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

Скрытый канал связи на основе схемы Эль-Гамаля

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
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

1 Теоретическая часть

Цель работы – изучение скрытого канала связи на основе схемы Эль-Гамаля.

1.1 Описание алгоритма

Основные параметры схемы Эль-Гамаля. Алиса подписывает открытое сообщение m, Боб проверяет подпись. Генерируются: p – большое простое число, с условием $m < p$; g – примитивный корень по модулю p; x – случайное числа Алисы, $1 < x < p$, с условием $(x, p - 1) = 1$; $y = g^x \bmod p$. Элементы $\{p, g, y\}$ объявляется открытым ключом, элемент $\{x\}$ – закрытым ключом Алисы. В случае скрытого канала $\{x\}$ должно быть известно и Бобу.

Генерация подписи

1. A: $\{k\}$, где k – скрываемое сообщение Алисы, оно должно удовлетворять условиям $(k, p - 1) = 1$, $1 < k < p - 1$;
2. A: $\{a\}$, где $a = g^k \bmod p$;
3. A: $\{b\}$, где $b = k^{-1} (m - xa) \bmod (p - 1)$, подписью являются a и b, которые удовлетворяют уравнению $m = (xa + kb) \bmod (p - 1)$. Для скрытого канала должно выполняться дополнительное условие: числа $(m - xa)$ и $(p - 1)$ должны быть взаимно простыми. Это условие выполнить несложно, поскольку m (безобидное сообщение) при необходимости всегда можно немного подправить.
4. $A \rightarrow W\,(B)$: $\{m, a, b\}$.

Проверка подписи

5. W: проверяет, что $y^a a^b \bmod p = g^m \bmod p$.
6. $W \rightarrow B$: $\{m, a, b\}$.
7. B: проверяет, что $y^a a^b \bmod p = g^m \bmod p$.

Получение секретного сообщения

8. B: извлекает секретное сообщение, $k = b^{-1} (m - xa) \bmod (p - 1)$.

2 Описание программы

Программа, представленная ниже, содержит следующие функции:

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

• pow(*x*, *y*) – возводить число *x* в степень *y*;

• binForm(*x*) – представление числа *x* в 2-чной системе счисления;

• decForm(*x*) – представление числа *x* в 10-чной системе счисления;

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

• hashStr(*str*) – выдаёт хэш строки *str*;

• usualEuclid(*a*, *b*) – вычисление НОД чисел *a* и *b* обычным алгоритмом Евклида;

• advancedEuclid(*a*, *b*) – вычисление обратного элемента для *a* в поле *b*;

• correctMessageHash(*m*) – корректировка строки *m*, если не выполняется условие взаимной простоты;

• correctMessagecode(*kCodeStr*) – корректировка строки *kCodeStr*, если не выполняется условие взаимной простоты;

• generatesimpleNum(*m*) – генерация простого числа размером ≈ *m* бит;

• generateKeys() – генерация ключей для криптосистемы Эль-Гамаля;

• generateSignature(*k*) – генерация подписи со скрытым сообщением *k*;

• checkSignature() – проверка подписи Эль-Гамаля;

• getSecret() – извлечение секретного сообщения из подпись Эль-Гамаля.

