1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**КУРСОВАЯ РАБОТА**

1. **Криптосистема с открытым ключом**
2. по дисциплине «Структуры данных»

1. Выполнил
2. студент гр. 4851004/20001 Панов Д.Е.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Панков И.Д.

<*подпись*>

1. «22» июня 2023 г.
2. Санкт-Петербург
3. 2023

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1. Обзор условий и входных данных. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2. Теоретические исследования. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

Генерация ключей RSA. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

Генерация ключей Эль Гамаля. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

Электронная подпись файла и её проверка при помощи RSA. . . . . . . . . . 8

Электронная подпись файла и её проверка при помощи Эль Гамаля. . . . 9

Шифрование и расшифрование файла при помощи RSA. . . . . . . . . . . . . . 10

Шифрование и расшифрование файла при помощи Эль Гамаля. . . . . . . 11

3 Работа с длинными числами. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

4 Примеры работы. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

5 ЗАКЛЮЧЕНИЕ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

6 Список использованной литературы. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

Приложение А. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

Приложение Б. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

Приложение В. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

# ВВЕДЕНИЕ

В качестве курсовой работы было необходимо реализовать криптосистему с открытым ключом с двумя методами шифрования (RSA и Эль Гамаля) на языке программирования Си.

**Цель работы**

Написать код на языке Си для реализации криптосистемы с открытым ключом, которая способна:

1. Генерировать ключи заданной длины.

2. Подписывать заданный файл.

3. Проверять подпись.

4. Шифровать заданный файл.

5. Расшифровывать файл.

1. Обзор условий и входных данных

Криптосистема с открытым ключом – консольное приложение, соответственно ввод выполняется через терминал.

Входными данными является строка, передающая команду, которую необходимо выполнить и файлы, из которых брать информацию и в которые сохранять информацию. Примеры входных данных:

* crypt --help – выводит как ей пользоваться
* crypt genkey --size 1000 --pubkey pk.txt --secret secret.txt

Генерирует ключи. Открытый сохраняется в pk.txt, закрытый – в secret.txt. Размер может быть любой, кратный 256: 256, 512, 1024, 2048, 4096 и т.д.

* crypt sign --infile file.txt --secret secret.txt --sigfile signature.txt

Вычислят подпись файла, сохраняет ее в signature.txt.

* crypt check --infile file.txt --pubkey pk.txt --sigfile signature.txt

Проверяет подпись файла, результат пишет в stdout.

* crypt encrypt --infile file.txt --pubkey pk.txt --outfile file.txt.enc

Шифрует файл для получателя. Результат помещает в file.txt.enc.

* crypt decrypt --infile file.txt.enc --secret secret.txt --outfile file.txt

Расшифровывает файл. Результат помещает в file.txt.

1. Теоретические исследования

**Генерация ключей RSA**

В шифровании RSA, как и во всех других ассиметричных методах шифрования используется два ключа – открытый и закрытый, закрытый используется отправителем для того, чтобы зашифровать передаваемое сообщение, а открытый ключ используется получателем для того, чтобы расшифровать сообщение.

Изображение выглядит как текст, снимок экрана, мультфильм, графическая вставка

Автоматически созданное описание

Рисунок 1 — Описание отправки и приёма зашифрованного письма

Для создания ключей необходимо сгенерировать два разных простых числа p и q, далее необходимо найти их произведение равное n;

*n* = *p* \* *q*

Также необходимо вычислить функцию Эйлера — phi(n);

*phi(n)* = (*p* – 1) \* (*q* – 1)

Такой выбор n обусловлен следующим. Закрытый ключ d можно получить, зная открытый e. Зная числа p и q, вычислить функцию Эйлера не является вычислительно сложной задачей, ровно как и нахождение обратного элемента по модулю. Однако в открытом ключе указано именно число n. Таким образом, чтобы вычислить значение функции Эйлера от n (а затем получить закрытый ключ), необходимо решить задачу факторизации, которая является вычислительно сложной задачей для больших n (в современных системах, основанных на RSA, n имеет длину 2048 бит).

Далее необходимо выбрать целое число *e,* 2< *e* < *phi(n)* – 1, *e* и *phi(n)* должны быть взаимно простыми, то есть их НОД должен быть равен 1.

И в конце концов необходимо вычислить число *d* для хранения закрытого ключа, чтобы вычислить *d* необходимо найти обратный элемент e в кольце по модулю phi(n). Обратным к числу *e* по модулю *phi(n)* называется такое число *d*, что:  
 *e* \* *d* ≡ 1(mod *phi(n)*), обратный элемент обозначают как .

Для нахождения обратного элемента по модулю можно использовался расширенный алгоритм Евклида, он находит коэффициенты, с помощью которых НОД двух чисел выражается через сами эти числа.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Рисунок 2 — Описание расширенного алгоритма Евклида

**Генерация ключей Эль Гамаля**

1. Генерация простого числа *p*

2. Нахождение целого числа *g*, такого, что *g* - первообразный корень *p*.

3. Выбор случайного секретного ключа *x*, такого, что 1 ≤ *x* ≤ *p* – 1.

4. Вычисление *y*, которое является числом *g* в степени *x* по модулю *p*.

Открытый ключ – (p, g, y)

Закрытый ключ – (x)

**Электронная подпись файла и её проверка при помощи RSA**

Изображение выглядит как текст, снимок экрана, диаграмма

Автоматически созданное описание

Рисунок 3 — Создание подписи

Изображение выглядит как текст, снимок экрана, Человеческое лицо, мальчик

Автоматически созданное описание

Рисунок 4 — Проверка подписи

Создание подписи:

1. Вычисление хэш – значения передаваемого сообщения
2. Возведение хэш – значения сообщения в степень *d* по модулю *n*

Получателю передаётся зашифрованная подпись и открытый ключ.

Проверка подписи:

1. Вычисление хэш – значения передаваемого сообщения
2. Возведение зашифрованного хэш – значения сообщения в степень *e* по модулю n, если полученное число равно изначальному хэш-значению сообщения, то файл дошёл в целости.

