МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Протокол Kerberos

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Алексеева Александра Александровича

Преподаватель
профессор, д.ф.-м.н. _____ В. Е. Новиков
подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

1 Теоретическая часть

Цель работы – изучение протокола Kerberos.

1.1 Описание алгоритма

Kerberos – вариант протокола Needham-Schroeder. В базовом протоколе Kerberos Version 5 у Алисы и Боба общие ключи с Трентом. Алиса хочет генерировать сеансовый ключ для сеанса связи с Бобом.

(1) Алиса посылает Тренту сообщение со своим именем и именем Боба:

$$A, B$$

(2) Трент создаёт сообщение, состоящее из метки времени, время жизни, случайного сеансового ключа и имени Алисы. Он шифрует сообщение ключом, общим для него и Боба. Затем он объединяет метку времени, время жизни, сеансовый ключ, имя Боба, и шифрует полученное сообщение ключом, общим для него и Алисы. Оба шифрованных сообщения он отправляет Алисе.

$$E_A(T, L, K, B), E_B(T, L, K, A)$$

(3) Алиса создаёт сообщение, состоящее из её имени и метки времени, шифрует его ключом К и отправляет Бобу. Алиса также посылает Бобу сообщение от Трента, шифрованное ключом Боба:

$$E_A(A, T), E_B(T, L, K, A)$$

(4) Боб создаёт сообщение, состоящее из метки времени плюс единица, шифрует его ключом K и отправляет Алисе:

$$E_K(T + 1)$$

Этот протокол работает, но только если часы каждого пользователя синхронизированы с часами Трента. На практике эффект достигается синхронизацией с надежным сервером времени с точностью в несколько минут и обнаружением повторной передачи в течение определённого интервала времени.

2 Описание программы

Программа, представленная ниже, содержит следующие функции:

• powClosed (*x*, *y*, *mod*) – возводит число *x* в степень *y* по модулю *mod*;

• miller_rabin (*n*, *k* = 10) – проверка числа *n* на простоту с вероятностью $\frac{1}{2^k}$;

• generateKeys(*x*) – генерация чисел *p*, *g* и *y*, которые используется в шифрсистеме Эль-Гамаля;

• encryption(*keysPGY*, *x*, *message*) – шифрование сообщения *message* с помощью ключей *p*, *g*, *x*, *y*;

• decryption(*keysPGY*, *x*, *cipherText*) – расшифрование шифртекста *cipherText* с помощью ключей *p*, *g*, *x*, *y*;

• kerberos(*users*) – реализация протокола Kerberos для пользователей *users*.

2.1 Примеры работы программы

```
        Протокол Kerberos
Введите логин и пароль пользователя 1: Alice passwd1
Введите логин и пароль пользователя 2: Bob   passwd2

Пользователь Alice зерегистрирован в системе Kerberos. Общий секретный ключ: 11286431034786165566
Пользователь Bob зарегистрирован в системе Kerberos. Общий секретный ключ: 11286429935274537355

Пользователь Alice отправляет Kerberos сообщение (Alice, Bob)

Kerberos создаёт сообщение (T, L, K, B) = (1697764529, 300, 1310378251, Bob) для пользователя Alice, генерирует открытые ключи (p, g, y) = (43118644324957, 306710170089
20, 42043022294815) и зашифровывает сообщение общим секретным ключом: (30671017008920:38662152833989) (35497622217332:17703173583086) (30671017008920:11110416642187) (4
1721858543332:8299778933108) (24525305006381:32751292310114) (28577961388746:9887928495749) (30671017008920:13994599599570) (20102141400682:12029892394766)

Kerberos создаёт сообщение (T, L, K, A) = (1697764529, 300, 1310378251, Alice) для пользователя Bob, генерирует открытые ключи (p, g, y) = (999389, 58995, 230962) и заш
ифровывает сообщение общим секретным ключом: (58995:506185) (935420:387623) (524865:314500) (58995:651345) (924388:821162) (924388:849438) (912660:7510) (792395:609779)
 (58995:230202) (58995:904067) (872431:459436) (120309:696914) (58995:395703) (792395:254787) (792395:340371) (120927:723853) (924388:304583) (935420:858097)

Зашифрованные сообщения и сгенерированные ключи отправляются пользователю Alice

Пользователь Alice получает сообщения. Сообщение для пользователя Alice: (1697764529, 300, 1310378251, Bob). Далее пользователя формирует собщение (A, T) = (Alice, 1697
764529) и зашифровывает сеансовым ключом : (472356537733136641:1379804745964323404) (472356537733136641:720742761351034137) (1111640556427965279:869096527023030554) (40
2607313388931318:1191406286304558486)
Сообщения (A, T) и (T, L, K, A) передаёт пользователю Bob

Пользователь Bob получает свои сообщение и расшифровывает: (A, T) = (Alice, 1697764529) и (T, L, K, A) = (1697764529, 300, 1310378251, Alice)

Пользователь Bob Создаёт сообщение (T + 1) = 1697764530, зашифровывает его сеансовым ключом и отправляет пользователю Alice: (1458086313259978459:622894517075784773) (9
35721804433532936:1222521543762319093)

Пользователь Alice получает сообщение от Bob и расшифровывает его: 1697764530
```

