Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»
Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 5

ОТЧЕТ
по лабораторной работе №2

Тема: «Алгоритм RSA»

Выполнил: студент группы РИС-19-1б
Миннахметов Э.Ю. _____
Проверил: доцент кафедры ИТАС
Шереметьев В. Г. _____
Дата _____

Пермь, 2021

ЦЕЛЬ РАБОТЫ

Получить практические навыки по использованию ассиметричных алгоритмов шифрования, на примере использования алгоритма RSA.

ЗАДАНИЕ

Вариант №2. Выполнить шифрование текста методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей шестнадцати.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Криптосистема RSA

Алгоритм RSA (R.Rivest, A.Shamir, L.Adleman) был предложен еще в 1977 году. С тех пор он весьма упорно противостоит различным атакам, и сейчас является самым распространенным криптоалгоритмом в мире. Он входит во многие криптографические стандарты, используется во многих приложениях и секретных протоколах (включая PEM, S-HTTP и SSL).

Основные принципы работы RSA

Сначала пара математических определений. Целое число называют простым, если оно делится нацело только на единицу и на само себя, иначе его называют составным. Два целых числа называют взаимно простым, если их наибольший общий делитель (НОД) равен 1.

Алгоритм работы RSA таков. Сначала надо получить открытый и секретный ключи:

1. Выбираются два простых числа p и q
2. Вычисляется их произведение n(=p*q)
3. Выбирается произвольное число e (e<n), такое, что НОД(e,(p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).
4. Методом Евклида решается в целых числах уравнение e*d+(p-1)(q-1)*y=1. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d,y), каждая из которых является решением уравнения в целых числах.
5. Два числа (e,n) – публикуются как открытый ключ.
6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e,n).

Как же производится собственно шифрование с помощью этих чисел:

1. Отправитель разбивает свое сообщение на блоки, равные k=[log2(n)] бит, где квадратные скобки обозначают взятие целой части от дробного числа.
2. Подобный блок, как Вы знаете, может быть интерпретирован как число из диапазона (0;2k-1). Для каждого такого числа (назовем его mi) вычисляется выражение ci=((mi)e)mod n. Блоки ci и есть зашифрованное сообщение Их можно спокойно передавать по открытому каналу, поскольку операция возведения в степень по модулю простого числа, является необратимой математической задачей. Обратная ей задача носит название «логарифмирование в конечном поле» и является на несколько порядков более сложной задачей. То есть даже если злоумышленник знает числа e и n, то по ci прочесть исходные сообщения mi он не может никак, кроме как полным перебором mi.

А вот на приемной стороне процесс дешифрования все же возможен, и поможет нам в этом хранимое в секрете число d. Достаточно давно была доказана теорема Эйлера, частный случай которой утверждает, что если число n представимо в виде двух простых чисел p и q, то для любого x имеет место равенство (x(p-1)(q-1))mod n = 1. Для дешифрования RSA-сообщений воспользуемся этой формулой. Возведем обе ее части в степень (-y) : (x(-y)(p-1)(q-1))mod n = 1(-y) = 1. Теперь умножим обе ее части на x : (x(-y)(p-1)(q-1)+1)mod n = 1*x = x.

А теперь вспомним как мы создавали открытый и закрытый ключи. Мы подбирали с помощью алгоритма Евклида d такое, что e*d+(p-1)(q-1)*y=1, то есть e*d=(-y)(p-1)(q-1)+1. А следовательно в последнем выражении предыдущего абзаца мы можем заменить показатель степени на число (e*d). Получаем (xe*d)mod n = x. То есть для того чтобы прочесть сообщение ci=((mi)e)mod n достаточно возвести его в степень d по модулю m : ((ci)d)mod n = ((mi)e*d)mod n = mi.

На самом деле операции возведения в степень больших чисел достаточно трудоемки для современных процессоров, даже если они производятся по оптимизированным по времени алгоритмам. Поэтому обычно весь текст сообщения кодируется обычным блочным шифром (намного более быстрым), но с использованием ключа сеанса, а вот сам ключ сеанса шифруется как раз асимметричным алгоритмом с помощью открытого ключа получателя и помещается в начало файла.