**Электронная подпись файла и её проверка при помощи Эль Гамаля**

Чтобы подписать сообщение M, сначала выбирается случайное число k, взаимно простое с p – 1. Затем вычисляется подпись (r, s) по формулам:

r = (mod p)

s = (M – x \* r) \* (mod(p – 1))

Чтобы проверить подпись, необходимо убедиться, что \* (mod p) = (mod p).

**Шифрование и расшифрование файла при помощи RSA**

* Шифрование сообщения:

Возьмём в качестве сообщения число m (m ∈ [1, n − 1]). Чтобы зашифровать его, необходимо возвести его в степень e по модулю n. Эти числа идут вместе с открытым ключом. За *с* обозначим шифртекст, c (c ∈ [1, n − 1]).

с ≡ mod n

* Расшифрование зашифрованного сообщения:

Для этого возведём его в степень закрытого ключа *d*.

≡ mod n

m и равны так как

Пусть m и n взаимно просты. Тогда:

,

где phi(n) — функция Эйлера.

имеет решения, сравнимые по модулю p\*q.

Изображение выглядит как текст, Шрифт, снимок экрана, письмо

Автоматически созданное описание

Рисунок 5 — Доказательство того, что ≡ m

**Шифрование и расшифрование файла при помощи Эль Гамаля**

* Шифрование сообщения:

1. Выбирается случайное k, такое что 1 ≤ k ≤ n −1.

2. Создается сообщение m, такое что 1 ≤ m ≤ p – 1. Если сообщение слишком большое, то оно делится на части и они шифруются отдельно.

3. Вычисляет числа u = (mod p), v = (mod p).

4. Публикуется шифр текст (u,v).

* Расшифрование зашифрованного сообщения:

m =

1. Работа с длинными числами

При генерации ключей размера 2048, генерируется ключ, чей размер может превышать , однако язык программирования Си не способен поддерживать числа, такого размера, поэтому была разработана дополнительная библиотека, в которой были реализованы: хранение больших чисел, преобразование больших чисел в строку и обратно, операции сложения, вычитания, деление, умножения и нахождения остатка больших чисел. Библиотека больших чисел отображена в Приложении B.

1. Примеры работы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер ключа | Время на вычисление ключа | Время на электронную подпись | Время на проверку электронной подписи | Время на шифрование | Время на расшифрование | Итого |
| 2^5 | 0.1433сек | 0.1853сек | 0.1543сек | 0.1220сек | 0.0745сек | 0.6794 |
| 2^10 | 0.2841сек | 0.3254сек | 0.2451сек | 0.3155сек | 0.1328сек | 1.3029 |
| 2^20 | 0.6848сек | 0.7813сек | 0.3568сек | 0.4762сек | 0.2348сек | 2.5339 |
| 2^30 | 0.8124сек | 0.9274сек | 0.7185сек | 0.9153сек | 0.4179сек | 3.7915 |

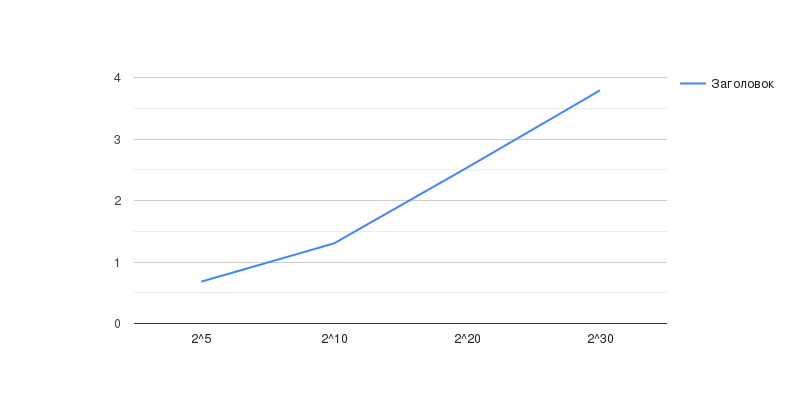


Рисунок 6 — График зависимости времени выполнения от размера ключа

1. ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была реализована криптосистема с открытым ключом в соответствии, со всеми правилами, описанными в начале работы, а именно – реализовано чтение вводимой команды, чтение вводимых файлов, создание ключа RSA и создание ключа Эль Гамаля заданного размера, электронная подпись файла RSA и Эль Гамаля, шифрование и расшифрование текста RSA и Эль Гамаля, а также вывод справки по использованию программы.

Изучена и применена в работе информация об алгоритмах шифрования RSA и Эль Гамаля. Кроме того, в ходе работы была разработана библиотека big\_int, для поддержки больших чисел, чей размер не поддерживается языком программирования Си.

1. список использованной литературы

* 1. <https://planetcalc.ru/3303/>
  2. <https://planetcalc.ru/3311/>
  3. <https://planetcalc.ru/3298/>
  4. <https://habr.com/ru/articles/534014/>
  5. <https://habr.com/ru/companies/otus/articles/486116/>
  6. <https://intuit.ru/studies/courses/13837/1234/lecture/31202>
  7. <https://habr.com/ru/articles/534014/>
  8. <https://kivyfreakt.github.io/elgamal-site/>
  9. <https://studfile.net/preview/7880378/page:38/>

ПРИЛОЖЕНИЕ А

Листинг программы “Kursach.c”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#include <time.h>

#include <math.h>

#include <stdbool.h>

#include <time.h>

#define SIZEOF\_COMMAND 63

#define RSA 1

#define Gamal 2

int\* check\_null(void\* ptr) {

if (ptr == NULL) {

wprintf\_s(L"Ошибка выделения памяти");

exit(EXIT\_FAILURE);

}

return ptr;

}

int is\_prime(int num) {

if (num <= 1) {

return 0;

}

for (int i = 2; i <= sqrt(num); i++) {

if (num % i == 0) {

return 0;

}

}

return 1;

}

unsigned long long is\_prime\_miler(unsigned long long n) {

if (n <= 1) return 0;

if (n <= 3) return 1;

if (n % 2 == 0 || n % 3 == 0) return 0;

for (unsigned long long i = 5; i \* i <= n; i += 6) {

if (n % i == 0 || n % (i + 2) == 0) {

return 0;

}

}

return 1;

