Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

Лабораторная работа №3

Асимметричное шифрование на основе алгоритма RSA

Выполнил
Глазков Андрей
студент 4 курса
группы 06-952

Казань – 2023

Цель работы

- Разработка программы, реализующей процедуру генерации открытого и закрытого ключей заданной длины L;
- Программно реализовать процедуры шифрования и дешифрования согласно алгоритму RSA;
- Проверить работоспособность алгоритма RSA в случае, когда одно из чисел p и q не является простым.
- Реализовать атаку на алгоритм RSA с использованием ρ-эвристики Полларда.
- Проанализировать ка различие битовой длинны влияет на сложность факторизации

Ход работы:

1. Генерация простых чисел p и q, расчет пары открытого и закрытого ключей.

   Для генерации генерации всех необходимых параметров разработали класс `class KeyGenerator:`

```
class KeyGenerator {

   public:

       KeyGenerator(int L, bool task_3 = false) {

           int n = L / 2;



           gmp_randstate_t state;

           gmp_randinit_default(state);

           gmp_randseed_ui(state,
std::chrono::system_clock::now().time_since_epoch().count());



           mpz_urandomb(p_.get_mpz_t(), state, n);

           mpz_urandomb(q_.get_mpz_t(), state, n);



           while (!mpz_probab_prime_p(p_.get_mpz_t(), 100))

               mpz_urandomb(p_.get_mpz_t(), state, n);



           if (!task_3) {

               while (!mpz_probab_prime_p(q_.get_mpz_t(), 100))

                   mpz_urandomb(q_.get_mpz_t(), state, n);
```

```
            } else {
                while (mpz_probab_prime_p(q_.get_mpz_t(), 100))
                    mpz_urandomb(q_.get_mpz_t(), state, n);
            }
            n_ = p_ * q_;
            phi_ = (q_ - 1) * (p_ - 1);

            mpz_urandomb(e_.get_mpz_t(), state, n);
            while (!mpz_probab_prime_p(e_.get_mpz_t(), 100)) {
                mpz_urandomb(e_.get_mpz_t(), state, n);
            }
            // choose_e();
            mpz_invert(d_.get_mpz_t(), e_.get_mpz_t(), phi_.get_mpz_t());
            gmp_randclear(state);
        }

        ~KeyGenerator() = default;

        mpz_class getP() const      { return p_; }
        mpz_class getQ() const      { return q_; }
        mpz_class getN() const      { return n_; }
        mpz_class getPhi() const    { return phi_; }
        mpz_class getE() const      { return e_; }
        mpz_class getD() const      { return d_; }

        void encrypt_str(const std::string &s_pl_file, const std::string&
pub_file);
        void decrypt_str(const std::string &s_enc_file, const std::string&
pvt_file);

    private:
        void choose_e() {
            mpz_class e {65337};
            mpz_class gcd;
```

```
            while ((mpz_gcd(gcd.get_mpz_t(), phi_.get_mpz_t(), e.get_mpz_t())),
gcd != 1){
                e += 2;
            }
            e_ = e;
        }


        mpz_class convertStingToNum(std::string& in) {
            mpz_class tmp;
            mpz_class tmp_data;
            for (auto&& c : in) {
                if (c == 0) {
                    break;
                }
                tmp= c;
                tmp += 100;
                tmp_data = tmp_data * 1000 + tmp;
            }
            return tmp_data;
        }


    private:
        mpz_class p_;
        mpz_class q_;
        mpz_class phi_;


        mpz_class n_;
        mpz_class e_;
        mpz_class d_;
};
```

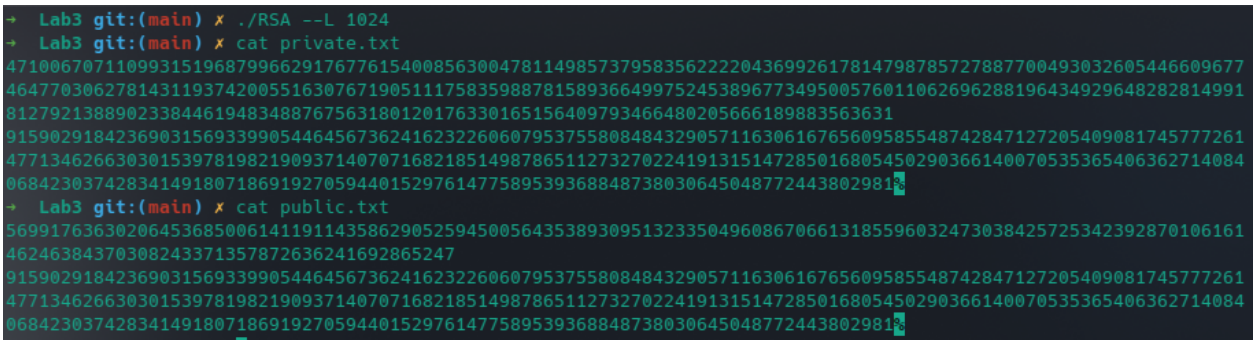


Рис.1 Результат работы программы (генерация приватного и публичного ключа для L = 1024)

2. Реализация шифрования и дешифрования согласно алгоритму RSA.

