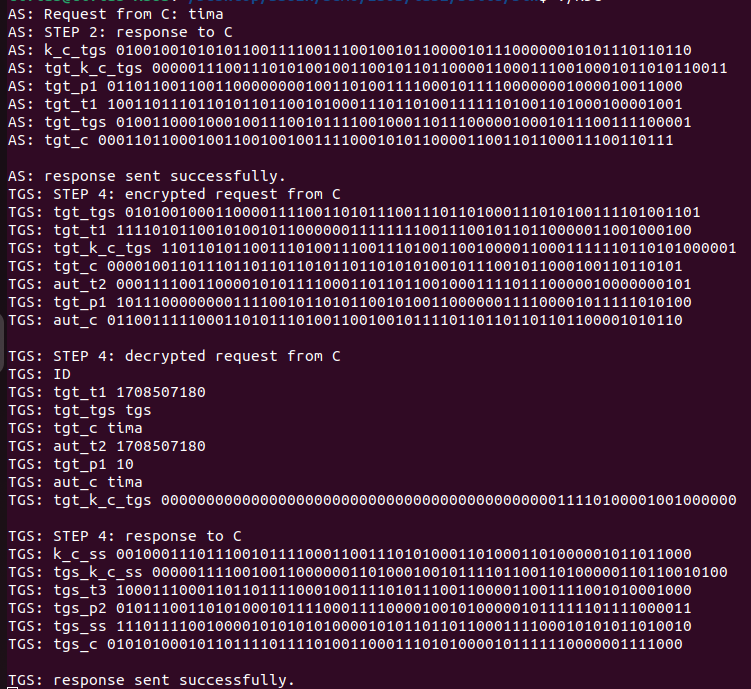


Рисунок 4 – Пример работы сервера *KDC*

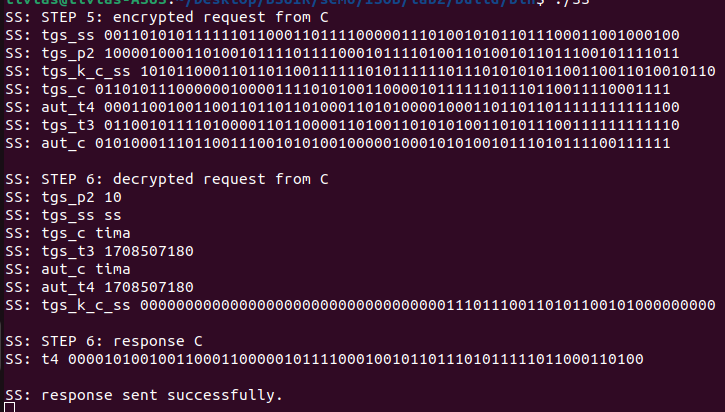


Рисунок 5 – Пример работы сервера *SS*

# ВЫВОДЫ

Для лабораторной работы по теме « Идентификация и аутентификация пользователей. Протокол *Kerberos*.» был написан код на языке *C*++ с использованием библиотеки *Boost*.*Asio*. Для шифра *DES* были реализованы функции *encrypt* и *decrypt*, которые принимают значение и ключ, по которому будет происходить шифрование, и возвращают новое значение. Так же были реализованы 2 сервера (*KDC*, состоящий из *AS* и *TGS*, и *SS*) с асинхронными операциями чтения и записи, которые служат для аутентификации пользователя и выдачи ему разрешений.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг программного кода

#pragma once

#include <bitset>

#include <stdexcept>

#include <vector>

namespace des {

std::bitset<32> S\_boxes(const std::bitset<48> &s) {

constexpr int boxes[8][4][16] = {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

std::bitset<32> result;

constexpr std::bitset<48> six\_set((2 << 6) - 1);

constexpr std::bitset<6> four\_set((2 << 4) - 1);

for (size\_t i = 0; i < 8; i++) {

std::bitset<6> box\_in(((s >> 6 \* i) & six\_set).to\_ulong());

std::bitset<4> column(((box\_in >> 1) & four\_set).to\_ulong());

std::bitset<4> box\_out(

boxes[i][box\_in[0] \* 2 + box\_in[5]][column.to\_ulong()]);

for (size\_t j = i \* 4; j < (i + 1) \* 4; j++) {

result[j] = box\_out[j - i \* 4];

}

}

return result;

}

void P(std::bitset<32> &s) {

constexpr int p[32] = {16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,

26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,

3, 9, 19, 13, 30, 6, 22, 11, 4, 25};

std::bitset<32> temp;

for (size\_t i = 0; i < 32; i++) {

temp[i] = s[p[i] - 1];

}

s = temp;

}

void initial\_permutation(std::bitset<64> &bs) {

constexpr int p[64] = {58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28,

20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56,

48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9,

1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37,

29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7};

std::bitset<64> temp;

for (size\_t i = 0; i < 64; i++) {

temp[i] = bs[p[i] - 1];

}

bs = temp;

}

void final\_permutation(std::bitset<64> &bs) {

constexpr int p[64] = {40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55,

