



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 p -эвристики Полларда.
- Проанализировать как 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_;

};

```

```
→ Lab3 git:(main) x ./RSA --L 1024
→ Lab3 git:(main) x cat private.txt
47100670711099315196879966291767761540085630047811498573795835622220436992617814798785727887700493032605446609677
46477030627814311937420055163076719051117583598878158936649975245389677349500576011062696288196434929648282814991
812792138890233844619483488767563180120176330165156409793466480205666189883563631
91590291842369031569339905446456736241623226060795375580848432905711630616765609585548742847127205409081745777261
47713462663030153978198219093714070716821851498786511273270224191315147285016805450290366140070535365406362714084
0684230374283414918071869192705944015297614775895393688487380306450487724438029812
→ Lab3 git:(main) x cat public.txt
56991763630206453685006141191143586290525945005643538930951323350496086706613185596032473038425725342392870106161
46246384370308243371357872636241692865247
91590291842369031569339905446456736241623226060795375580848432905711630616765609585548742847127205409081745777261
47713462663030153978198219093714070716821851498786511273270224191315147285016805450290366140070535365406362714084
0684230374283414918071869192705944015297614775895393688487380306450487724438029812
```

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 = convertStringToNum(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) x 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) x ./RSA --task 2 --file text --key public.txt
→ Lab3 git:(main) x cat encrypted.txt
94793824257841873917240668637800145660287327016096502746578907974292438593119825566642729442658319963999496882849
68139856207565775248093367999415170181120544960617481139521908860908539592613957345207451886322558578776677272129
49535756348798206462949948705717007448884820908452874633308645134470272358117469 42017768698835571261156993222937
47846986309583522571697008853857476574611753796592967217278263601366039878464375381603071003667605472979788855977
89544852872556627016978390807327222582742079400933394725731841240817171581332656017174879918028888724330469833187
00324209183127335242978920239812905844799976249456 33158797692584997397249170350981244041523273492086958106134129
0702801893315260548451854709705974347289330449647382581450741116177380349797344818170560547815852481716994642981
8870319618446775449746865577024109497357732477651005475704584410025119031185173220582182293557556681624904823807
63906832334073445801 42546034928010152350541559382690133931431642494656663260304530354119950422984270585645756235
23434542241896500060152219835306348130263645387103866490372469632393545319326945025188848984044054285081503999248
6411330816086593266881978619641631009908781139836893651096115141650938044203417052600233242206354829485 351219545
4577152738896746310738883514888685384029305991313222967946470091155989900940906430591439825175298566303055436353
71023515121102054401791706948688604483117387697702422306356376300713020066050928290579429203639866112282771982129
9296953040474542760320321629748237001825565784726405954090432338821081883 713669059212091156211121135759962488917
44930328299448450474234422842624883410394493576484365818181747218142497662311612332880875889320855033230907938490
96945482198765371780608331357223023090238010913016977070419438557075873391984513775964531703276977849052930662975
79828523849270449795267479942856885570816 78154692963966684717667284860718999675456473007849331383035382946647580

```

```

> ✎ Lab3 : zsh — Konsole
New Tab Split View Copy Paste Find
→ Lab3 git:(main) x 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.

```


+ Lab3 git:(main) x ./RSA --task 3 --file text --key public.txt
+ Lab3 git:(main) x cat decrypted.txt
Pd 0 {>=Z>#C  h  AnJ^
. Qm&a6' {^oQ x&h. lj ~W. rMsrgr_ b  G
@  %  WP  h7G6  Q  WG_  v
? *
- HKh | \K> 0S| 3t  +aV_ZS
B_
~  W ^u
+  Q  -#<^  Q  W  D2  U  '  ]3  u
  Q
  Q<b/  'y}  ?k  Ce6  "  Q  "
  4T  =Y(  TU  /  m  Q  Q
  Jlvv:ZJ-'  u3  M  S  :?M  |3  &\  |3
4  2  ^$n!
.  V
  pB  =~  *  Q  QzV  w6  ^(>'  Q
9_  U5}^$n!
.  Qm&  <b  Q
2
U  Wf06  m_  <{l  R  -k  Pu'
9_  6  I  ^$n!
S  xM*  <b  Q  Q  #  _  :  Q
2
FNV  E  gi  Q  Q  }&  h  &.A  nJ^  Q  J-  4  %/u
S  xM*  <bEV  i  i  2  vI  3I  pBd  -1  Q  Q

```

Рис.3 Результат дешифровки если одно из чисел не простое

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

```
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;  
}
```



The screenshot shows a terminal window titled "Lab3: bc — Konsole". The terminal displays the output of a script that runs the pollard_rho function on a large number. The output shows the prime factors p and q, the execution time, and the contents of the private key file. The terminal also shows the output of the 'bc' command, which displays the version and copyright information.