Для этого разработали два метода:

```
void encrypt_str(const std::string &s_pl_file, const std::string& s_pub_file);

void decrypt_str(const std::string &s_enc_file, const std::string &s_pvt_file);

void KeyGenerator::encrypt_str(const std::string &s_pl_file, const std::string&
s_pub_file) {
    std::ofstream enc("encrypted.txt");
    std::ifstream pl_file(s_pl_file, std::ios::binary | std::ios::ate);
    std::ifstream pub_file(s_pub_file);
    if (!pl_file.is_open() || !pub_file.is_open()) {
        throw std::runtime_error("Error: failed open plain text file or pubfile");
    }

    pub_file >> e_ >> n_;
    int f_size = pl_file.tellg();
    --f_size;
    char bytes[4];
    std::string tmp;
    mpz_class M, enc_data;
    int i,k;
    k = 0;
    pl_file.seekg(0);
    while (k <= f_size/4) {
        i = 0;
        if (k == f_size/4) {
            pl_file.read(bytes, f_size%4);
        } else {
            pl_file.read(bytes, 4);
        }
        while (i < 4) {
            tmp += bytes[i++];
        }
        ++k;
        pl_file.seekg(4 * k);
        for (int j = 0; j < 4; ++j) {
            bytes[j] = 32;
        }
        M = convertStingToNum(tmp);
        mpz_powm(enc_data.get_mpz_t(), M.get_mpz_t(), e_.get_mpz_t(), n_.get_mpz_t());
        enc << enc_data << " ";
        tmp = "";
    }
    enc.close();
    pl_file.close();
    pub_file.close();
}

void KeyGenerator::decrypt_str(const std::string &s_enc_file, const std::string
&s_pvt_file) {
```

```
    std::ofstream dec("decrypted.txt");
    std::ifstream enc_file(s_enc_file);
    std::ifstream pvt_file(s_pvt_file);
    if (!enc_file.is_open() || !pvt_file.is_open()){
        throw std::runtime_error("Error: failed open file enc or pvt_file");
    }

    mpz_class tmp, num;
    pvt_file >> d_ >> n_;
    int t, i;
    std::string dec_text;
    while (enc_file >> tmp) {
        mpz_powm(num.get_mpz_t(), tmp.get_mpz_t(), d_.get_mpz_t(), n_.get_mpz_t());
        i = 1;
        while (i < 5) {
            tmp = num / std::pow(10, 12 - 3 * i);
            num = num % std::pow(10, 12 - 3 * i);
            tmp -= 100;
            t = mpz_get_ui(tmp.get_mpz_t());
            dec_text += static_cast<char>(t);
            i++;
        }
        dec << dec_text;
        dec_text = "";
    }
    dec.close();
    enc_file.close();
    pvt_file.close();
}
```

```
→  Lab3 git:(main) ✗ cat text
The class template std::optional manages an optional contained value, i.e. a value that may or may not be present
.
A common use case for optional is the return value of a function that may fail. As opposed to other approaches, s
uch as std::pair<T,bool>, optional handles expensive-to-construct objects well and is more readable, as the inten
t is expressed explicitly.
Any instance of optional<T> at any given point in time either contains a value or does not contain a value.
If an optional<T> contains a value, the value is guaranteed to be allocated as part of the optional object footpr
int, i.e. no dynamic memory allocation ever takes place. Thus, an optional object models an object, not a pointer
, even though operator*() and operator->() are defined.
When an object of type optional<T> is contextually converted to bool, the conversion returns true if the object c
ontains a value and false if it does not contain a value.
The optional object contains a value in the following conditions:
    The object is initialized with/assigned from a value of type T or another optional that contains a value.
The object does not contain a value in the following conditions:
    The object is default-initialized.
    The object is initialized with/assigned from a value of type std::nullopt_t or an optional object that does n
ot contain a value.
    The member function reset() is called.
```