В 1990-х годах с помощью распределенных вычислений через Internet предпринимались удачные попытки факторизовать некоторые произвольные большие числа максимальное факторизованное число имело 140 десятичных разрядов (1999 год), на что ушло около 2000 MY (Mips/Year – годовая работа компьютера мощностью в миллион целочисленных операций в секунду). Учитывая это, сейчас специалисты

(Лаборатория RSA, www. rsa.com / rsalabs) рекомендуют использовать минимальную длину ключа n, не менее чем 768 бит (~230 десятичных рзрядов) для малосекретной информации, 1024 бит для обычной и 2048 для особо секретной. Использующаяся в старых продуктах длина ключа в 512 бит (~160 разрядов) уже под угрозой взлома. Известно, что RSA имеет низкую криптостойкость при шифровании коротких блоков. В таких случаях злоумышленник может взять от блока шифротекста корень степени e по модулю n, что в данном случае будет намного быстрее факторизации. Поэтому короткие блоки обязательно надо «набивать» дополнительными битами. Еще один интересный вопрос касается простых чисел. Для построения ключей алгоритму RSA необходимо найти два простых числа. Благо, среди чисел простые попадаются довольно часто: на отрезке от 1 до n примерно n/ln(n) чисел являются простыми. Поэтому можно просто брать псевдослучайные числа нужной длины и проверять их на простоту. Проверку числа на простоту можно делать двумя способами: «в лоб», перебором всех его делителей (от 2 до округленного корня из n), или с помощью более «хитрых» тестов на делимость. При переборе всех делителей мы гарантированно можем утверждать, что прошедшее такую проверку число является простым. Однако, время работы такой процедуры будет очень велико. Среди «хитрых» тестов надо выделить тест Миллера–Рабина, так как он на сегодняшний день является наиболее лучшим по всем параметрам. Так вот, проверка ллера–Рабина работает намного быстрее перебора делителей, но в отличие от него, число, выдаваемое им как простое, с некоторой очень маленькой вероятностью может оказаться составным. На практике обычно применяют последовательно несколько разных «хитрых» тестов, минимизируя тем самым вероятность ошибки. Программная реализация RSA работает медленнее примерно на два порядка по сравнению с симметричными алгоритмами. RSA часто используют вместе с каким-нибудь симметричным шифром. При таком способе все сообщения шифруются с помощью более быстрого симметричного алгоритма, а для пересылки сессионного секретного ключа этого симметричного алгоритма используется RSA. Получается вычислительно дешево и сердито.

Проясним использование алгоритма RSA на конкретном примере. Выбираем два простые числа p=7; q=17 (на практике эти числа во много раз длиннее). В этом случае n = p*q будет равно 119. Теперь необходимо выбрать e, выбираем e=5. Следующий шаг связан с формированием числа d так, чтобы (d*e) mod [(p-1)(q-1)] =1. d=77 (использован расширенный алгоритм Эвклида). d – секретный ключ, а e и n характеризуют открытый ключ. Пусть текст, который нам нужно зашифровать представляется M=19. C = Memod n. Получаем зашифрованный текст C=66. Этот «текст» может быть послан

соответствующему адресату. Получатель дешифрует полученное сообщение, используя M= Cdmod n и C=66. В результате получается M=19.

На практике общедоступные ключи могут помещаться в специальную базу данных. При необходимости послать партнеру зашифрованное сообщение можно сделать сначала запрос его открытого ключа. Получив его, можно запустить программу шифрации, а результат ее работы послать адресату. На использовании общедоступных ключей базируется и так называемая электронная подпись, которая позволяет однозначно идентифицировать отправителя. Сходные средства могут применяться для предотвращения внесения каких-либо корректив в сообщение на пути от отправителя к получателю. Быстродействующие аппаратные 512-битовые модули могут обеспечить скорость шифрования на уровне 64 кбит в сек. Готовятся ИС, способные выполнять такие операции со скоростью 1 Мбайт/сек. Разумный выбор параметра e позволяет заметно ускорить реализацию алгоритма.

ХОД РАБОТЫ

Было написано 2 приложения:

- REST-сервис на языке Kotlin для генерации чисел длинной арифметики, поскольку в C++ таковой нет, как и самой поддержки чисел длинной арифметики, использовался фреймворк Spring (исходный код в Приложении А);

- десктопное приложение на языке C++ с использованием фреймворка Qt с подготовленным классом BigInt, который позволит выполнять возведение в степень и вычислять деление по модулю (исходный код в Приложении Б, за исключением класса BigInt, поскольку он слишком большой и был взят из сети Интернет).

На рисунке 1 представлен вывод REST-сервиса в браузер 16-разрядных чисел ***p*** и ***q***, а также полученным по ним закрытым и открытым ключам.