```
Lab3 git:(main) x ./RSA --L 100 && ./RSA --task 4  
p:185038766878297  
q:569387734519061  
The time: 6 s  
Lab3 git:(main) x cat private.txt  
27560707653197012603927928091  
1053588042710341899836137191173  
Lab3 git:(main) x bc  
bc 1.07.1  
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.  
This is free software with ABSOLUTELY NO WARRANTY.  
For details type `warranty'.  
185038766878297*569387734519061  
105358804271034189983613719117
```

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

Построили график зависимости времени факторизации ключа от длины ключа.

L, бит	t, с
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

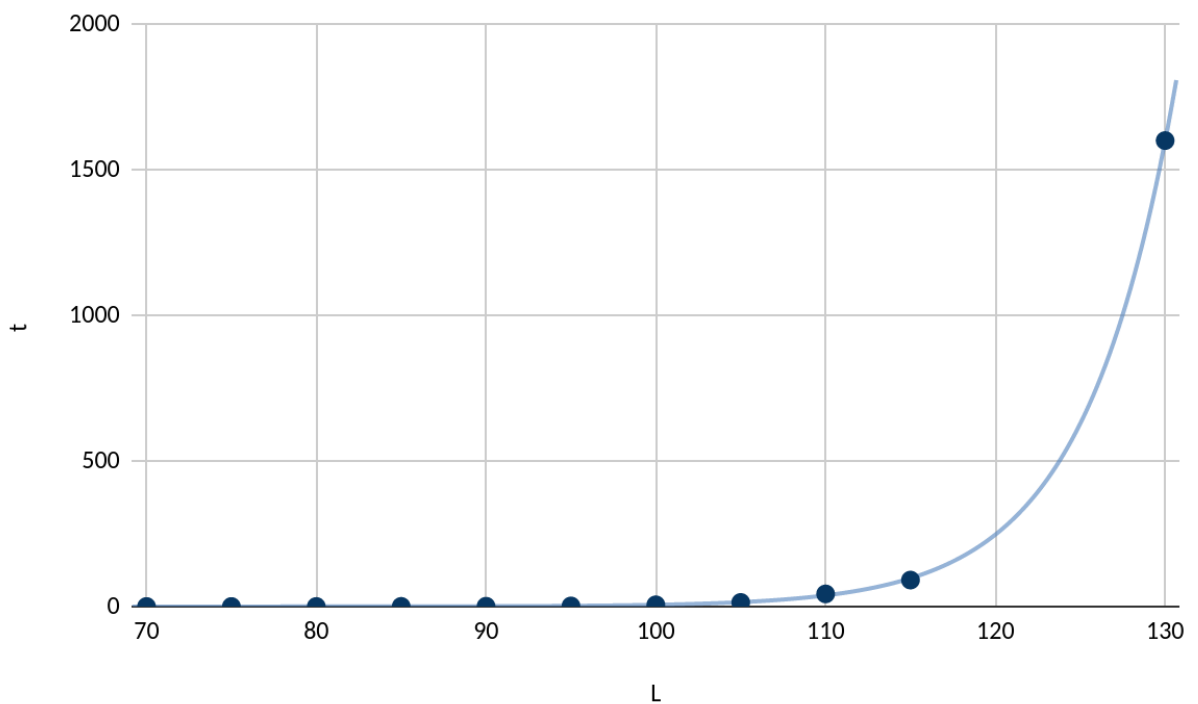


Рис. 5 График зависимости времени факторизации ключа от его длины

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 разлчие битовой длины чисел p и q увеличивает криптографическую стойкость.