Lab3 : zsh — Konsole

New Tab | Split View | Copy | Paste | Find

```
→  Lab3 git:(main) ✗ ./RSA --task 2 --file text --key public.txt
→  Lab3 git:(main) ✗ cat encrypted.txt
947938242578418739172406686378001456602873270160965027465789079742924385931198255666427294426583199639994968828496813985620756577524809336799941517018112054496061748113952190886090853959261395734520745188632255857877667727212949535756348798206462949948705717007448884820908452874633308645134470272358117469 420177686988355712611569932229374784698630958352257169700885385747657461175379659296721727826360136603987846437538160307100366760547297978885597789544852872556627016978390807327222582742079400933394725731841240817171581332656017174879918028888724330469833187003242091831273352429789202398129058447999762494563315879769258499739724917035098124404152327349208695810613412907028018933152605484518547097059743472893304496473825814507411116177380349797344818170560547815852481716994642981887031961844677544974686557702410949735773247765100547570458444100251190311851732205821822935575566816249048238076390683233407344580142546034928010152350541559382690133931431642494656663260304530354119950422984270585645756235234345422418965000601522198353063481302636453871038664903724696323935453193269450251888489840440542850815039992486411330816086593266881978619641631009908781139836893651096115141650938044203417052600233242206354829485 351219545457715273889674631073888351488868538402930599131322296794647009115599899009409064305914398251752985663030554363537102351512110205440179170694868860448311738769770242230635637630071302006605092829057942920363986611228277198212992969530404745427603203216297482370018255657847264059540904323388210818837136690592120911562111211357599624889174493032829944845047423442284262488341039449357648436581818174721814249766231161233288087588932085503323090793849096945482198765371780608331357223023090238010913016977070419438557075873391984513775964531703276977849052930662975798285238492704497952674799428568855708167815469296396668471766728486071899967545647300784933138303538294664758 0
```

Lab3 : zsh — Konsole

New Tab | Split View | Copy | Paste | Find

```
→  Lab3 git:(main) ✗ cat decrypted.txt
The class template std::optional manages an optional contained value, i.e. a value that may or may not be present
.
A common use case for optional is the return value of a function that may fail. As opposed to other approaches, s
uch as std::pair<T,bool>, optional handles expensive-to-construct objects well and is more readable, as the inten
t is expressed explicitly.
Any instance of optional<T> at any given point in time either contains a value or does not contain a value.
If an optional<T> contains a value, the value is guaranteed to be allocated as part of the optional object footpr
int, i.e. no dynamic memory allocation ever takes place. Thus, an optional object models an object, not a pointer
, even though operator*() and operator->() are defined.
When an object of type optional<T> is contextually converted to bool, the conversion returns true if the object c
ontains a value and false if it does not contain a value.
The optional object contains a value in the following conditions:
    The object is initialized with/assigned from a value of type T or another optional that contains a value.
The object does not contain a value in the following conditions:
    The object is default-initialized.
    The object is initialized with/assigned from a value of type std::nullopt_t or an optional object that does n
ot contain a value.
    The member function reset() is called.
```

Рис.2 Результат шифрования и дешифрования записанные в файлы

3. Если одно из чисел p или q будет составным, то декодирование будет невозможным.