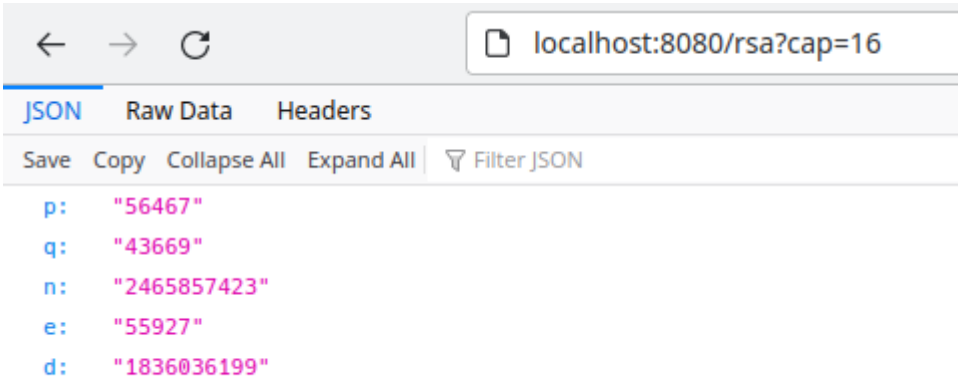

Рисунок 1 – Главная форма программы.

На рисунке 2 представлен графический интерфейс десктопного приложения.

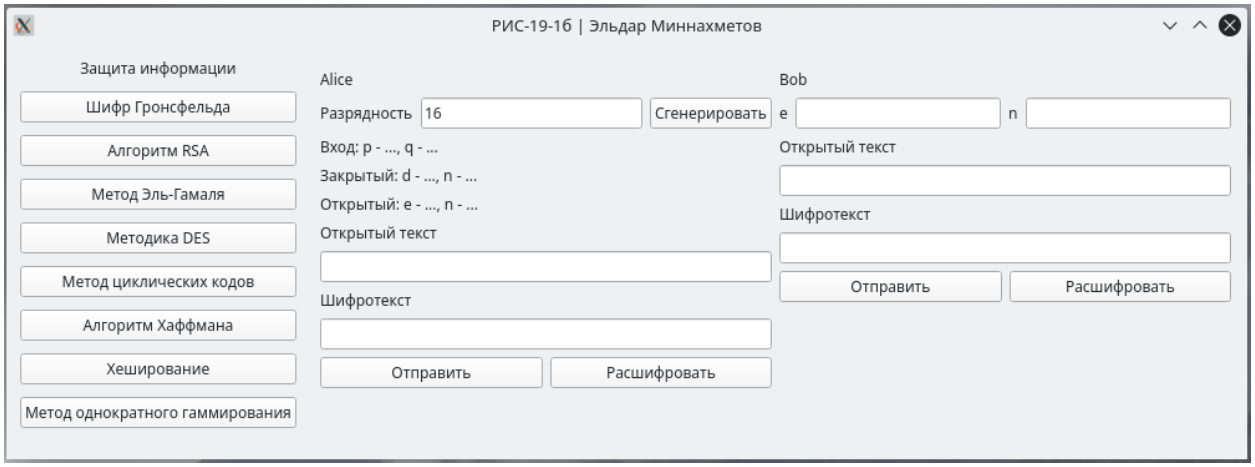

Рисунок 2 – Пример работы программы.

После нажатия на кнопку «Сгенерировать» десктопное приложение отправляет запрос ключей с указанной разрядностью чисел ***p*** и ***q*** на входе генерации. Результат генерации представлен на рисунке 3.