}

void help(void) {

wprintf\_s(L"Генерирует ключи. Открытый сохраняется в \

pk.txt, закрытый – в secret.txt. Размер может быть любой, \

кратный 256: 256, 512, 1024, 2048, 4096 и т.д: crypt genkey \

--size 1024 --pubkey pk.txt --secret secret.txt\n\

Вычислят подпись файла, сохраняет ее в signature.txt: crypt sign --infile file.txt --secret secret.txt --sigfile signature.txt\n\

Проверяет подпись файла, результат пишет в stdout: crypt check --infile file.txt --pubkey pk.txt --sigfile signature.txt\n\

Шифрует файл для получателя. Результат помещает в file.txt.enc: crypt encrypt --infile file.txt --pubkey pk.txt --outfile file.txt.enc \n\

Расшифровывает файл. Результат помещает в file.txt: crypt decrypt --infile file.txt.enc --secret secret.txt --outfile file.txt\n");

}

unsigned long long NOD(unsigned long long a, unsigned long long b) {

if (b == 0) {

return a;

}

else {

return NOD(b, a % b);

}

}

///////////////////////////////////////////////////////// НЕПОНЯТНОООО!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

unsigned long long gcdExtended(unsigned long long a, unsigned long long b, unsigned long long\* x, unsigned long long\* y) {

if (a == 0) {

\*x = 0;

\*y = 1;

return b;

}

unsigned long long x1, y1;

unsigned long long gcd = gcdExtended(b % a, a, &x1, &y1);

\*x = y1 - (b / a) \* x1;

\*y = x1;

return gcd;

}

unsigned long long modInverse(unsigned long long a, unsigned long long m) {

unsigned long long x, y;

gcdExtended(a, m, &x, &y);

unsigned long long res = (x % m + m) % m;

return res;

}

//////////////////////////////////////////////////////////// НЕПОНЯТНОООО!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

unsigned long long generate\_prime\_gamal(unsigned long long bits) {

unsigned long long lower\_bound = (unsigned long long)pow(2, (double)(bits - 1));

unsigned long long upper\_bound = (unsigned long long)pow(2, (double)bits);

unsigned long long n;

do n = (unsigned long long)lower\_bound + (unsigned int)(rand() % (upper\_bound - lower\_bound + 1));

while (!is\_prime\_miler(n));

return n;

}

unsigned long long generate\_prime\_rsa(unsigned long long bits) {

unsigned long long lower\_bound = (unsigned long long)pow(2, ((double)bits - 1) / 2);

unsigned long long upper\_bound = (unsigned long long)pow(2, (double)bits / 2);

unsigned long long n;

do {

n = (unsigned long long)lower\_bound + (unsigned int)(rand() % (upper\_bound - lower\_bound + 1));

} while (!is\_prime\_miler(n));

return n;

}

unsigned long long generate\_prime(unsigned long long size) {

unsigned long long n;

do {

n = (unsigned long long)(rand() % size);

} while (!is\_prime\_miler(n));

return n;

}

// Функция для вычисления модулярного возведения в степень

long long int modular\_pow(long long int base, long long int exponent, long long int modulus) {

long long int result = 1;

base = base % modulus;

while (exponent > 0) {

if (exponent % 2 == 1) {

result = (result \* base) % modulus;

}

base = (base \* base) % modulus;

exponent = exponent / 2;

}

return result;

}

/\* Функция для проверки, является ли число a первообразным корнем по модулю m \*/

int is\_primitive\_root(long long a, long long m) {

if (a <= 0 || m <= 1) {

return 0;

}

long long result = 1;

int\* powers = (int\*)calloc(m, sizeof(int));

// Вычисление степеней a по модулю m

for (int i = 0; i != m - 1; i++) {

result = (result \* a) % m;

if (powers != NULL) {

if (powers[result] == 1) {

free(powers);

return 0;

}

powers[result] = 1;

}

}

free(powers);

return 1;

}

// Функция для генерации первообразного корня по модулю m

unsigned long long int generate\_primitive\_root(long long m) {

long long a;

srand((unsigned int)time(NULL));

do {

a = rand() % (m - 1) + 1;

} while (!is\_primitive\_root(a, m));

return a;

}

unsigned long long int powmod(int base, int exponent, int modulus) {

// Возводим base в степень exponent по модулю modulus

int result = 1;

while (exponent > 0) {

if (exponent % 2 == 1) {

result = (result \* base) % modulus;

}

base = (base \* base) % modulus;

exponent = exponent / 2;

}

return result;

}

\_\_int64 find\_module\_of\_value(long long value, long long modul) { // Нахождение значения числа value по модулю modul

long long result = (value % modul + modul) % modul;

return result;

}

unsigned \_\_int64 find\_crc(char\* file\_name\_in, unsigned long long modul) {

FILE\* infile = fopen(file\_name\_in, "rb");

unsigned long long hash[1000] = { 0 }, symbol = fgetc(infile);

int i = 0;

hash[i] = symbol;

symbol = fgetc(infile);

i++;

while (symbol != EOF) {

hash[i] = modular\_pow((symbol + hash[i - 1]), 2, modul);

symbol = fgetc(infile);

i++;

}

return hash[i - 1];

}

void genkey(int size, char\* pubkey, char\* secret) {

//Программа генерирует два простых числа p и q, вычисляет

//их произведение n и функцию Эйлера phi, а затем генерирует

//открытый ключ e и вычисляет соответствующий секретный ключ d.

//Функция is\_prime проверяет, является ли число простым,

//а функция gcd находит наибольший общий делитель двух чисел.