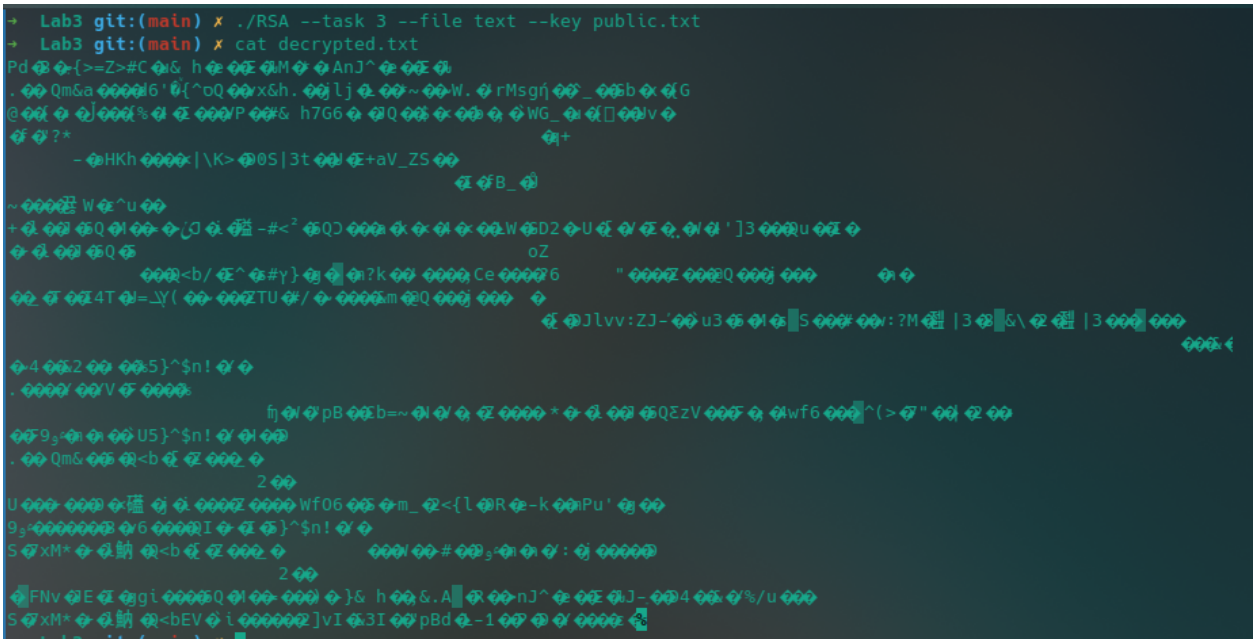


Рис.3 Результат дешифровки если одно из чисел не простое

4. Реализация атаки на алгоритм RSA с использованием ρ-эвристики Полларда.

```cpp
mpz_class pollard_rho(mpz_class n) {
    mpz_class x = 2, y = 2, d = 1;
    while (d == 1) {
        x = (x * x + 1) % n;
        y = (y * y + 1) % n;
        y = (y * y + 1) % n;
        d = gcd(abs(x - y), n);
    }
    if (d == n) {
        return 0;
    }
    return d;
}
```

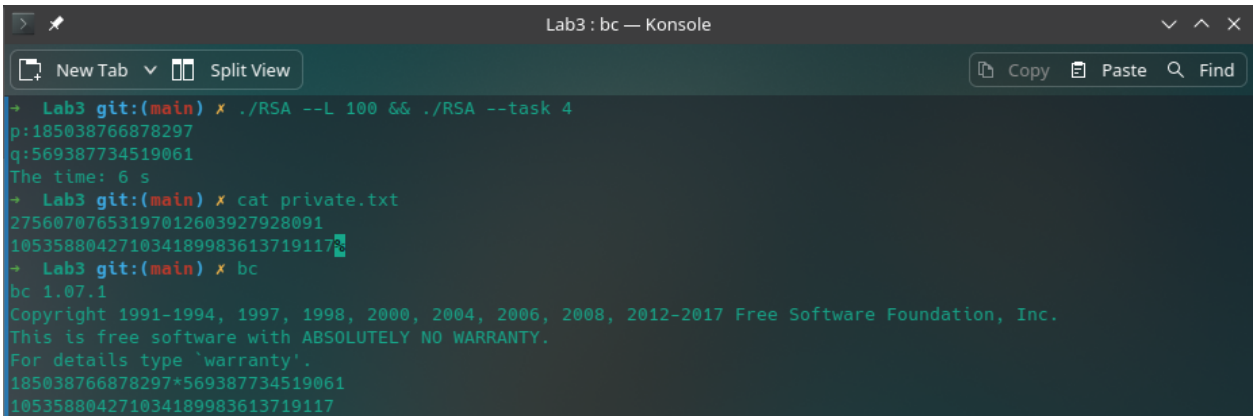


Рис.4 Результат атаки на алгоритм RSA с использованием ρ-эвристики Полларда

Построили график зависимости времени факторизации ключа от длины ключа.

L,бит	t,c
70	0.06
75	0.08
80	0.21
85	0.45
90	0.95
95	2
100	5.9
105	14
110	43.3
115	90.8
120	383.5
130	1600