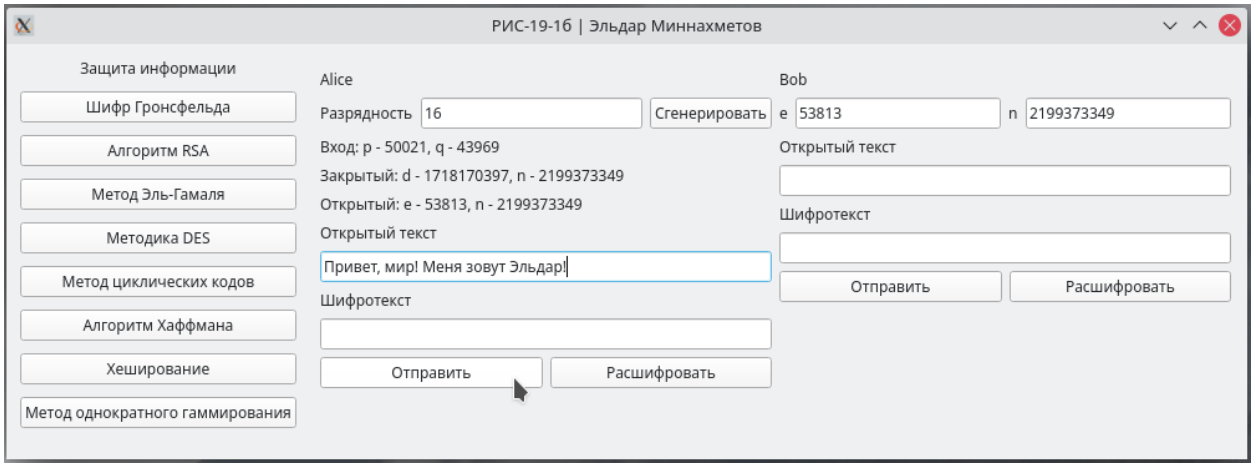

Рисунок 3 – Результат генерации ключей

В приложении показано общение Алисы и Боба с использованием технологии шифрования RSA. Закрытый ключ хранится у Алисы – она с его помощью шифрует сообщение и отправляет Бобу – в случае, если Боб сможет расшифровать сообщение своим открытым ключом, будет подтверждена личность Алисы. На рисунке 4 показана отправка сообщения Алисой Бобу.

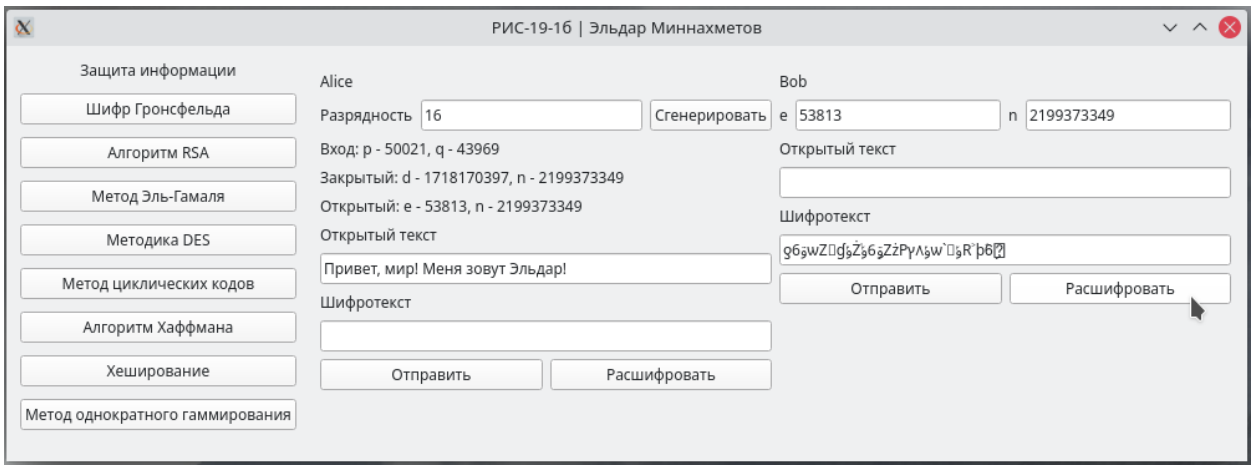

Рисунок 4 – Отправка Алисой зашифрованного сообщения Бобу

Далее Бобу требуется расшифровать полученное сообщение с помощью своего открытого ключа, что показано на рисунке 5.