3 Листинг кода

```
#include <iostream>
#include "cmath"
#include "vector"
#include "string"
#include "chrono"
#include "map"
#include "fstream"
#include "boost/multiprecision/cpp_int.hpp"

using namespace std;
using namespace std::chrono;
using namespace boost::multiprecision;

map <char, string> book{ {'0', "111"}, {'1', "112"}, {'2', "113"}, {'3', "114"},
{'4', "115"}, {'5', "116"}, {'6', "117"}, {'7', "118"}, {'8', "119"},
                         {'9', "121"}, {' ', "122"}, {'!', "123"}, {'"', "124"},
{'#', "125"}, {'$', "126"}, {'%', "127"}, {'^', "128"}, {'&', "129"},
                         {'\'', "131"}, {'(', "132"}, {')', "133"}, {'*',
"134"}, {'+', "135"}, {',', "136"}, {'-', "137"}, {'.', "138"}, {'/', "139"},
                         {':', "141"}, {';', "142"}, {'<', "143"}, {'=', "144"},
{'>', "145"}, {'?', "146"}, {'@', "147"}, {'[', "148"}, {'\\', "149"},
                         {']', "151"}, {'_', "152"}, {'`', "153"}, {'{', "154"},
{'}', "155"}, {'|', "156"}, {'~', "157"}, {'\n', "158"}, {'a', "159"},
                         {'b', "161"}, {'c', "162"}, {'d', "163"}, {'e', "164"},
{'f', "165"}, {'g', "166"}, {'h', "167"}, {'i', "168"}, {'j', "169"},
                         {'k', "171"}, {'l', "172"}, {'m', "173"}, {'n', "174"},
{'o', "175"}, {'p', "176"}, {'q', "177"}, {'r', "178"}, {'s', "179"},
                         {'t', "181"}, {'u', "182"}, {'v', "183"}, {'w', "184"},
{'x', "185"}, {'y', "186"}, {'z', "187"}, {'A', "188"}, {'B', "189"},
                         {'C', "191"}, {'D', "192"}, {'E', "193"}, {'F', "194"},
{'G', "195"}, {'H', "196"}, {'I', "197"}, {'J', "198"}, {'K', "199"},
                         {'L', "211"}, {'M', "212"}, {'N', "213"}, {'O', "214"},
{'P', "215"}, {'Q', "216"}, {'R', "217"}, {'S', "218"}, {'T', "219"},
                         {'U', "221"}, {'V', "222"}, {'W', "223"}, {'X', "224"},
{'Y', "225"}, {'Z', "226"} };

map <string, char> bookRvs{ {"111", '0'}, {"112", '1'}, {"113", '2'}, {"114",
'3'}, {"115", '4'}, {"116", '5'}, {"117", '6'}, {"118", '7'}, {"119", '8'},
                            {"121", '9'}, {"122", ' '}, {"123", '!'}, {"124",
'"'}, {"125", '#'}, {"126", '$'}, {"127", '%'}, {"128", '^'}, {"129", '&'},
                            {"131", '\''}, {"132", '('}, {"133", ')'}, {"134",
'*'}, {"135", '+'}, {"136", ','}, {"137", '-'}, {"138", '.'}, {"139", '/'},
                            {"141", ':'}, {"142", ';'}, {"143", '<'}, {"144",
'='}, {"145", '>'}, {"146", '?'}, {"147", '@'}, {"148", '['}, {"149", '\\'},
                            {"151", ']'}, {"152", '_'}, {"153", '`'}, {"154",
'{'}, {"155", '}'}, {"156", '|'}, {"157", '~'}, {"158", '\n'}, {"159", 'a'},
                            {"161", 'b'}, {"162", 'c'}, {"163", 'd'}, {"164",
'e'}, {"165", 'f'}, {"166", 'g'}, {"167", 'h'}, {"168", 'i'}, {"169", 'j'},
                            {"171", 'k'}, {"172", 'l'}, {"173", 'm'}, {"174",
'n'}, {"175", 'o'}, {"176", 'p'}, {"177", 'q'}, {"178", 'r'}, {"179", 's'},
                            {"181", 't'}, {"182", 'u'}, {"183", 'v'}, {"184",
'w'}, {"185", 'x'}, {"186", 'y'}, {"187", 'z'}, {"188", 'A'}, {"189", 'B'},
                            {"191", 'C'}, {"192", 'D'}, {"193", 'E'}, {"194",
'F'}, {"195", 'G'}, {"196", 'H'}, {"197", 'I'}, {"198", 'J'}, {"199", 'K'},
                            {"211", 'L'}, {"212", 'M'}, {"213", 'N'}, {"214",
