Министерство образования и науки Российской Федерации

Московский авиационный институт  
(национальный исследовательский университет)

Кафедра №304

Отчет по лабораторной работе №5

«**Реализация 128-битных алгоритмов шифрования Эль-Гамаля и RSA**»

по дисциплине «Защита информации»

Выполнил

Студент группы 3О-410Б

Сомов Д.Н.

Принял

доцент каф.304, к.т.н.

Петренко Ю.И.

Москва, 2017

**Задача:** разработать программу, которая реализует алгоритмы шифрования Эль-Гамаля и RSA для 128-битных чисел.

**Теория:**

Эль-Гамаль и RSA – алгоритмы шифрования с открытым ключом. Они основаны на на теориях, касающихся функций, которые сами по себе вычисляются довольно просто, но вот обратную к ним функцию вычислить несоизмеримо сложнее. Таким образом, если злоумышленник не узнал закрытую часть ключа, взлом шифра будет представлять для него большую проблему, в то время как абоненты, знающие необходимую пару ключей, производят расшифровку легко.

Алгоритм Эль-Гамаля (улучшенный):

1. Выбрано большое простое число Q. Найдем от него функцию Эйлера (количество натуральных чисел, меньших данного числа и взаимно простых с ним; для простых чисел она равна числу без единицы):

М = Q – 1.

1. Каждый из абонентов выбирает себе число Сx (x – номер абонента), меньшее М и такое, что оно взаимно простое с М.
2. Находится число Dх, такое, что у каждого абонента

Cх \* Dх = 1 (mod M).

1. Есть некоторое сообщение w < M. Первый абонент шифрует его своим паролем Са: w1 = wCa mod Q. Шифровка передается второму абоненту.
2. Второй абонент шифрует шифровку своим ключом Сb: w2 = w1Cb mod Q = wCaCb mod Q.
3. Полученная шифровка передается первому абоненту, тот возводит ее в степень Da по модулю Q, получая wCb mod Q (т.к. Cа \* Dа = 1 (mod M)).
4. Данное сообщение передается второму абоненту, и тот благополучно расшифровывает его, возводя в степень своего второго ключа Db.

Основная идея состоит в том, чтобы по открытому каналу всегда передавались только зашифрованные сообщения, при этом передавать ни ключ С, ни ключ D не нужно, абонентам достаточно договориться только о числе Q.

Алгоритм шифрования RSA:

1. Выбирается два простых числа P и Q и вычисляется их произведение

N = P \* Q.

1. Находится функция Эйлера от N: M = (Q - 1) \* (P - 1).
2. Выбирается число Е, называемое закрытой частью ключа. Пару (Е, М) держат в секрете как закрытый ключ.
3. Вычисляется число D, называемое открытой частью ключа, исходя из соотношения:

D \* E = 1 (mod M).

Пару (D, M) объявляют открытым ключом и рассылают всем абонентам. Даже если она будет перехвачена, расшифровка не представляется возможной без знания числа E и N.

1. Сообщение, которое нужно передать, возводится в степень D по модулю N.
2. Абонент, принявший шифровку, расшифровывает ее путем возведения шифровки в степень E по модулю N.

**Программа на языке Си *(часть кода, отвечающая за реализацию алгоритмов шифрования):***

***bignumbers.h и bignumbers.cpp остаются с л/р 4 неизменными.***

***Main.cpp:***

#include "bignumbers.h"

int main(void)

{

/\* El Gamal: \*/

printf("-------- El Gamal: ---------\n");

Big \*Q = loadBigNumber("ElG\_Q.txt");

Big \*M;

Big \*Ca = loadBigNumber("ElG\_Ca.txt");

Big \*Cb = loadBigNumber("ElG\_Cb.txt");

Big \*Da = loadBigNumber("ElG\_Da.txt");

Big \*Db = (Big \*)calloc(1, sizeof(Big));

Big \*mes = loadBigNumber("ElG\_Mes.txt");

Big \*minusOne = (Big \*)calloc(1, sizeof(Big));

minusOne->aa[1] = 1;

minusOne = bigNegate(minusOne);

M = bigAdd(Q, minusOne);

printf("M: ");

printBigDecimal(M);