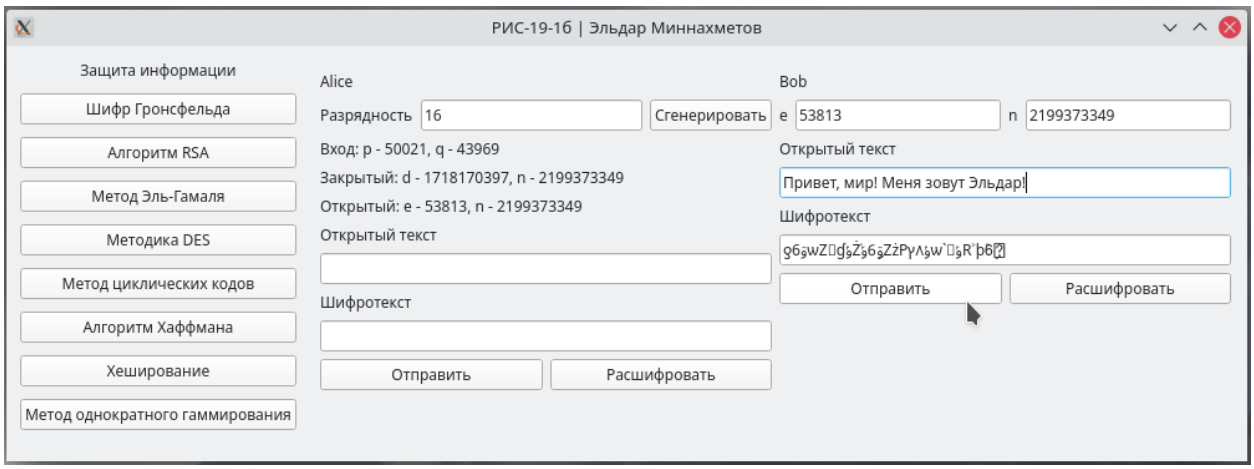

Рисунок 5 – Расшифровывание полученного сообщения Бобом

Сообщение успешно расшифровано. Теперь же, чтобы удостовериться, что открытый и закрытый ключи синергируют друг с другом, следует отправить это же сообщение обратно Алисе. Выполнение текущего шага показано на рисунке 6.

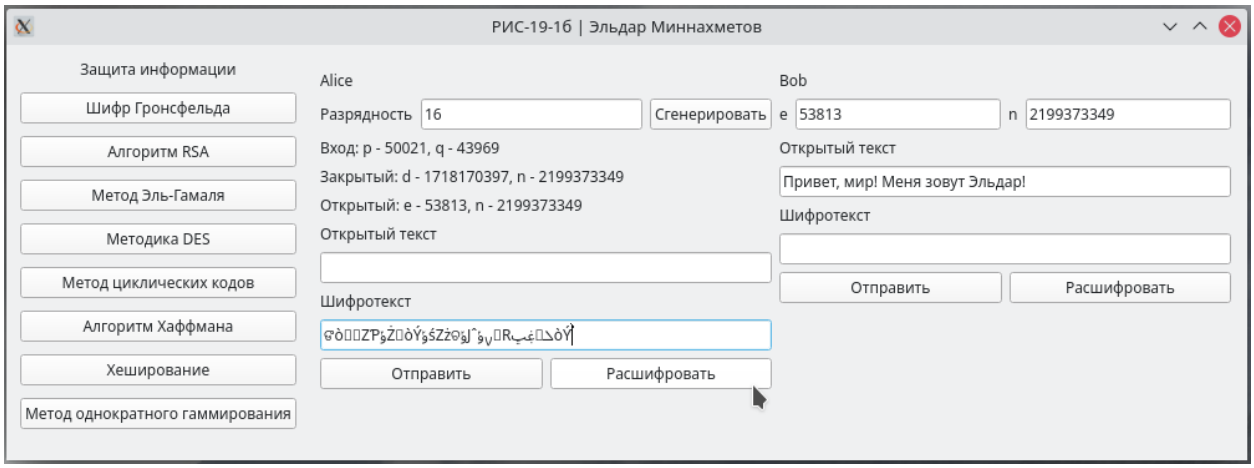

Рисунок 6 – Отправка Бобом зашифрованного сообщения Алисе

Зашифрованное сообщение получено Алисой. Следует обратить внимание на шифротексты Алисы и Боба, представленные на рисунках 5 и 6 – они различны, поскольку были получены шифрованием разными ключами. Однако результат разшифровки Алисой покажет, что открытый и закрытый ключи отлично синергируют друг с другом. Он представлен на рисунке 7.