'O'}, {"215", 'P'}, {"216", 'Q'}, {"217", 'R'}, {"218", 'S'}, {"219", 'T'},
                            {"221", 'U'}, {"222", 'V'}, {"223", 'W'}, {"224",
'X'}, {"225", 'Y'}, {"226", 'Z'} };


vector <cpp_int> deg2(cpp_int el, cpp_int n) {
```

```
    vector <cpp_int> res;
    while (n != 0) {
        if (n / el == 1) {
            res.push_back(el);
            n -= el;
            el = 1;
        }
        else
            el *= 2;
    }
    return res;
}


cpp_int multMod(cpp_int n, cpp_int mod, vector <pair <cpp_int, cpp_int>> lst) {
    if (lst.size() == 1) {
        cpp_int res = 1;
        for (int i = 0; i < lst[0].second; i++)
            res = res * lst[0].first % mod;
        return res;
    }
    else if (lst[0].second == 1) {
        cpp_int el = lst[0].first;
        lst.erase(lst.begin());
        return (el * multMod(n, mod, lst)) % mod;
    }
    else {
        for (int i = 0; i < lst.size(); i++)
            if (lst[i].second > 1) {
                lst[i].first = (lst[i].first * lst[i].first) % mod;
                lst[i].second /= 2;
            }
        return multMod(n, mod, lst);
    }
}


cpp_int powClosed(cpp_int x, cpp_int y, cpp_int mod) {
    if (y == 0)
        return 1;

    vector <cpp_int> lst = deg2(1, y);
    vector <pair <cpp_int, cpp_int>> xDegs;
    for (int i = 0; i < lst.size(); i++)
        xDegs.push_back(make_pair(x, lst[i]));

    cpp_int res = multMod(x, mod, xDegs);
    return res;
}


cpp_int usualEuclid(cpp_int a, cpp_int b) {
    if (a < b)
        swap(a, b);
    if (a < 0 || b < 0)
        throw string{ "Выполнение невозможно: a < 0 или b < 0" };
    else if (b == 0)
        return a;

    cpp_int r = a % b;
    return usualEuclid(b, r);
}
```

```
cpp_int decForm(string x) {
    cpp_int res = 0, deg = 1;
    if (x.back() == '1')
        res += 1;
    for (int i = 1; i < x.length(); i++) {
        deg = deg * 2;
        if (x[x.length() - i - 1] == '1')
            res += deg;
    }
    return res;
}


bool miller_rabin(cpp_int n, int k = 10) {
    if (n == 0 || n == 1)
        return false;

    cpp_int d = n - 1;
    cpp_int s = 0;
    while (d % 2 == 0) {
        s++;
        d = d / 2;
    }

    cpp_int nDec = n - 1;
    for (int i = 0; i < k; i++) {
        cpp_int a = rand() % nDec;
        if (a == 0 || a == 1)
            a = a + 2;

        cpp_int x = powClosed(a, d, n);
        if (x == 1 || x == nDec)
            continue;

        bool flag = false;
        for (int j = 0; j < s; j++) {
            x = (x * x) % n;
            if (x == nDec) {
                flag = true;
                break;
            }
        }
        if (!flag)
            return false;
    }

    return true;
}


vector <cpp_int> generateKeys(cpp_int x) {
    cpp_int q = rand();
    while (!miller_rabin(q))
        q++;

    cpp_int s, p = 2, pDec;
    while (!miller_rabin(p)) {
        string sBin = "";
        int sBinSize = rand() % 50 + 1;
        for (int i = 0; i < sBinSize; i++)
            sBin = sBin + to_string(rand() % 2);
        s = decForm(sBin);
```

```
        p = (q * s) + 1;
        pDec = p - 1;
    }

    cpp_int a = 2, g;
    while (pDec > a) {
        g = powClosed(a, pDec / q, p);
        if (g == 1) {
            a++;
            continue;
        }
        break;
    }

    cpp_int y = powClosed(g, x % p, p);