Big \*another = (Big \*)calloc(1, sizeof(Big));

extendedBinaryEuclidian(Cb, M, Db, another);

printf("Db: ");

printBigDecimal(Db);

printf("another: ");

printBigDecimal(bigNegate(another));

printf("Changing message...\nW0 = ");

printBigDecimal(mes);

mes = bigModPwr(mes, Ca, Q);

printf("W1 = ");

printBigDecimal(mes);

mes = bigModPwr(mes, Cb, Q);

printf("W2 = ");

printBigDecimal(mes);

mes = bigModPwr(mes, Da, Q);

printf("W3 = ");

printBigDecimal(mes);

mes = bigModPwr(mes, Db, Q);

printf("W4 = ");

printBigDecimal(mes);

system("pause>nul");

/\* RSA: \*/

printf("\n\n--------- RSA: ----------\n");

Big \*P = loadBigNumber("RSA\_P.txt");

Q = loadBigNumber("RSA\_Q.txt");

mes = loadBigNumber("RSA\_Mes.txt");

Big \*D = loadBigNumber("RSA\_D.txt");

// Found with help of another program

// after calculating value of M here.

Big \*E = loadBigNumber("RSA\_E.txt");

Big \*fakeM = (Big \*)calloc(1, sizeof(Big));

fakeM->aa[0] = UINT64\_MAX;

fakeM->aa[1] = UINT64\_MAX;

Big \*N = bigModMult(P, Q, fakeM);

printf("N: ");

printBigDecimal(N);

bigAssign(Ca, bigAdd(P, minusOne)); // (P - 1)

bigAssign(Cb, bigAdd(Q, minusOne)); // (Q - 1)

M = bigModMult(Ca, Cb, fakeM);

printf("M: ");

printBigDecimal(M);

printf("Changing message...\nm0 = ");

printBigDecimal(mes);

mes = bigModPwr(mes, E, N);

printf("m1 = ");

printBigDecimal(mes);

mes = bigModPwr(mes, D, N);

printf("m2 = ");

printBigDecimal(mes);

system("pause>nul");

return 0;

}

**Результаты запуска программы:**

-------- El Gamal: ---------

ElG\_Q.txt: 290 240 842 373 741 630 598 113 635 750 625 829 313

ElG\_Ca.txt: 252 529 549 474 974 097 353 623 767 491 716 932 139

ElG\_Cb.txt: 210 069 209 862 722 085 970 920 785 543 112 481 003

ElG\_Da.txt: 239 877 721 972 003 714 223 385 004 317 028 607 747

ElG\_Mes.txt: 227 608 870 250 470 074 588 217 803 594 508 029 379

M: 0 000 000 290 240 842 373 741 630 598 113 635 750 625 829 312

Db: 0 000 000 042 382 770 461 072 795 281 320 666 542 201 807 107

another: 1 000 000 030 675 610 743 597 301 025 829 929 709 176 343 985

Changing message...

W0 = 0 000 000 227 608 870 250 470 074 588 217 803 594 508 029 379

W1 = 0 000 000 141 566 298 202 362 039 619 884 708 936 956 230 857

W2 = 0 000 000 165 848 771 323 115 304 925 627 790 758 561 697 261

W3 = 0 000 000 140 503 024 812 399 399 137 284 020 691 611 597 575

W4 = 0 000 000 227 608 870 250 470 074 588 217 803 594 508 029 379

--------- RSA: ----------

RSA\_P.txt: 18 446 744 073 511 635 041

RSA\_Q.txt: 13 846 748 073 511 634 959

RSA\_Mes.txt: 13 846 744 073 499 265 217

RSA\_D.txt: 4 294 961 897

RSA\_E.txt: 43 096 150 582 108 858 056 794 774 070 297 743 513

N: 0 000 000 255 427 417 962 459 401 994 780 644 373 224 998 319

M: 0 000 000 255 427 417 962 459 401 962 487 152 226 201 728 320

Changing message...

m0 = 0 000 000 000 000 000 000 000 000 013 846 744 073 499 265 217

m1 = 0 000 000 168 783 682 789 785 681 933 955 467 250 513 172 073

m2 = 0 000 000 000 000 000 000 000 000 013 846 744 073 499 265 217