23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5,

45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60,

28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10,

50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25};

std::bitset<64> temp;

for (size\_t i = 0; i < 64; i++) {

temp[i] = bs[p[i] - 1];

}

bs = temp;

}

template <size\_t N>

void split\_in\_half(const std::bitset<N \* 2> &full, std::bitset<N> &left,

std::bitset<N> &right) {

left = std::bitset<N>((full >> N).to\_ullong());

right = std::bitset<N>(

(full & static\_cast<std::bitset<2 \* N>>((2ULL << (N - 1)) - 1))

.to\_ullong());

}

template <size\_t N>

std::bitset<2 \* N> merge\_halves(const std::bitset<N> &left,

const std::bitset<N> &right) {

std::bitset<2 \* N> result;

for (size\_t i = 0; i < N; i++) {

result[i] = right[i];

}

for (size\_t i = 0; i < N; i++) {

result[N + i] = left[i];

}

return result;

}

std::bitset<56> PC\_1(const std::bitset<64> &s) {

constexpr int p[56] = {57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34,

26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3,

60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7,

62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37,

29, 21, 13, 5, 28, 20, 12, 4};

std::bitset<56> result;

for (size\_t i = 0; i < 56; i++) {

result[i] = s[p[i] - 1];

}

return result;

}

std::bitset<48> PC\_2(const std::bitset<56> &s) {

constexpr int p[48] = {14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32};

std::bitset<48> result;

for (size\_t i = 0; i < 48; i++) {

result[i] = s[p[i] - 1];

}

return result;

}

void cyclic\_shift(std::bitset<28> &key, int step) {

constexpr int step\_shifts[16] = {1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1};

int shift = step\_shifts[step];

if (shift == 1) {

auto k27 = key[27];

key <<= shift;

key[0] = k27;

} else {

auto k26 = key[26], k27 = key[27];

key <<= shift;

key[0] = k26;

key[1] = k27;

}

}

std::bitset<48> expand(const std::bitset<32> &Ri) {

std::bitset<48> result;

result[0] = Ri[31];

result[47] = Ri[0];

for (size\_t i = 0; i < 8; i++) {

if (i != 0) result[i \* 6] = Ri[i \* 4 - 1];

if (i != 7) result[i \* 6 + 5] = Ri[(i + 1) \* 4];

for (size\_t j = 1; j < 5; j++) {

result[i \* 6 + j] = Ri[i \* 4 + j - 1];

}

}

return result;

}

std::bitset<64> encrypt(std::bitset<64> data, std::bitset<64> key) {

initial\_permutation(data);

std::bitset<32> l, r;

split\_in\_half<32>(data, l, r);

auto K = PC\_1(key);

std::bitset<28> c, d;

split\_in\_half<28>(K, c, d);

for (int step = 0; step < 16; step++) {

cyclic\_shift(c, step);

cyclic\_shift(d, step);

auto prev\_l = l;

l = r;

auto er = expand(r);

auto cd = PC\_2(merge\_halves<28>(c, d));

er ^= cd;

auto box = S\_boxes(er);

P(box);

r = prev\_l ^ box;

}

auto lr = merge\_halves<32>(l, r);

final\_permutation(lr);

return lr;

}

void get\_keys(std::vector<std::bitset<48>> &keys, const std::bitset<64> &key) {

auto K = PC\_1(key);

std::bitset<28> c, d;

split\_in\_half<28>(K, c, d);

for (int step = 0; step < 16; step++) {

cyclic\_shift(c, step);

cyclic\_shift(d, step);

auto c\_d = PC\_2(merge\_halves<28>(c, d));

keys.push\_back(c\_d);

}

}

std::bitset<64> decrypt(std::bitset<64> data, std::bitset<64> key) {

std::bitset<32> l, r;

std::vector<std::bitset<48>> keys;

keys.reserve(16);

get\_keys(keys, key);

split\_in\_half<32>(data, l, r);

initial\_permutation(data);

split\_in\_half<32>(data, l, r);

for (int step = 0; step < 16; step++) {

auto prev\_r = r;

r = l;

auto el = expand(l);

el ^= keys[15 - step];

auto box = S\_boxes(el);

P(box);

l = prev\_r ^ box;

}

auto lr = merge\_halves<32>(l, r);

final\_permutation(lr);

return lr;

}

std::bitset<64> str\_to\_bits(const std::string &s) {

std::bitset<64> bits;

std::bitset<8> temp;

for (size\_t i = 0; i < s.size(); i++) {

temp = std::bitset<8>(s[i]);

for (size\_t j = 0; j < 8; j++) {

bits[j + 8 \* i] = temp[j];

}

}

return bits;

}

std::string bits\_to\_str(const std::bitset<64> &bits) {

std::string s;

std::bitset<8> temp;

for (size\_t i = 0; i < 8; i++) {

for (size\_t j = 0; j < 8; j++) {

temp[j] = bits[j + i \* 8];

}

if (temp.to\_ulong() == 0) break;

s += static\_cast<char>(temp.to\_ulong());

}

return s;

}

} // namespace des