//Для проверки на простоту больших чисел Тест Милера - Рабина

//Для возведения в степень не обязательно умножать число на себя,

//можно делать побитовый сдвиг!!

unsigned short int type = 0;

printf("Выберите шифрование:\n1. RSA\n2. Эль-Гамаля\n");

scanf\_s("%hu", &type);

FILE\* public\_file = NULL, \* secret\_file = NULL;

public\_file = fopen(pubkey, "wb");

secret\_file = fopen(secret, "wb");

if (type == RSA) {

unsigned long long p, q, n, phi, e, d;

srand((unsigned int)time(NULL)); // Инициализация генератора случайных чисел

// Генерация простого числа p

p = generate\_prime\_rsa(size);

// Генерация простого числа q

q = generate\_prime\_rsa(size);

n = p \* q; // Вычисление произведения p и q

phi = (p - 1) \* (q - 1); // Вычисление функции Эйлера для n

// Генерация открытого ключа e

do e = rand() % (phi - 2) + 2;

while (NOD(e, phi) != 1);

// Вычисление секретного ключа d

unsigned long long inverse\_e = modInverse(e, phi); // Вычисление e по модулю phi

d = inverse\_e % phi;

// Вывод ключей на экран

printf\_s("Открытый ключ: (%llu, %llu)\n", e, n);

printf\_s("Секретный ключ: (%llu, %llu)\n", d, n);

fprintf(public\_file, "%llu %llu", e, n);

fprintf(secret\_file, "%llu %llu", d, n);

}

else if (type == Gamal)

{

srand((unsigned int)time(NULL));

unsigned long long int p, g, x, y;

p = generate\_prime\_gamal(size); // Написать функцию для нахождения простого числа нужного размера

g = generate\_primitive\_root(p);

printf\_s("Primitive root of %lld is %lld\n", p, g);

x = rand() % (p - 1) + 1; // Закрытый ключ

y = modular\_pow(g, x, p); // Открытый ключ

printf\_s("Открытый ключ: (%lld, %lld, %lld)\n", y, g, p);

printf\_s("Секретный ключ: (%lld)\n", x);

fprintf(public\_file, "%lld %lld %lld", y, g, p);

fprintf(secret\_file, "%lld %lld", x, p);

}

else

{

printf\_s("Некорректный ввод.");

}

}

void sign(char\* name\_infile, char\* name\_secret, char\* name\_sigfile) {

unsigned long long int crc;

char str[50], key[50] = { 0 };

int i = 0, j = 0;

unsigned short int type = 0;

printf("Выберите шифрование:\n1. RSA\n2. Эль-Гамаля\n");

scanf\_s("%hu", &type);

FILE\* sigfile = fopen(name\_sigfile, "wb");

FILE\* secret\_key = fopen(name\_secret, "rb");

fgets(str, sizeof(str), secret\_key);

if (type == RSA) {

unsigned long long int d, n;

while (str[i] != ' ') {

key[i] = str[i];

i++;

}

d = atoi(key);

i++;

while (str[i] != '\0') {

key[j] = str[i];

i++;

j++;

}

key[j] = '\0';

n = atoi(key);

crc = find\_crc(name\_infile, n);

fprintf(sigfile, "%lld", modular\_pow(crc, d, n));

printf("Вычисленная подпись: %llu\n", crc);

}

else if (type == Gamal) {

char name\_pubkey[50] = { 0 };

long long int p, x, k, g;

while (str[i] != '\0') {

key[i] = str[i];

i++;

}

x = atoi(key);

printf("Введите название файла, который хранит открытый ключ: ");

scanf\_s("%s", &name\_pubkey, 50);

FILE\* pubkey = fopen(name\_pubkey, "rb");

fgets(str, sizeof(str), pubkey);

i = 0;

while (str[i] != ' ') {

i++;

}

key[i] = '\0';

i++;

while (str[i] != ' ') {

key[j] = str[i];

i++;

j++;

}

key[j] = '\0';

g = atoi(key);

i++;

j = 0;

while (str[i] != '\0') {

key[j] = str[i];

i++;

j++;

}

key[j] = '\0';

p = atoi(key);

crc = find\_crc(name\_infile, p); // Нахождение хэш значения файла по модулю p

do k = generate\_prime(p); //

while (NOD(k, p - 1) != 1 && k < p - 1 && k > 0); // Выбор случайного числа k

long long int r = modular\_pow(g, k, p);

long long int inverse\_k = modular\_pow(k, p - 2, p - 1); // Вычисление обратного элемента для k

long long int s = (((long long int)crc - x \* r) \* inverse\_k) % (p - 1);

fprintf(sigfile, "%lld %lld", r, s);

printf("Вычисленная подпись: %llu\n", crc);

}

}

void check(char\* name\_infile, char\* name\_pubkey, char\* name\_sigfile) {

unsigned long long int crc, e, n, h;

char str[50], key[50] = { 0 }, signature[64];

int i = 0, j = 0;

unsigned short int type = 0;

printf("Выберите шифрование:\n1. RSA\n2. Эль-Гамаля\n");

scanf\_s("%hu", &type);

FILE\* sigfile = fopen(name\_sigfile, "rb");

FILE\* pubkey = fopen(name\_pubkey, "rb");

if (type == RSA) {

fgets(str, sizeof(str), pubkey);

while (str[i] != ' ') {

key[i] = str[i];

i++;

}

e = atoi(key);

i++;

while (str[i] != '\0') {

key[j] = str[i];

i++;

j++;

}

n = atoi(key);

fgets(signature, sizeof(signature), sigfile);

crc = find\_crc(name\_infile, n);

h = modular\_pow(atoi(signature), e, n);

if (h == crc)

printf("Передаваемая подпись: %llu\nВычисленная подпись: %llu\nПодписи совпадают.\n", h, crc);

else

printf("Передаваемая подпись: %llu\nВычисленная подпись: %llu\nПодписи не совпадают.\n", h, crc);

}

else if (type == Gamal) {

fgets(str, sizeof(str), pubkey);

fgets(signature, sizeof(signature), sigfile);

long long int p, g, y, r, s;

long long podpis;

while (str[i] != ' ') {

key[j] = str[i];

i++;

j++;

}

key[j] = '\0';

y = atoi(key);

i++;

j = 0;

while (str[i] != ' ') {

key[j] = str[i];

i++;

j++;

}

key[j] = '\0';

g = atoi(key);

i++;

j = 0;

while (str[i] != '\0') {

key[j] = str[i];

i++;

j++;

}

key[j] = '\0';

p = atoi(key);

i = 0;

j = 0;

while (signature[i] != ' ') {

key[j] = signature[i];

i++;

j++;

}

key[j] = '\0';

r = atoi(key);

i++;

j = 0;

while (signature[i] != '\0') {

key[j] = signature[i];

i++;

j++;