2.1 Примеры работы программы

```
        Скрытый канал на основе схемы Эль-Гамаля
Введите подписываемое сообщение m: Google.com
Хэш подписываемого сообщения m: 54563233620417677831278941240405813182 7

Открытый ключ {p, g, y}: {333525701521923016013996095632435802199064 83, 135079486414583166822808633720936470400900 6, 212
81469467657884887428450935040031419781086}
Закрытый ключ {x}: {4164568549672231937}. Данный ключ также известен и Бобу

Введите скрываемое сообщение k: Yandex.ru
Код скрываемого сообщения k: 865974636485387882
НОД(k, p-1) != 1. Корректировка сообщения k: yandex.ru\0
Код откорректированого скрываемого сообщения k: 8659746364853878824911

a = 20916048702144722991803209880544188955179404

b = 22258743116257285133473629164274629605227391

Алиса отправляет Уолтеру тройку {m, a, b}: {Google.com, 20916048702144722991803209880544188955179404, 222587431162572851
33473629164274629605227391}

Уолтер проверяет, что y^a * a^b (mod p) = g^m (mod p): 17433249088562948644194721319719525145043 81 = 1743324908856294864
41947213197195251450438 1
Проверка пройдена! Уолтер отправляет тройку {m, a, b} Бобу

Боб проверяет, что y^a * a^b (mod p) = g^m (mod p): 17433249088562948644194721319719525145043 81 = 1743324908856294864419
472131971952514504381

Боб извлекает секретное сообщение k = b^-1 * (m - xa) (mod p-1) = 8659746364853878824911 = yandex.ru\0
```

```
        Скрытый канал на основе схемы Эль-Гамаля
Введите подписываемое сообщение m: Hello, World!
Хэш подписываемого сообщения m: 12594911688942956799146188455433980663 58

Открытый ключ {p, g, y}: {536773708316316035193421045696683411023464 79, 498888421236740364824370302901407136361734 19, 45
74369140378668908109768348501222683047904 0}
Закрытый ключ {x}: {1421768862481214017}. Данный ключ также известен и Бобу

Введите скрываемое сообщение k: my cvc = 777
Код скрываемого сообщения k: 738622628362224422181818
НОД(k, p-1) != 1. Корректировка сообщения k: my cvc = 777\0
Код откорректированого скрываемого сообщения k: 7386226283622244221818184911

a = 475642179409873142551747250799885135807917 65

b = 284627688565482195787193622098630801452790 25

Алиса отправляет Уолтеру тройку {m, a, b}: {Hello, World!, 475642179409873142551747250799885135807917 65, 284627688565482
19578719362209863080145279025}

Уолтер проверяет, что y^a * a^b (mod p) = g^m (mod p): 19641802113486650786279374747048317687810778 = 196418021134866507
86279374747048317687810778
Проверка пройдена! Уолтер отправляет тройку {m, a, b} Бобу

Боб проверяет, что y^a * a^b (mod p) = g^m (mod p): 19641802113486650786279374747048317687810778 = 196418021134866507862
79374747048317687810778

Боб извлекает секретное сообщение k = b^-1 * (m - xa) (mod p-1) = 7386226283622244221818184911 = my cvc = 777\0
```

3 Листинг кода

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <set>
#include <boost/multiprecision/cpp_int.hpp>

using namespace std;
using namespace boost::multiprecision;

map <char, string> book{ {'0', "11"}, {'1', "12"}, {'2', "13"}, {'3', "14"},
{'4', "15"}, {'5', "16"}, {'6', "17"}, {'7', "18"}, {'8', "19"},
                         {'9', "21"}, {' ', "22"}, {'!', "23"}, {'"', "24"},
{'#', "25"}, {'$', "26"}, {'%', "27"}, {'^', "28"}, {'&', "29"},
                         {'\'', "31"}, {'(', "32"}, {')', "33"}, {'*', "34"},
{'+', "35"}, {',', "36"}, {'-', "37"}, {'.', "38"}, {'/', "39"},
                         {':', "41"}, {';', "42"}, {'<', "43"}, {'=', "44"},
{'>', "45"}, {'?', "46"}, {'@', "47"}, {'[', "48"}, {'\\', "49"},
                         {']', "51"}, {'_', "52"}, {'`', "53"}, {'{', "54"},
{'}', "55"}, {'|', "56"}, {'~', "57"}, {'\n', "58"}, {'a', "59"},
                         {'b', "61"}, {'c', "62"}, {'d', "63"}, {'e', "64"},
{'f', "65"}, {'g', "66"}, {'h', "67"}, {'i', "68"}, {'j', "69"},
                         {'k', "71"}, {'l', "72"}, {'m', "73"}, {'n', "74"},
{'o', "75"}, {'p', "76"}, {'q', "77"}, {'r', "78"}, {'s', "79"},
                         {'t', "81"}, {'u', "82"}, {'v', "83"}, {'w', "84"},
{'x', "85"}, {'y', "86"}, {'z', "87"} };