    return vector <cpp_int> {p, g, y};
}


////////////////////////////////////////////////////////////////////////РЕЖИМ
ШИФРОВАНИЯ////////////////////////////////////////////////////////////////////
/
vector <pair <cpp_int, cpp_int>> encryption(vector <cpp_int> keysPGY, cpp_int x,
string message) {
    vector <pair <cpp_int, cpp_int>> res;
    cpp_int p = keysPGY[0], g = keysPGY[1], y = keysPGY[2];

    string codeSymbs = "";
    for (int i = 0; i < message.length(); i++)
        codeSymbs += book[message[i]];

    int offset = to_string(p).size();
    for (int i = 0; i < codeSymbs.length(); i += offset) {
        cpp_int M(codeSymbs.substr(i, offset));
        if (M / p != 0) {
            string help = to_string(M);
            help.pop_back();
            cpp_int m(help);
            M = m;
            i--;
        }

        cpp_int k = 2;
        while (usualEuclid(k, p - 1) != 1) {
            string kBin = "";
            int kBinSize = rand() % offset;
            for (int i = 0; i < kBinSize; i++)
                kBin = kBin + to_string(rand() % 2);
            k = decForm(kBin + "1") % p;
        }

        res.push_back(make_pair(powClosed(g, k, p), powClosed(y, k, p) * M %
p));
    }
    return res;
}


////////////////////////////////////////////////////////////////////////РЕЖИМ
РАСШИФРОВАНИЯ////////////////////////////////////////////////////////////////
///
```

```
string decryption(vector <cpp_int> keysPGY, cpp_int x, vector <pair <cpp_int,
cpp_int>> ciphertext) {
    string res = "";
    cpp_int p = keysPGY[0], g = keysPGY[1], y = keysPGY[2];

    string codeSymbs = "";
    for (int i = 0; i < ciphertext.size(); i++) {
        cpp_int M, a = ciphertext[i].first, b = ciphertext[i].second;
        M = powClosed(a, p - 1 - (x % p), p) * b % p;
        codeSymbs += to_string(M);
    }

    for (int i = 0; i < codeSymbs.length(); i += 3) {
        string M = codeSymbs.substr(i, 3);
        res += bookRvs[M];
    }
    return res;
}


void kerberos(pair <string, string>* users) {
    hash <string> hashStr;
    cpp_int keyUser1(hashStr(users[0].second)),
keyUser2(hashStr(users[1].second));
    cout << "\nПользователь " << users[0].first << " зерегистрирован в системе
Kerberos. Общий секретный ключ: " << keyUser1;
    cout << "\nПользователь " << users[1].first << " зарегистрирован в системе
Kerberos. Общий секретный ключ: " << keyUser2;

    cout << "\n\nПользователь " << users[0].first << " отправляет Kerberos
сообщение (" << users[0].first << ", " << users[1].first << ")";

    uint64_t sec =
duration_cast<seconds>(system_clock::now().time_since_epoch()).count();
    cpp_int timestamp(sec), ttl = 300, mainKey = abs(rand() * rand() * rand());
    string messageForUser1 = "(" + to_string(timestamp) + ", " + to_string(ttl)
+ ", " + to_string(mainKey) + ", " + users[1].first + ")";
    vector <cpp_int> keysPGYforUser1 = generateKeys(keyUser1);
    vector <pair <cpp_int, cpp_int>> encMessageForUser1 =
encryption(keysPGYforUser1, keyUser1, messageForUser1);
    cout << "\n\nKerberos создаёт сообщение (T, L, K, B) = " << messageForUser1
<< " для пользователя " << users[0].first << ", генерирует открытые ключи ";
    cout << "(p, g, y) = (" << keysPGYforUser1[0] << ", " << keysPGYforUser1[1]
<< ", " << keysPGYforUser1[2] << ") и зашифровывает сообщение общим секретным
ключом: ";