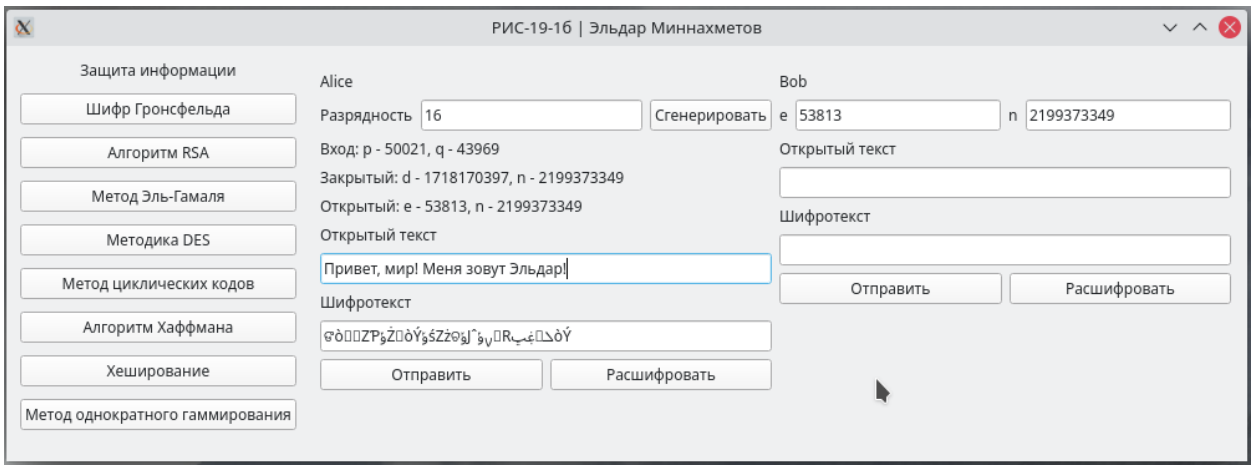

Рисунок 7 – Расшифровывание полученного сообщения Алисой

По итогу был получен изначальный текст, отправленный Алисой.

Вывод: расшифровка шифротекста Бобом подтвердила личность Алисы и между ними было установлено взаимодействие.

ПРИЛОЖЕНИЕ А

Листинг файла BackApplication.kt

```
package org.eldarian.back

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestParam
import org.springframework.web.bind.annotation.RestController

@SpringBootApplication
class BackApplication

fun main(args: Array<String>) {
    runApplication<BackApplication>(*args)
}

@RestController
class LController {
    @GetMapping("/rsa")
    fun rsa(@RequestParam(value = "cap", defaultValue = "16") bits: Int): RsaResponse {
        return generate(bits)
    }
}
```

Листинг файла RSA.kt

```
package org.eldarian.back

import java.math.BigInteger
import java.util.*

data class RsaResponse (
    val p: String,
    val q: String,
    val n: String,
    val e: String,
    val d: String,
) {
    constructor(
        p: BigInteger,
        q: BigInteger,
        n: BigInteger,
        e: BigInteger,
        d: BigInteger,
    ) : this(
        p.toString(),
        q.toString(),
        n.toString(),
        e.toString(),
        d.toString(),
    )
}

fun generate(bits: Int): RsaResponse {
    val rnd = Random()
    var gcd: BigInteger
    val p = BigInteger.probablePrime(bits, rnd)
    var q: BigInteger
    var e: BigInteger

    do {
        q = BigInteger.probablePrime(bits, rnd)
    } while (p == q)

    val n: BigInteger = p.multiply(q)
    val phi: BigInteger = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE))

    do {
        e = BigInteger.probablePrime(bits, rnd)
        gcd = e.gcd(phi)
    } while (gcd != BigInteger.ONE)

    val d: BigInteger

    var i = BigInteger.ONE
    while(true) {
        val top = i.multiply(phi).add(BigInteger.ONE)
```

```
        if(top > e && e / top.gcd(e) == BigInteger.ONE) {
            d = top.divide(e)
            break
        }
        ++i
    }

    return RsaResponse(p, q, n, e, d)
}
```

ПРИЛОЖЕНИЕ Б

Листинг класса RsaTask.h

```
#pragma once

#include <QWidget>
#include <QLabel>
#include <utility>
#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QJsonObject>
#include <QJsonDocument>
#include <QHBoxLayout>
#include <QLineEdit>
#include <QPushButton>

#include "Task.h"
#include "BigInt.h"

class RsaClient {
private:
   BigInt e, n;
   int k;

   QString crypt(const BigInt &c);

public:
   RsaClient(const BigInt& e, const BigInt& n);
   QString crypt(const QString &in);

   const BigInt &E() { return e; }
   const BigInt &N() { return n; }
};

class RsaTask;

class RsaLoader : public QObject {
Q_OBJECT
private:
   QNetworkAccessManager* manager;
   RsaTask* task;
   BigInt p, q, e, n, d;

public:
   explicit RsaLoader(RsaTask* task);
   void download(QString capacity);

private:
   void done(const QUrl& url, const QByteArray& array);

private slots:
```

```
    void slotFinished(QNetworkReply* reply);

};

class RsaTask: public QObject, public Task {
Q_OBJECT
private:
    bool isAlice, isCrypt;

    QHBoxLayout *lytMain;
    QVBoxLayout *lytAlice;
    QHBoxLayout *lytCapacity;
    QVBoxLayout *lytBob;
    QHBoxLayout *lytABtns;
    QHBoxLayout *lytEN;
    QHBoxLayout *lytBBtns;

    QLabel *lblAlice;
    QLabel *lblBob;
    QLabel *lblCapacity;
    QLabel *lblE;
    QLabel *lblN;
    QLabel *lblInput;
    QLabel *lblPrivate;
    QLabel *lblPublic;
    QLabel *lblAIn;
    QLabel *lblAOut;
    QLabel *lblBIn;
    QLabel *lblBOut;

    QLineEdit *txtCapacity;
    QLineEdit *txtE;
    QLineEdit *txtN;
    QLineEdit *txtAIn;
    QLineEdit *txtAOut;
    QLineEdit *txtBIn;
    QLineEdit *txtBOut;

    QPushButton *btnLoad;
    QPushButton *btnACrypt;
    QPushButton *btnADecrypt;
    QPushButton *btnBCrypt;
    QPushButton *btnBDecrypt;

    RsaClient *alice = nullptr;
    RsaClient *bob = nullptr;
    RsaLoader *loader;

public slots:
    void getRsa();
    void crypt(bool isAlice, bool isCrypt);
    void aCrypt();
    void aDecrypt();
```

```
    void bCrypt();
    void bDecrypt();

public:
    RsaTask();
    void setRsa(const BigInt &p, const BigInt &q, const BigInt &e, const BigInt &n, const BigInt &d);
    void initWidget(QWidget *wgt) override;
    void run() const override;

};
```