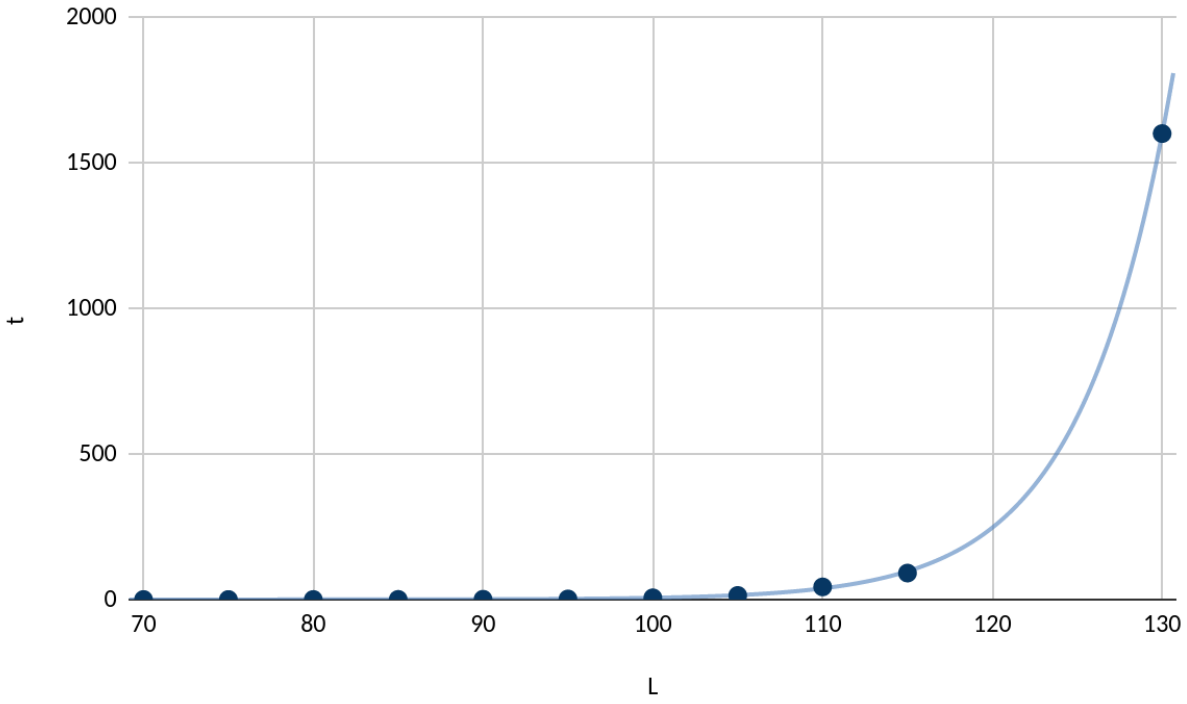


Рис. 5 График зависимости времени факторизации ключа от его длины

Как видно по результатам, время атаки с увеличением длины ключа тоже увеличивается. Но также в ходе атак были замечены случаи, когда при одинаковых длинах ключа длительности атак заметно различаются, что связано с тем, что при атаке ищется минимальное из чисел p или q.

5. Выяснили что с различие битовой длины чисел p и q увеличивает криптографическую стойкость.

Построили график зависимости времени факторизации числа n = p·q от r для r = 0,25 … 0,5

r	t, мсек.
0,25	15
0,275	20
0,3	52
0,325	72
0,35	234
0,375	266
0,4	1552
0,425	2305
0,45	4992
0,475	11132
0,5	14133