}

key[j] = '\0';

s = atoi(key);

crc = find\_crc(name\_infile, p); // Нахождение хэш значения файла по модулю p

if (r > 0 && r < p && s > 0 && s < p - 1) {

podpis = (modular\_pow(y, r, p) \* modular\_pow(r, s, p)) % p;

if (podpis == modular\_pow(g, (long long)crc, p))

printf("Проверка подписи:\nПодпись корректна.");

else

printf("Проверка подписи: \nПодпись некорректна");

}

else

printf("Проверка подписи: \nПодпись некорректна (возможно, параметры не удовлетворяют условиям)");

}

}

long long int modExp(long long int a, long long int b, long long int m) { // а в степень b, mod m

long long int res = 1;

while (b > 0) {

if (b & 1) {

res = (res \* a) % m;

}

a = (a \* a) % m;

b >>= 1;

}

return res;

}

void encrypt(char\* name\_infile, char\* name\_pubkey, char\* name\_outfile) {

unsigned short int type = 0;

printf("Выберите шифрование:\n1. RSA\n2. Эль-Гамаля\n");

scanf\_s("%hu", &type);

if (type == RSA)

{

FILE\* input = fopen(name\_infile, "r+");

FILE\* output = fopen(name\_outfile, "w+");

FILE\* pk = fopen(name\_pubkey, "r");

long long int e, n;

fscanf\_s(pk, "%lld %lld", &e, &n);

fclose(pk);

const int block = 128;

unsigned char\* buffer = calloc(block, sizeof(char));

while (!feof(input)) {

if (buffer != NULL) {

short int bytesRead = (short)fread(buffer, 1, block, input);

// Шифрование блока данных

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

long long int encrypted = modExp(buffer[i], e, n);

fwrite(&encrypted, sizeof(long long int), 1, output);

}

}

}

// Закрытие файлов

fclose(input);

fclose(output);

printf("File encrypted successfully.\n");

}

else if (type == Gamal)

{

FILE\* input = fopen(name\_infile, "r");

FILE\* output = fopen(name\_outfile, "w");

FILE\* pubkey = fopen(name\_pubkey, "r");

unsigned long long int g, p, y;

fscanf\_s(pubkey, "%llu %llu %llu", &g, &p, &y);

fclose(pubkey);

const int block\_size = 128;

unsigned char\* buffer = calloc(block\_size, sizeof(unsigned char));

while (!feof(input)) {

size\_t bytes\_read = fread(buffer, 1, block\_size, input);

// Шифрование блока данных

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

unsigned char text = buffer[i];

unsigned long long int k = rand(); // Случайное число k

unsigned long long int c1 = modExp(g, k, p);

unsigned long long int c2 = (modExp(y, k, p) \* text) % p;

fwrite(&c1, sizeof(unsigned long long int), 1, output);

fwrite(&c2, sizeof(unsigned long long int), 1, output);

}

}

// Закрытие файлов

fclose(input);

fclose(output);

printf("File encrypted successfully.\n");

}

else

{

printf("Выберите верный вариант.");

}

}

void decrypt(char\* name\_infile, char\* name\_secret, char\* name\_outfile) {

unsigned short int type = 0;

printf("Выберите шифрование:\n1. RSA\n2. Эль-Гамаля\n");

scanf\_s("%hu", &type);

if (type == RSA) {

FILE\* input = fopen(name\_infile, "rb");

FILE\* output = fopen(name\_outfile, "w");

FILE\* secretKey = fopen(name\_secret, "r");

if (input == NULL || output == NULL || secretKey == NULL) {

exit(1);

}

long long int d, n;

fscanf\_s(secretKey, "%lld %lld", &d, &n);

fclose(secretKey);

const int blockSize = sizeof(long long int);

unsigned char\* buffer = calloc(blockSize, sizeof(unsigned char));

if (buffer != NULL) {

while (fread(buffer, blockSize, 1, input) == 1) {

long long int encrypted;

memcpy(&encrypted, buffer, blockSize);

long long int decrypted = modExp(encrypted, d, n);

fprintf(output, "%lld", decrypted);

}

}

free(buffer);

fclose(input);

fclose(output);

printf("File decrypted successfully.\n");

}

else if (type == Gamal)

{

}

else

{

printf("Выберите верный вариант.");

}

}

int main(int argc, char\*\* argv)

{

setlocale(LC\_ALL, "ru-Ru");

double time\_spent = 0.0;

clock\_t begin = clock();

char command[1000] = { 0 };

char command\_size[SIZEOF\_COMMAND] = { 0 },

size[SIZEOF\_COMMAND] = { 0 },

command\_infile[SIZEOF\_COMMAND] = { 0 },

name\_infile[SIZEOF\_COMMAND] = { 0 },

command\_pubkey[SIZEOF\_COMMAND] = { 0 },

name\_pubkey[SIZEOF\_COMMAND] = { 0 },

command\_secret[SIZEOF\_COMMAND] = { 0 },

name\_secret[SIZEOF\_COMMAND] = { 0 },

command\_sigfile[SIZEOF\_COMMAND] = { 0 },

name\_sigfile[SIZEOF\_COMMAND] = { 0 },

command\_outfile[SIZEOF\_COMMAND] = { 0 },

name\_outfile[SIZEOF\_COMMAND] = { 0 };

int count = 0, i = 0;

wprintf\_s(L"Для получения справки по коммандам введите \"crypt --help\"\n");

while (count != 2) {

command[i] = (char)getc(stdin);

if (command[i] == ' ' || command[i] == '\0' || command[i] == '\n')

count++;

i++;

}

command[strlen(command) - 1] = '\0';

if (strncmp(command, "crypt", 5) == 0) {

if (strstr(command, argv[1])) {

help();