    for (int i = 0; i < encMessageForUser1.size(); i++)
        cout << "(" << encMessageForUser1[i].first << ":" <<
encMessageForUser1[i].second << ") ";

    string messageForUser2 = "(" + to_string(timestamp) + ", " + to_string(ttl)
+ ", " + to_string(mainKey) + ", " + users[0].first + ")";
    vector <cpp_int> keysPGYforUser2 = generateKeys(keyUser2);
    vector <pair <cpp_int, cpp_int>> encMessageForUser2 =
encryption(keysPGYforUser2, keyUser2, messageForUser2);
    cout << "\n\nKerberos создаёт сообщение (T, L, K, A) = " << messageForUser2
<< " для пользователя " << users[1].first << ", генерирует открытые ключи ";
    cout << "(p, g, y) = (" << keysPGYforUser2[0] << ", " << keysPGYforUser2[1]
<< ", " << keysPGYforUser2[2] << ") и зашифровывает сообщение общим секретным
ключом: ";
    for (int i = 0; i < encMessageForUser2.size(); i++)
        cout << "(" << encMessageForUser2[i].first << ":" <<
encMessageForUser2[i].second << ") ";
```

```
    cout << "\n\nЗашифрованные сообщения и сгенерированные ключи отправляются
пользователю " << users[0].first;

    string decMessageForUser1 = decryption(keysPGYforUser1, keyUser1,
encMessageForUser1);
    cout << "\n\nПользователь " << users[0].first << " получает сообщения.
Сообщение для пользователя " << users[0].first << ": " << decMessageForUser1;

    string message = "(" + users[0].first + ", " + to_string(timestamp) + ")";
    vector <cpp_int> keys = generateKeys(mainKey);
    vector <pair <cpp_int, cpp_int>> encMessage = encryption(keys, mainKey,
message);
    cout << ". Далее пользователя формирует собщение (A, T) = " << message << "
и зашифровывает сеансовым ключом : ";
    for (int i = 0; i < encMessage.size(); i++)
        cout << "(" << encMessage[i].first << ":" << encMessage[i].second << ")
";
    cout << "\nСообщения (A, T) и (T, L, K, A) передаёт пользователю " <<
users[1].first;

    string decMessageForUser2 = decryption(keysPGYforUser2, keyUser2,
encMessageForUser2);
    string decMessage = decryption(keys, mainKey, encMessage);
    cout << "\n\nПользователь " << users[1].first << " получает свои сообщение и
расшифровывает: ";
    cout << "(A, T) = " << decMessage << " и (T, L, K, A) = " <<
decMessageForUser2;

    timestamp += 1;
    message = to_string(timestamp);
    encMessage = encryption(keys, mainKey, message);
    cout << "\n\nПользователь " << users[1].first << " Создаёт сообщение (T + 1)
= " << message << ", зашифровывает его сеансовым ключом и отправляет";
    cout << " пользователю " << users[0].first << ": ";
    for (int i = 0; i < encMessage.size(); i++)
        cout << "(" << encMessage[i].first << ":" << encMessage[i].second << ")
";

    cout << "\n\nПользователь " << users[0].first << " получает сообщение от "
<< users[1].first << " и расшифровывает его: " << decryption(keys, mainKey,
encMessage);
}


int main() {
    srand(time(0));
    setlocale(LC_ALL, "ru");
    cout << "\tПротокол Kerberos";

    pair <string, string>* users = new pair <string, string>[2];
    string login, password;
    cout << "\nВведите логин и пароль пользователя 1: ";
    cin >> login >> password;
    users[0] = make_pair(login, password);
    cout << "Введите логин и пароль пользователя 2: ";
    cin >> login >> password;
    users[1] = make_pair(login, password);

    kerberos(users);
    cout << endl;
    return 0;
}
```