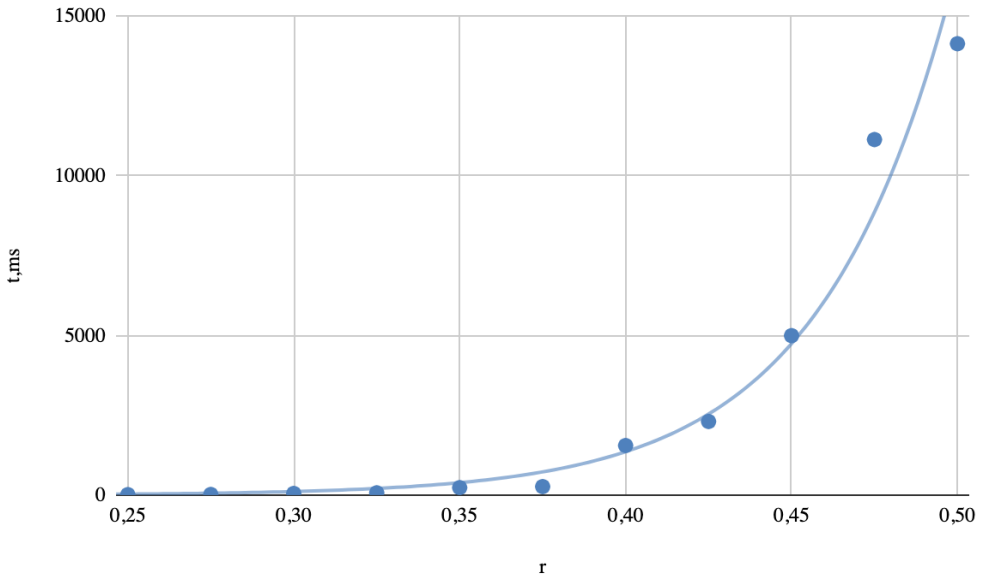
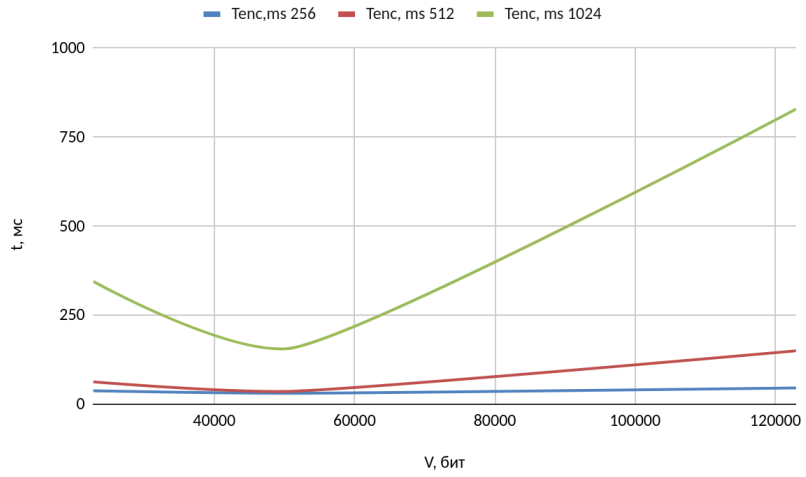


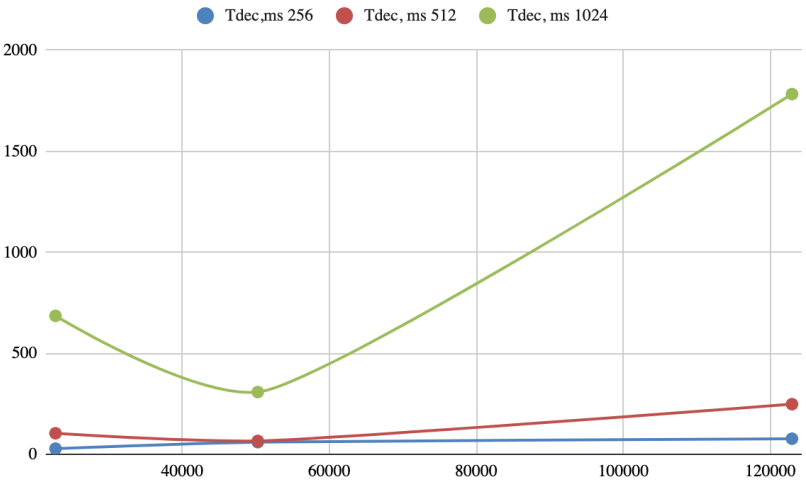
Рис. 6 График зависимости времени факторизации ключа от различия битовой длины p и q

На графике можно видеть экспоненциальную зависимость. Это говорит о том, что с уменьшением расстояния длин p и q увеличивается время атаки. Соответственно для шифрования лучше использовать p и q одинаковой длины.

6. Исследование зависимости длительности Tenc (V) шифрования и длительности Tdec (V) дешифрования от битовой длины V сообщения при трех различных длинах ключа L = {256; 512; 1024} бит;