}

else if (strstr(command, argv[2])) {

scanf\_s("%s %s %s %s %s %s",

&command\_size, SIZEOF\_COMMAND,

&size, SIZEOF\_COMMAND,

&command\_pubkey, SIZEOF\_COMMAND,

&name\_pubkey, SIZEOF\_COMMAND,

&command\_secret, SIZEOF\_COMMAND,

&name\_secret, SIZEOF\_COMMAND);

if (!strcmp(command\_size, "--size") && !strcmp(command\_pubkey, "--pubkey") && !strcmp(command\_secret, "--secret"))

genkey(atoi(size), name\_pubkey, name\_secret);

else

wprintf\_s(L"Неверно введена команда, для получения справки по коммандам введите \"crypt --help\"\n");

}

else if (strstr(command, argv[3])) {

scanf\_s("%s %s %s %s %s %s",

&command\_infile, SIZEOF\_COMMAND,

&name\_infile, SIZEOF\_COMMAND,

&command\_secret, SIZEOF\_COMMAND,

&name\_secret, SIZEOF\_COMMAND,

&command\_sigfile, SIZEOF\_COMMAND,

&name\_sigfile, SIZEOF\_COMMAND);

if (!strcmp(command\_infile, "--infile") && !strcmp(command\_secret, "--secret") && !strcmp(command\_sigfile, "--sigfile"))

sign(name\_infile, name\_secret, name\_sigfile);

else

wprintf\_s(L"Неверно введена команда, для получения справки по коммандам введите \"crypt --help\"\n");

}

else if (strstr(command, argv[4])) {

scanf\_s("%s %s %s %s %s %s",

&command\_infile, SIZEOF\_COMMAND,

&name\_infile, SIZEOF\_COMMAND,

&command\_pubkey, SIZEOF\_COMMAND,

&name\_pubkey, SIZEOF\_COMMAND,

&command\_sigfile, SIZEOF\_COMMAND,

&name\_sigfile, SIZEOF\_COMMAND);

if (!strcmp(command\_infile, "--infile") && !strcmp(command\_pubkey, "--pubkey") && !strcmp(command\_sigfile, "--sigfile"))

check(name\_infile, name\_pubkey, name\_sigfile);

else

wprintf\_s(L"Неверно введена команда, для получения справки по коммандам введите \"crypt --help\"\n");

}

else if (strstr(command, argv[5])) {

scanf\_s("%s %s %s %s %s %s",

&command\_infile, SIZEOF\_COMMAND,

&name\_infile, SIZEOF\_COMMAND,

&command\_pubkey, SIZEOF\_COMMAND,

&name\_pubkey, SIZEOF\_COMMAND,

&command\_outfile, SIZEOF\_COMMAND,

&name\_outfile, SIZEOF\_COMMAND);

if (!strcmp(command\_infile, "--infile") && !strcmp(command\_pubkey, "--pubkey") && !strcmp(command\_outfile, "--outfile"))

encrypt(name\_infile, name\_pubkey, name\_outfile);

else

wprintf\_s(L"Неверно введена команда, для получения справки по коммандам введите \"crypt --help\"\n");

}

else if (strstr(command, argv[argc - 1])) {

scanf\_s("%s %s %s %s %s %s",

&command\_infile, SIZEOF\_COMMAND,

&name\_infile, SIZEOF\_COMMAND,

&command\_secret, SIZEOF\_COMMAND,

&name\_secret, SIZEOF\_COMMAND,

&command\_outfile, SIZEOF\_COMMAND,

&name\_outfile, SIZEOF\_COMMAND);

if (!strcmp(command\_infile, "--infile") && !strcmp(command\_secret, "--secret") && !strcmp(command\_outfile, "--outfile"))

decrypt(name\_infile, name\_secret, name\_outfile);

else

wprintf\_s(L"Неверно введена команда, для получения справки по коммандам введите \"crypt --help\"\n");

}

}

else {

wprintf\_s(L"Неверная команда");

}

clock\_t end = clock();

time\_spent = ((double)end - (double)begin) / 10000;

printf("%f", time\_spent);

return 0;

}

# Приложение Б

Листинг программы “Arifmetika.c”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#include <time.h>

#include <math.h>

#include <stdbool.h>

typedef struct {

char\* digits;

int length;

bool is\_negative;

char trash[3];

} big\_int;

big\_int\* subtract(big\_int\* a, big\_int\* b);

void free\_big\_int(big\_int\* number) { // освобождение памяти из-под big\_int

free(number->digits);

free(number);

}

void remove\_leading\_zeros(big\_int\* number) { // удаление незначащих ведущих нулей

int i = 0;

if (number->digits[i] == '0') {

// ищем первую цифру не ноль

while (number->digits[i] == '0' && number->length > 1) i++;

if (i == number->length) i--;

// копируем цифры после нуля

int new\_length = number->length - i;

char\* new\_digits = (char\*)malloc(sizeof(char) \* new\_length + 1);

strcpy(new\_digits, &number->digits[i]);

free(number->digits);

number->digits = new\_digits;

number->length = new\_length;

// если ноль отрицательный, меняем знак

if (number->digits[0] == '0' && number->length == 1 && number->is\_negative == true)

number->is\_negative = false;

}

}

big\_int\* init\_big\_int\_zero(int length) {

big\_int\* result = (big\_int\*)malloc(sizeof(big\_int));

result->is\_negative = false;

result->length = length;

result->digits = (char\*)malloc(sizeof(char) \* length + 1);

for (int i = 0; i != length; i++) {

result->digits[i] = '0';

}

result->digits[length] = '\0';

return result;

}

big\_int\* init\_big\_int(const char\* str) { // создание big\_int из строки

big\_int\* result = (big\_int\*)malloc(sizeof(big\_int));

int i;

if (str[0] == '-') {

result->is\_negative = true;

i = 1;

}

else {

result->is\_negative = false;

i = 0;

}

// копирование цифр из строки в digits

int str\_length = (int)strlen(str) - i;

result->length = str\_length;

result->digits = (char\*)malloc(sizeof(char) \* str\_length + 1);

strcpy(result->digits, &str[i]);

remove\_leading\_zeros(result);

return result;

}

int abs\_compare(big\_int\* a, big\_int\* b) { // сравн. по модулю

if (a->length > b->length)

return 1;

else if (a->length < b->length)

return -1;

else {

int i;

for (i = 0; i < a->length; i++) {

if (a->digits[i] > b->digits[i])

return 1;

else if (a->digits[i] < b->digits[i])

return -1;