Построили график зависимости времени факторизации числа $n = p \cdot q$ от r для $r = 0,25 \dots 0,5$

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

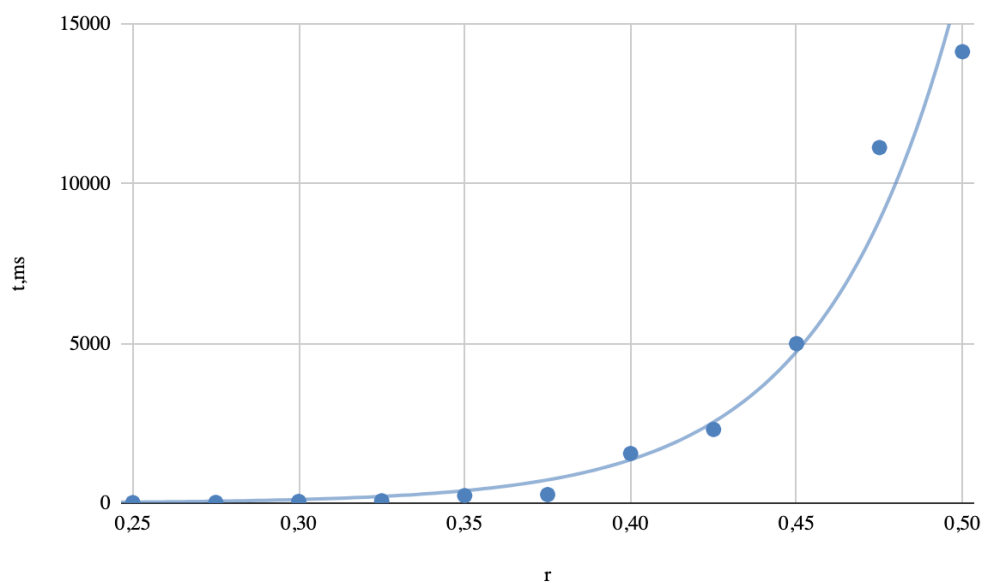
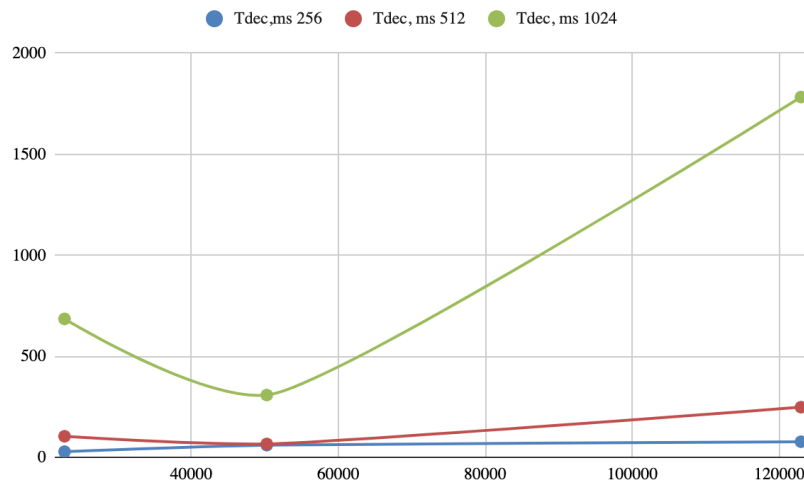
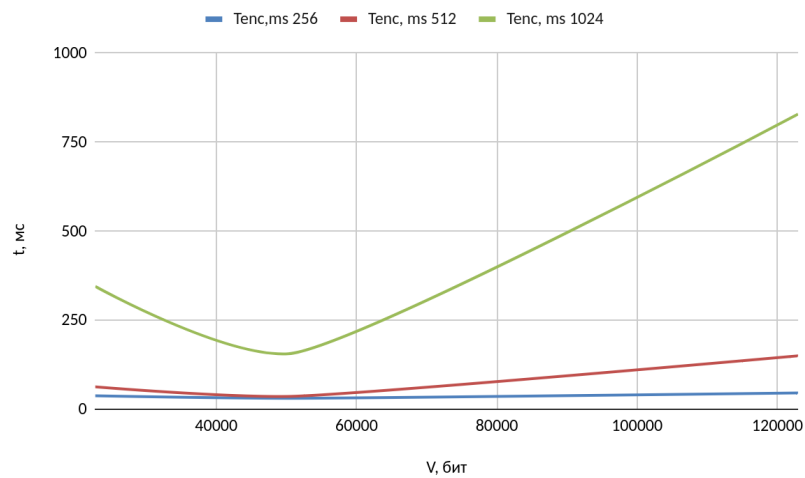


Рис. 6 График зависимости времени факторизации ключа от различия битовой длины p и q

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.

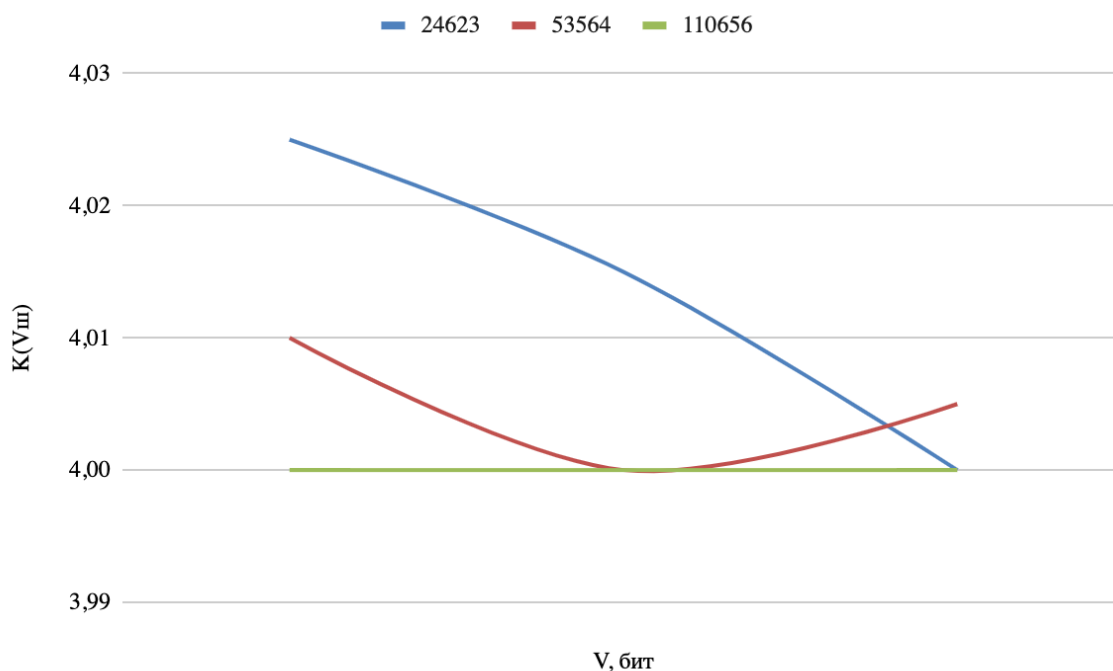
6. Исследование зависимости длительности $T_{enc}(V)$ шифрования и длительности $T_{dec}(V)$ дешифрования от битовой длины V сообщения при трех различных длинах ключа $L = \{256; 512; 1024\}$ бит;

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 $K_v = V_{ш}/V$ (где $V_{ш}$ – 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 $e = 65537$.