Листинг класса RsaTask.cpp

```
#include "RsaTask.h"

RsaClient::RsaClient(const BigInt& e, const BigInt& n) : e(e), n(n) {
    k = BigInt::log2(n);
    k > 16 ? k /= 16 : k = 1;
}

QString RsaClient::crypt(const QString &in) {
    QString result;
    BigInt multiplier(65536);
    for (int i = 0; i < in.size(); i += k) {
        BigInt symbol(0);
        for (int j = i; j < in.size() && j < i + k; ++j) {
            symbol *= multiplier;
            symbol += BigInt((int)in[j].unicode());
        }
        result.append(crypt(symbol));
    }
    return result;
}

QString RsaClient::crypt(const BigInt &c) {
    BigInt r = BigInt::pow(c, e) % n;
    BigInt nul(0);
    BigInt multiplier(65536);
    QString result;
    for(int i = 0; i < k && r != nul; ++i) {
        result = QChar(stoi(BigInt::to_string(r % multiplier))) + result;
        r /= multiplier;
    }
    return result;
}

RsaLoader::RsaLoader(RsaTask* task) {
    this->task = task;
    manager = new QNetworkAccessManager(this);
    connect(manager, SIGNAL(finished(QNetworkReply*)), SLOT(slotFinished(QNetworkReply*)));
}

void RsaLoader::download(QString capacity) {
    manager->get(QNetworkRequest(QUrl("http://localhost:8080/rsa?cap=" + capacity)));
}

void RsaLoader::done(const QUrl& url, const QByteArray& array) {
    QJsonObject json = QJsonDocument::fromJson(array).object();
    p = BigInt(json["p"].toString().toStdString());
    q = BigInt(json["q"].toString().toStdString());
    e = BigInt(json["e"].toString().toStdString());
    n = BigInt(json["n"].toString().toStdString());
    d = BigInt(json["d"].toString().toStdString());
    task->setRsa(p, q, e, n, d);
```

```
}

void RsaLoader::slotFinished(QNetworkReply* reply) {
    if (reply->error() == QNetworkReply::NoError) {
        done(reply->url(), reply->readAll());
    }
    reply->deleteLater();
}

RsaTask::RsaTask(): Task("Алгоритм RSA") {
    loader = new RsaLoader(this);
}

void RsaTask::getRsa() {
    loader->download(txtCapacity->text());
}

void RsaTask::crypt(bool isAlice, bool isCrypt) {
    this->isAlice = isAlice;
    this->isCrypt = isCrypt;
    const QString &in = (isAlice ? (isCrypt ? txtAIn : txtAOut) : (isCrypt ? txtBIn : txtBOut))->text();
    const QString &out = (isAlice ? alice : bob)->crypt(in);
    (isAlice ? (isCrypt ? txtBOut : txtAIn) : (isCrypt ? txtAOut : txtBIn))->setText(out);
}

void RsaTask::aCrypt() {
    crypt(true, true);
}

void RsaTask::aDecrypt() {
    crypt(true, false);
}

void RsaTask::bCrypt() {
    crypt(false, true);
}

void RsaTask::bDecrypt() {
    crypt(false, false);
}

void RsaTask::setRsa(const BigInt &p, const BigInt &q, const BigInt &e, const BigInt &n, const BigInt &d) {
    if(alice) {
        delete alice;
        delete bob;
        alice = nullptr;
        bob = nullptr;
    }
    alice = new RsaClient(d, n);
    bob = new RsaClient(e, n);
    lblInput->setText(("Вход: p - " + BigInt::to_string(p) + ", q - "  + BigInt::to_string(q)).c_str());
    lblPrivate->setText(("Закрытый: d - " + BigInt::to_string(d) + ", n - "  + BigInt::to_string(n)).c_str());