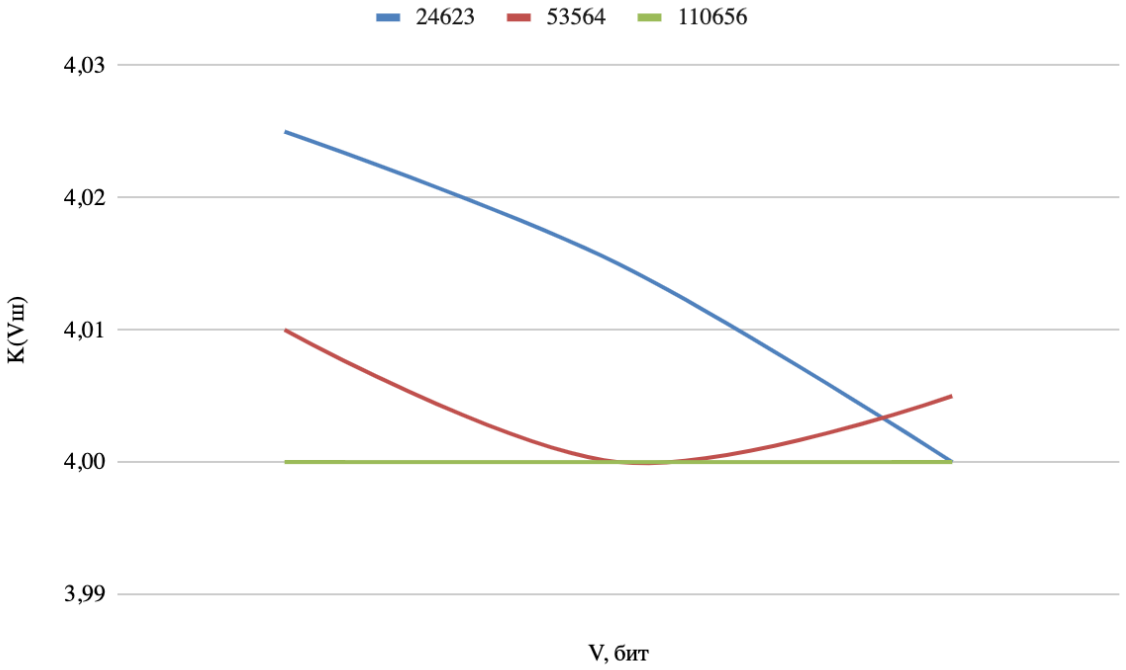
V, бит	Tenc, ms 256	Tenc, ms 512	Tenc, ms 1024	V, бит	Tdec, ms 256	Tdec, ms 512	Tdec, ms 1024
22760	37	62	345	22760	28	104	685
50272	30	35	156	50272	60	66	308
122944	45	149	829	122944	77	248	1783





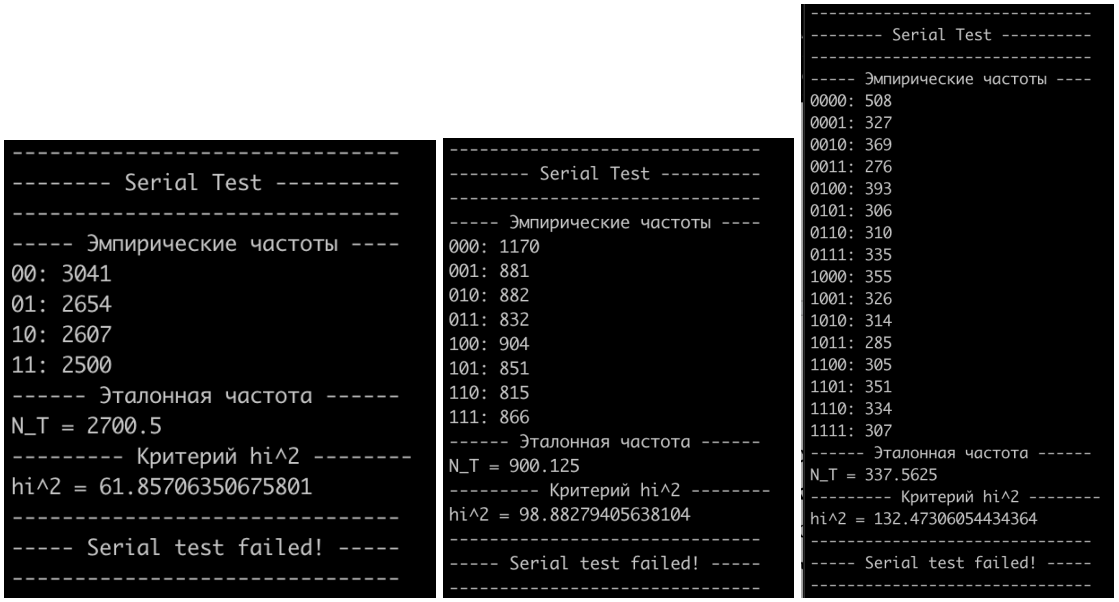
Как можем видеть из графиков, при шифровании линии функции для различных длин ключей лежат на графике рядом, при дешифровании же функции с увеличением длины сообщения расходятся, что может говорить о том, что время дешифровки с увеличением длины ключа и длины сообщения увеличивается сильнее, чем при шифровании.