map <string, char> bookRvs{ {"11", '0'}, {"12", '1'}, {"13", '2'}, {"14", '3'},
{"15", '4'}, {"16", '5'}, {"17", '6'}, {"18", '7'}, {"19", '8'},
                           {"21", '9'}, {"22", ' '}, {"23", '!'}, {"24", '"'},
{"25", '#'}, {"26", '$'}, {"27", '%'}, {"28", '^'}, {"29", '&'},
                           {"31", '\''}, {"32", '('}, {"33", ')'}, {"34", '*'},
{"35", '+'}, {"36", ','}, {"37", '-'}, {"38", '.'}, {"39", '/'},
                           {"41", ':'}, {"42", ';'}, {"43", '<'}, {"44", '='},
{"45", '>'}, {"46", '?'}, {"47", '@'}, {"48", '['}, {"49", '\\'},
                           {"51", ']'}, {"52", '_'}, {"53", '`'}, {"54", '{'},
{"55", '}'}, {"56", '|'}, {"57", '~'}, {"58", '\n'}, {"59", 'a'},
                           {"61", 'b'}, {"62", 'c'}, {"63", 'd'}, {"64", 'e'},
{"65", 'f'}, {"66", 'g'}, {"67", 'h'}, {"68", 'i'}, {"69", 'j'},
                           {"71", 'k'}, {"72", 'l'}, {"73", 'm'}, {"74", 'n'},
{"75", 'o'}, {"76", 'p'}, {"77", 'q'}, {"78", 'r'}, {"79", 's'},
                           {"81", 't'}, {"82", 'u'}, {"83", 'v'}, {"84", 'w'},
{"85", 'x'}, {"86", 'y'}, {"87", 'z'} };

set <char> workSymbs{ '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ',
'!', '\"', '#', '$', '%', '^', '&', '\'', '(', ')', '*', '+', ',', '-', '.',
'/',
                      ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '_',
'`', '{', '}', '|', '~', '\n', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
                      'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
'u', 'v', 'w', 'x', 'y', 'z' };

set <char> upSymbs{ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };



class Pattern
{
private:
```

```
    static vector <cpp_int> deg2(cpp_int el, cpp_int n) {//Раскладываем число на
степени двойки
        vector <cpp_int> res;
        while (n != 0) {
            if (n / el == 1) {
                res.push_back(el);
                n -= el;
                el = 1;
            }
            else
                el *= 2;
        }
        return res;
    }


    static cpp_int multMod(cpp_int n, cpp_int mod, vector <pair <cpp_int,
cpp_int>> lst) {//Умножаем число по модулю
        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);
        }
    }
public:
    static cpp_int powClosed(cpp_int x, cpp_int y, cpp_int mod) {//Возводим
число в степени по модулю
        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;
    }


    static cpp_int pow(cpp_int x, cpp_int y) {
        cpp_int res = 1;
        for (int i = 0; i < y; i++)
            res *= x;
        return res;
    }


    static cpp_int decForm(string x) {
```

```
    cpp_int res = 0, deg = 1;
    if (!x.empty() && x.back() == '1')
        res += 1;
    for (short i = x.length() - 2; i >= 0; i--) {
        deg = deg * 2;
        if (x[i] == '1')
            res += deg;
    }
    return res;
}


static string binForm(cpp_int x) {
    string bitter = "";
    while (x != 0) {
        bitter = (x % 2 == 0 ? "0" : "1") + bitter;
        x = x / 2;
    }
    if (bitter == "")
        return "0";
    return bitter;
}


static 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;
}


static cpp_int hashStr(string str) {
    while (str.length() < 2)
```

```
            str += " ";
        string res = "";
        hash <string> hashStr;

        unsigned short offset = ceil(str.length() / 2.0);
        for (unsigned short i = 0; i < str.length(); i += offset)
            res += to_string(hashStr(str.substr(i, offset)));
        return cpp_int(res);
    }