```

```
    lblPublic->setText(("Открытый: e - " + BigInt::to_string(e) + ", n - " + BigInt::to_string(n)).c_str());
    txtE->setText(BigInt::to_string(e).c_str());
    txtN->setText(BigInt::to_string(n).c_str());
}

void RsaTask::initWidget(QWidget *wgt) {
    lytMain = new QHBoxLayout();
    lytAlice = new QVBoxLayout();
    lytBob = new QVBoxLayout();
    lytCapacity = new QHBoxLayout();
    lytABtns = new QHBoxLayout();
    lytEN = new QHBoxLayout();
    lytBBtns = new QHBoxLayout();

    lytAlice->setAlignment(Qt::Alignment::enum_type::AlignTop);
    lytBob->setAlignment(Qt::Alignment::enum_type::AlignTop);

    lblAlice = new QLabel("Alice");
    lblBob = new QLabel("Bob");
    lblCapacity = new QLabel("Разрядность");
    lblE = new QLabel("e");
    lblN = new QLabel("n");
    lblInput = new QLabel("Вход: p - ..., q - ...");
    lblPrivate = new QLabel("Закрытый: d - ..., n - ...");
    lblPublic = new QLabel("Открытый: e - ..., n - ...");
    lblAIn = new QLabel("Открытый текст");
    lblAOut = new QLabel("Шифротекст");
    lblBIn = new QLabel("Открытый текст");
    lblBOut = new QLabel("Шифротекст");

    txtCapacity = new QLineEdit("16");
    txtE = new QLineEdit();
    txtN = new QLineEdit();
    txtAIn = new QLineEdit();
    txtAOut = new QLineEdit();
    txtBIn = new QLineEdit();
    txtBOut = new QLineEdit();

    btnLoad = new QPushButton("Сгенерировать");
    btnACrypt = new QPushButton("Отправить");
    btnADecrypt = new QPushButton("Расшифровать");
    btnBCrypt = new QPushButton("Отправить");
    btnBDecrypt = new QPushButton("Расшифровать");

    wgt->setLayout(lytMain);
    lytMain->addLayout(lytAlice);
    lytAlice->addWidget(lblAlice);
    lytAlice->addLayout(lytCapacity);
    lytCapacity->addWidget(lblCapacity);
    lytCapacity->addWidget(txtCapacity);
    lytCapacity->addWidget(btnLoad);
    lytAlice->addWidget(lblInput);
    lytAlice->addWidget(lblPrivate);
```

```
    lytAlice->addWidget(lblPublic);
    lytAlice->addWidget(lblAIn);
    lytAlice->addWidget(txtAIn);
    lytAlice->addWidget(lblAOut);
    lytAlice->addWidget(txtAOut);
    lytAlice->addLayout(lytABtns);
    lytABtns->addWidget(btnACrypt);
    lytABtns->addWidget(btnADecrypt);
    lytMain->addLayout(lytBob);
    lytBob->addWidget(lblBob);
    lytBob->addLayout(lytEN);
    lytEN->addWidget(lblE);
    lytEN->addWidget(txtE);
    lytEN->addWidget(lblN);
    lytEN->addWidget(txtN);
    lytBob->addWidget(lblBIn);
    lytBob->addWidget(txtBIn);
    lytBob->addWidget(lblBOut);
    lytBob->addWidget(txtBOut);
    lytBob->addLayout(lytBBtns);
    lytBBtns->addWidget(btnBCrypt);
    lytBBtns->addWidget(btnBDecrypt);

    connect(btnLoad, SIGNAL(released()), this, SLOT(getRsa()));
    connect(btnACrypt, SIGNAL(released()), this, SLOT(aCrypt()));
    connect(btnADecrypt, SIGNAL(released()), this, SLOT(aDecrypt()));
    connect(btnBCrypt, SIGNAL(released()), this, SLOT(bCrypt()));
    connect(btnBDecrypt, SIGNAL(released()), this, SLOT(bDecrypt()));
}

void RsaTask::run() const {

}
```