7. Исследование зависимость коэффициента разрастания шифрограммы Kv=Vш/V (где Vш – битовая длина шифрограммы) от битовой длины сообщения при трех различных длинах ключа L = {256; 512; 1024} бит;

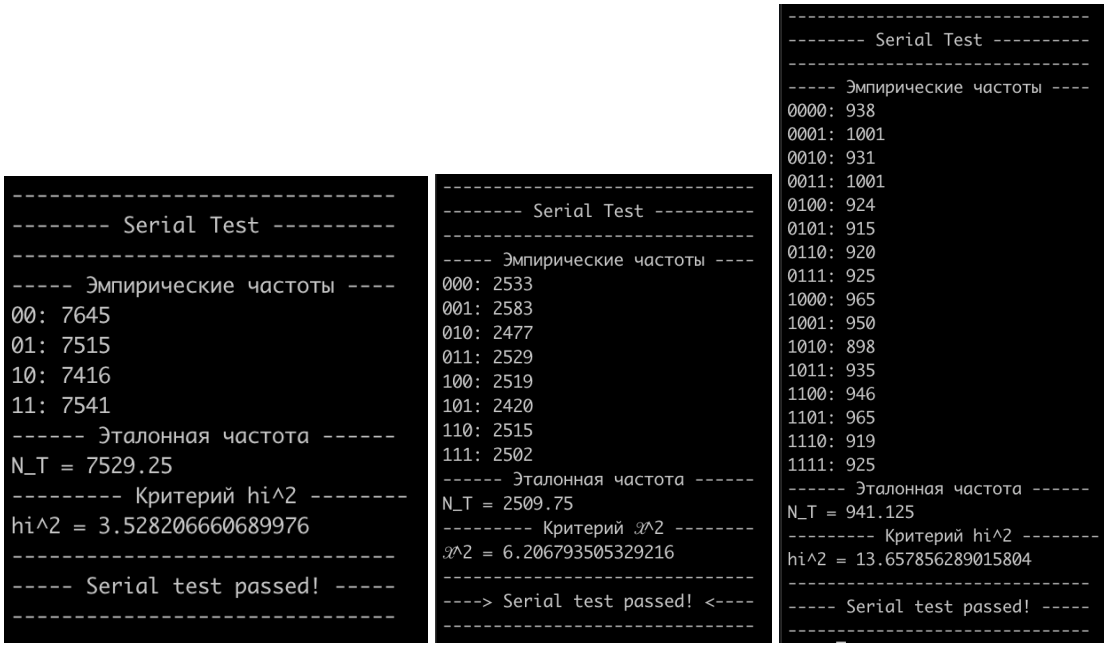


Исходя из графика, можно сделать вывод, что при увеличении длины сообщения коэффициент расширения для различных длин ключей становится приблизительно одинаковым и равным 4.

8. Тестирование битового потока шифрограммы на равномерность при двух значениях открытой экспоненты: e = 3 и e = 65537.



Для e = 3 все три теста провалились.



Для e = 65537 все три теста прошли статистический тест.

Это говорить о том, что шифрование открытой экспонентной, равной 3, не является безопасным, с точки зрения криптографии. При шифровании открытой экспонентной e=65537 видим, что шифрограмма проходит тест для серий 2,3 и 4 очень хорошо. Исходя из результатов, можно сделать вывод, что при шифровании RSA, не стоит использовать открытую экспоненту, равную 3, а предпочитать e=65537.

Вывод:

В ходе данной лабораторной работы были выполнены все поставленные задачи.
Исходя из результатов данной лабораторной работы можно сделать следующие выводы:

- Алгоритм RSA не может однозначно декодировать, если одно из чисел p и q составное. Объяснить это можно тем, что вычисление обратного модуля ищет простые множители с использованием Евклидова алгоритма, и если n не является произведением двух простых чисел, то он даст другой ответ. Если посмотреть на теорему Эйлера, можно увидеть, что принятие неправильного значения для $\phi(n)$ приведет к остатку $\neq 1 \bmod n$ при дешифровании;
- Время атаки с увеличением длины ключа тоже увеличивается. Но также в ходе атак были замещены случаи, когда при одинаковых длинах ключа длительности атак заметно различаются, объясняется это природой эвристики.
- С уменьшением расстояния длин p и q увеличивается время атаки. То есть следует выбирать p и q одинаковых длин, т.к. уменьшение длины одной из простых чисел ведет к ускорению атаки.
- Время дешифровки с увеличением длины ключа и длины сообщения увеличивается сильнее, чем при шифровании;
- При увеличении длины сообщения коэффициент расширения для различных длин ключей становится приблизительно одинаковым;
- Шифрование открытой экспонентной, равной 3, не является безопасным, с точки зрения криптографии и стоит использовать открытую экспоненту, равную e=65537.