```

----- Serial Test -----
----- Эмпирические частоты ----
00: 3041
01: 2654
10: 2607
11: 2500
----- Эталонная частота -----
N_T = 2700.5
----- Критерий  $hi^2$  -----
 $hi^2 = 61.85706350675801$ 
----- Serial test failed! -----

```

```

----- Serial Test -----
----- Эмпирические частоты ----
000: 1170
001: 881
010: 882
011: 832
100: 904
101: 851
110: 815
111: 866
----- Эталонная частота -----
N_T = 900.125
----- Критерий  $hi^2$  -----
 $hi^2 = 98.88279405638104$ 
----- Serial test failed! -----

```

```

----- Serial Test -----
----- Эмпирические частоты ----
0000: 508
0001: 327
0010: 369
0011: 276
0100: 393
0101: 306
0110: 310
0111: 335
1000: 355
1001: 326
1010: 314
1011: 285
1100: 305
1101: 351
1110: 334
1111: 307
----- Эталонная частота -----
N_T = 337.5625
----- Критерий  $hi^2$  -----
 $hi^2 = 132.47306054434364$ 
----- Serial test failed! -----

```

Для $e = 3$ все три теста провалились.

```

----- Serial Test -----
----- Эмпирические частоты ----
00: 7645
01: 7515
10: 7416
11: 7541
----- Эталонная частота -----
N_T = 7529.25
----- Критерий  $hi^2$  -----
 $hi^2 = 3.528206660689976$ 
----- Serial test passed! -----

```

```

----- Serial Test -----
----- Эмпирические частоты ----
000: 2533
001: 2583
010: 2477
011: 2529
100: 2519
101: 2420
110: 2515
111: 2502
----- Эталонная частота -----
N_T = 2509.75
----- Критерий  $hi^2$  -----
 $hi^2 = 6.206793505329216$ 
-----> Serial test passed! <-----

```

```

----- Serial Test -----
----- Эмпирические частоты ----
0000: 938
0001: 1001
0010: 931
0011: 1001
0100: 924
0101: 915
0110: 920
0111: 925
1000: 965
1001: 950
1010: 898
1011: 935
1100: 946
1101: 965
1110: 919
1111: 925
----- Эталонная частота -----
N_T = 941.125
----- Критерий  $hi^2$  -----
 $hi^2 = 13.657856289015804$ 
----- Serial test passed! -----

```

Для $e = 65537$ все три теста прошли статистический тест.

Это говорит 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 замечены 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 экспонентой, равной 3, не является безопасным, с точки зрения криптографии и стоит использовать открытую экспоненту, равную $e=65537$.