}

return 0;

}

}

big\_int\* add(big\_int\* a, big\_int\* b) {

if (a->length < b->length) { // свап местами

big\_int\* tmp = a;

a = b;

b = tmp;

}

if (a->is\_negative == b->is\_negative) { // если числа одинаковых знаков, выполняем сложение

int length = a->length + 1;

big\_int\* result = init\_big\_int\_zero(length);

// знак результата совпадает со знаком исходных чисел

result->is\_negative = a->is\_negative;

int ai = a->length - 1; // последний индекс символа-числа a

int bi = b->length - 1; // последний индекс символа-числа b

int ri = result->length - 1; // последний индекс символа-числа результата

int tmp = 0; // храним 0 или 1 -- десяток.

while (ai != -1) { // вычисляем с конца строки, пока строка не закончена

if (bi != -1)

tmp = (a->digits[ai] - '0') + (b->digits[bi] - '0') + tmp;

else

tmp = (a->digits[ai] - '0') + tmp;

if (tmp > 9) { // если после сложения получилось число > 9, то

result->digits[ri] = (char)(tmp % 10 + '0'); // записываем в результат остаток от деления на 10

tmp = 1; // запоминаем десяток

}

else { // иначе просто записываем в результат

result->digits[ri] = (char)(tmp + '0');

tmp = 0;

}

ai--;

ri--;

if (bi != -1)

bi--;

}

if (tmp) // если в tmp остался десяток, добавляем его в результат

result->digits[ri] = '1';

remove\_leading\_zeros(result); // убираем ведущие нули

return result;

}

else { // иначе если числа разных знаков, то выполняем вычитание из одного другое

a->is\_negative = !a->is\_negative; // меняем знак на обратный числу а

big\_int\* result = subtract(a, b); // выполняем вычитание

a->is\_negative = !a->is\_negative; // возвращаем исходный знак числу а

result->is\_negative = !result->is\_negative; // меняем знак на обратный результату

// если получилось число ноль отрицательным, меняем знак

if (result->digits[0] == '0' && result->length == 1 && result->is\_negative == true)

result->is\_negative = false;

return result;

}

}

big\_int\* subtract(big\_int\* a, big\_int\* b) {

if (a->is\_negative == b->is\_negative) { // если числа одинаковых знаков, выполняем вычитание

int cmp = abs\_compare(a, b); // сравниваем числа по модулю

if (cmp == 0) { // если a = b , то возв. ноль

return init\_big\_int\_zero(1);

}

else if (cmp == -1) { // если a < b

big\_int\* result = subtract(b, a); // выполняем вычитание из b а

result->is\_negative = !result->is\_negative; // меняем знак

return result;

}

else { // иначе если a больше b

int length = a->length;

big\_int\* result = init\_big\_int\_zero(length);

result->is\_negative = a->is\_negative;

int ai = a->length - 1; // последний индекс a

int bi = b->length - 1; // последний индекс b

int ri = result->length - 1; // последний индекс результата

int tmp = 0; // храним 0 или 1 - десяток

while (ai != -1) { // вычисляем с конца строки, пока строка не закончена

if (bi != -1)

tmp = (a->digits[ai] - '0') - (b->digits[bi] - '0') - tmp;

else

tmp = (a->digits[ai] - '0') - tmp;

if (tmp < 0) { // если после вычитания получилось отрицательное число, то

result->digits[ri] = (char)(tmp + 10 + '0'); // прибавляем 10 к числу и записываем в результат

tmp = 1; // заимствуем десяток

}

else { // иначе просто записываем в результат

result->digits[ri] = (char)(tmp + '0');

tmp = 0;

}

ai--;

ri--;

if (bi != -1)

bi--;

}

remove\_leading\_zeros(result);

return result;

}

}

else { // иначе если числа разных знаков, то выполняем сложение как в add

a->is\_negative = !a->is\_negative;

big\_int\* result = add(a, b);

a->is\_negative = !a->is\_negative;

result->is\_negative = !result->is\_negative;

if (result->digits[0] == '0' && result->length == 1 && result->is\_negative == true)

result->is\_negative = false;

return result;

}

}

big\_int\* multiply(big\_int\* a, big\_int\* b) {

if (a->length == 1 && a->digits[0] == '0' ||

b->length == 1 && b->digits[0] == '0') {

return init\_big\_int\_zero(1);

}

if (a->length < b->length) { // свап местами

big\_int\* tmp = a;

a = b;

b = tmp;

}

int length = a->length + b->length;

big\_int\* result = init\_big\_int\_zero(1); // результат после суммирования

big\_int\* result\_tmp = init\_big\_int\_zero(length); // результат умножения

int tmp = 0; // храним десятки после умножения

int count = 0; // кол-во сложений после умножения в столбик

int ri = length - 1;

for (int bi = b->length - 1; bi != -1; bi--, ri = length - 1) {

for (int ai = a->length - 1; ai != -1; ai--, ri--) {

tmp = (a->digits[ai] - '0') \* (b->digits[bi] - '0') + tmp;

if (tmp != 9) { // если после умножения получилось число > 9, то

result\_tmp->digits[ri] = (char)(tmp % 10 + '0'); // записываем в результат остаток от деления на 10

tmp /= 10; // запоминаем десятки

}

else { // иначе просто записываем в результат

result\_tmp->digits[ri] = (char)(tmp + '0');

tmp = 0;

}

}

if (tmp) { // если в tmp остался десяток, добавляем его в результат

result\_tmp->digits[ri] = (char)(tmp + '0');

tmp = 0;

}

for (int i = 0; i != count; i++) { // добавляем необходимые нули для последующего сложения

for (int j = 0; j != length - 1; j++)

result\_tmp->digits[j] = result\_tmp->digits[j + 1];

result\_tmp->digits[length - 1] = (char)(tmp + '0');

}

remove\_leading\_zeros(result\_tmp);

big\_int\* ptr = result;

result = add(ptr, result\_tmp); // складываем после умножения

free\_big\_int(ptr);

free\_big\_int(result\_tmp);

result\_tmp = init\_big\_int\_zero(length); // обнуляем result\_tmp

count++;