    static 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);
    }


    static pair <cpp_int, cpp_int> advancedEuclid(cpp_int a, cpp_int b) {
        if (a < 0 || b < 0)
            throw string{ "Выполнение невозможно: a < 0 или b < 0" };

        cpp_int q, aPrev = a, aCur = b, aNext = -1;
        cpp_int xPrev = 1, xCur = 0, xNext;
        cpp_int yPrev = 0, yCur = 1, yNext;
        while (aNext != 0) {
            q = aPrev / aCur;
            aNext = aPrev % aCur;
            aPrev = aCur; aCur = aNext;

            xNext = xPrev - (xCur * q);
            xPrev = xCur; xCur = xNext;

            yNext = yPrev - (yCur * q);
            yPrev = yCur; yCur = yNext;
        }

        return make_pair(xPrev, yPrev);
    }
};



class HiddenElGamal : private Pattern
{
private:
    string correctMessageHash(string m)
    {
        cpp_int pDec = p - 1;
        cpp_int m_xa = 2;
        char symb = 20;
        while (usualEuclid(m_xa, pDec) != 1)
        {
            symb++;
            if (symb == 256)
            {
                m = m + char(rand() % (256 - 21) + 21);
```

```
                mHash = hashStr(m);
                symb = 21;
            }

            m_xa = ((hashStr(m + char(symb)) - x * a) + (cpp_int(1e19) * pDec))
% pDec;
            while (m_xa < 0)
                m_xa += pDec;
        }

        return m + char(symb);
    }


    string correctMessageCode(string kCodeStr)
    {
        unsigned short symb = 11;
        while (usualEuclid(cpp_int(kCodeStr + to_string(symb)), p - 1) != 1)
        {
            symb++;
            while (bookRvs.find(to_string(symb)) == bookRvs.end() && symb <= 87)
                symb++;
            if (symb > 87)
            {
                kCodeStr = kCodeStr + book[char(rand() % (122 - 97) + 97)];
                symb = 11;
            }
        }

        return kCodeStr + to_string(symb);
    }
public:
        string m;
        cpp_int mHash, p, g, x, y;
    cpp_int a, b;

        HiddenElGamal(string m)
        {
                this->m = m;
        this->mHash = hashStr(m);
        }

    void generateKeys() {
        cpp_int q = rand();
        while (!miller_rabin(q))
            q++;

        cpp_int s, p = 2, pDec;
        while (!miller_rabin(p)) {
            string sBin = "1";
            int sBinSize = binForm(this->mHash).length();
            for (int i = 0; i < sBinSize; i++)
                sBin = sBin + to_string(rand() % 2);
            s = decForm(sBin);

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

        cpp_int a = 2, g;
        while (pDec > a) {
            g = powClosed(a, pDec / q, p);
```

```
            if (g == 1) {
                a++;
                continue;
            }
            break;
        }
        this->g = g;

        this->x = pow(rand() * rand() * rand(), 2) % p;
        while (usualEuclid(x, p - 1) != 1)
            x++;
        this->y = powClosed(g, this->x, p);
    }


    void generateSignature(string k)
    {
        string kCodeStr = "";
        for (unsigned short i = 0; i < k.length(); i++)
            kCodeStr += book[k[i]];
        cpp_int kCode = cpp_int(kCodeStr);
        cout << "Код скрываемого сообщения k: " << kCode;

        cpp_int pDec = p - 1;
        if (kCode >= pDec)
            throw string{ "Код сообщения k должен быть меньше p - 1!" };

        if (usualEuclid(kCode, pDec) != 1)
        {
            cout << "\nНОД(k, p-1) != 1. Корректировка сообщения k: ";
            kCodeStr = correctMessageCode(kCodeStr + book['\\']);

            for (unsigned short i = 2 * k.length(); i < kCodeStr.length(); i +=
2)
                k += bookRvs[kCodeStr.substr(i, 2)];
            kCode = cpp_int(kCodeStr);