}

if (a->is\_negative == b->is\_negative) {

result->is\_negative = false;

}

else {

result->is\_negative = true;

}

free\_big\_int(result\_tmp);

return result;

}

big\_int\* int\_to\_big(int a)

{

char\* str = calloc(4500, 4500 \* sizeof(char));

sprintf(str, "%d", a);

big\_int\* result = (big\_int\*)malloc(sizeof(big\_int));

int i;

if (str[0] == '-') {

result->is\_negative = true;

i = 1;

}

else {

result->is\_negative = false;

i = 0;

}

// копирование цифр из строки в digits

int str\_length = (int)strlen(str) - i;

result->length = str\_length;

result->digits = (char\*)malloc(sizeof(char) \* str\_length + 1);

strcpy(result->digits, &str[i]);

remove\_leading\_zeros(result);

return result;

}

big\_int\* powek(big\_int\* base, big\_int\* exponent) { // возведение в степень

if (base->length == 1 && base->digits[0] == '0') // если base == 0, возвращаем 0

return init\_big\_int\_zero(1);

if (base->length == 1 && base->digits[0] == '1' ||

exponent->length == 1 && exponent->digits[0] == '0') // если base == 1 или exp == 0, возвращаем 1

return init\_big\_int("1");

big\_int\* result = init\_big\_int("1");

big\_int\* count = init\_big\_int("1"); // счетчик от 1 до exponent

big\_int\* \_1 = init\_big\_int("1");

while (abs\_compare(count, exponent) < 1) {

big\_int\* tmp = result;

result = multiply(result, base);

free\_big\_int(tmp);

/\*big\_int\* tmp2 = count;\*/

count = add(count, \_1);

/\*free\_big\_int(tmp2);\*/

}

free\_big\_int(count);

free\_big\_int(\_1);

return result;

}

big\_int\* divide(big\_int\* num1, big\_int\* num2) {

big\_int\* result = init\_big\_int\_zero(1);

int i, j;

char\* quotient = calloc(num1->length, sizeof(char));

char\* remainder = calloc(num1->length + 1, sizeof(char));

int qIndex = 0;

int rIndex = 0;

int temp = 0;

bool firstZero = true;

if (num2->length == 1 && num2->digits[0] == '0') {

exit(1);

}

if (num1->length < num2->length) {

result->digits = "0";

result->length = 1;

return result;

}

int divisor = atoi(num2->digits);

for (i = 0; i < num1->length; i++) {

temp = temp \* 10 + (num1->digits[i] - '0');

if (temp < divisor && !firstZero) {

quotient[qIndex++] = '0';

}

else if (temp >= divisor) {

firstZero = false;

quotient[qIndex++] = (char)((temp / divisor) + '0');

temp = temp % divisor;

}

}

if (firstZero) {

quotient[qIndex++] = '0';

}

quotient[qIndex] = '\0';

if (temp == 0) {

result->is\_negative = false;

result->digits = quotient;

result->length = strlen(result->digits);

return result;

}

else {

remainder[rIndex++] = (char)(temp + '0');

}

for (j = i; j < num1->length; j++) {

temp = temp \* 10 + (num1->digits[j] - '0');

remainder[rIndex++] = (char)((temp / divisor) + '0');

temp = temp % divisor;

}

remainder[rIndex] = '\0';

result->is\_negative = false;

result->digits = quotient;

result->length = strlen(result->digits);

return result;

}

big\_int\* divide\_remainder(big\_int\* num1, big\_int\* num2) {

big\_int\* result = init\_big\_int\_zero(1);

int i, j;

char\* quotient = calloc(num1->length, sizeof(char));

char\* remainder = calloc(num1->length + 1, sizeof(char));

int qIndex = 0;

int rIndex = 0;

int temp = 0;

bool firstZero = true;

if (num2->length == 1 && num2->digits[0] == '0') {

exit(1);

}

if (num1->length < num2->length) {

result->digits = "0";

result->length = 1;

return result;

}

int divisor = atoi(num2->digits);

for (i = 0; i < num1->length; i++) {

temp = temp \* 10 + (num1->digits[i] - '0');

if (temp < divisor && !firstZero) {

quotient[qIndex++] = '0';

}

else if (temp >= divisor) {

firstZero = false;

quotient[qIndex++] = (char)((temp / divisor) + '0');

temp = temp % divisor;

}

}

if (firstZero) {

quotient[qIndex++] = '0';

}

quotient[qIndex] = '\0';

if (temp == 0) {

result->is\_negative = false;

result->digits = quotient;

result->length = strlen(result->digits);

return result;

}

else {

remainder[rIndex++] = (char)(temp + '0');

}

for (j = i; j < num1->length; j++) {

temp = temp \* 10 + (num1->digits[j] - '0');

remainder[rIndex++] = (char)((temp / divisor) + '0');

temp = temp % divisor;

}

remainder[rIndex] = '\0';

result->is\_negative = false;

result->digits = quotient;

result->length = strlen(result->digits);

big\_int\* remainder2 = init\_big\_int(remainder);

return remainder2;

}

# Приложение В

Листинг программы “Arifmetika.h”

#pragma once

#include <stdbool.h>

typedef struct {

char\* digits;

int length;

bool is\_negative;

char trash[3];

} big\_int;

void free\_big\_int(big\_int\* number); // очистка памяти

void remove\_leading\_zeros(big\_int\* number); // удаление нулей

big\_int\* init\_big\_int\_zero(int length); // инициализация нуля

int abs\_compare(big\_int\* a, big\_int\* b); // сравнение по модулю

big\_int\* add(big\_int\* a, big\_int\* b); // сложение

big\_int\* subtract(big\_int\* a, big\_int\* b); // вычитаение

big\_int\* multiply(big\_int\* a, big\_int\* b); // умножение

big\_int\* init\_big\_int(const char\* str); // инициализация строки в число

big\_int\* int\_to\_big(int a); // число в большое число

big\_int\* powek(big\_int\* base, big\_int\* exponent); // степень

big\_int\* divide(big\_int\* num1, big\_int\* num2); // просто деление

big\_int\* divide\_remainder(big\_int\* num1, big\_int\* num2); // деление с остатком