            cout << k;
            cout << "\nКод откорректированого скрываемого сообщения k: " <<
kCode;

            if (kCode >= pDec)
                throw string{ "Сообщение невозможно встроить, т.к. k > p - 1!"
};
        }

        this->a = powClosed(g, kCode, p);
        cout << "\n\na = " << this->a;

        cpp_int m_xa = (mHash - x * a) + (cpp_int(1e18) * (pDec));
        while (m_xa < 0)
            m_xa = (m_xa + pDec) % pDec;
        if (usualEuclid(m_xa, pDec) != 1) {
            cout << "\n\nНОД(m - xa, p-1) != 1. Корректировка сообщения m: ";
            this->m = correctMessageHash(m);
            cout << this->m;
            this->mHash = hashStr(m);
            cout << "\nХэш откорректированного сообщения m: " << this->mHash;
        }

        this->b = advancedEuclid(kCode, pDec).first * (mHash - x * a) % pDec;
        while (b < 0)
            b += pDec;
```

```
        cout << "\n\nb = " << b;

        cout << "\n\nАлиса отправляет Уолтеру тройку {m, a, b}: {" << this->m <<
", " << this->a << ", " << this->b << "}";
    }


    void checkSignature()
    {
        cout << "\n\nУолтер проверяет, что y^a * a^b (mod p) = g^m (mod p): ";
        cpp_int leftSide = powClosed(y, a, p) * powClosed(a, b, p) % p;
        cpp_int rightSide = powClosed(g, mHash, p);
        if (leftSide == rightSide)
            cout << leftSide << " = " << rightSide;
        else
            throw string (to_string(leftSide) + " != " + to_string(rightSide));
        cout << "\nПроверка пройдена! Уолтер отправляет тройку {m, a, b} Бобу";

        cout << "\n\nБоб проверяет, что y^a * a^b (mod p) = g^m (mod p): ";
        leftSide = powClosed(y, a, p) * powClosed(a, b, p) % p;
        rightSide = powClosed(g, mHash, p);
        if (leftSide == rightSide)
            cout << leftSide << " = " << rightSide;
        else
            throw string(to_string(leftSide) + " != " + to_string(rightSide));
    }


    void getSecret()
    {
        cout << "\n\nБоб извлекает секретное сообщение k = b^-1 * (m - xa) (mod
p-1) = ";
        cpp_int kCode = advancedEuclid(b, p - 1).first * (mHash - x * a) % (p -
1);
        while (kCode < 0)
            kCode += p - 1;

        string kCodeStr = to_string(kCode);
        cout << kCodeStr << " = ";
        string k = "";
        for (unsigned short i = 0; i < kCodeStr.length(); i += 2)
            k += bookRvs[kCodeStr.substr(i, 2)];
        cout << k;
    }
};



int main() {
        srand(time(NULL));
        setlocale(LC_ALL, "ru");
        cout << "\tСкрытый канал на основе схемы Эль-Гамаля";

        string m;
        cout << "\nВведите подписываемое сообщение m: ";
        getline(cin, m);

    try
    {
        HiddenElGamal heg(m);
        heg.generateKeys();

        cout << "Хэш подписываемого сообщения m: " << heg.mHash;
```

```
        cout << "\n\nОткрытый ключ {p, g, y}: {" << heg.p << ", " << heg.g << ",
" << heg.y << "}";
        cout << "\nЗакрытый ключ {x}: {" << heg.x << "}. Данный ключ также
известен и Бобу";

        string k;
        cout << "\n\nВведите скрываемое сообщение k: ";
        getline(cin, k);
        for (unsigned short i = 0; i < k.length(); i++)
            if (upSymbs.find(k[i]) != upSymbs.end())
                k[i] = k[i] + 32;

        heg.generateSignature(k);
        heg.checkSignature();
        heg.getSecret();
    }
    catch (string& error)
    {
        cout << endl << error;
    }

    cout << endl;
    return 0;
}
```