

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 №3

Тема: «Цифровая подпись методом Эль-Гамала»

Выполнил: студент группы РИС-19-16

Миннахметов Э.Ю. _____

Проверил: доцент кафедры ИТАС

Шереметьев В. Г. _____

Дата _____

Пермь, 2021

ЦЕЛЬ РАБОТЫ

Получить практические навыки по использованию несимметричных алгоритмов шифрования, на примере использования алгоритма Диффи-Хеллмана и Эль-Гамала, а также по применению электронных подписей на примере метода Эль-Гамала.

ЗАДАНИЕ

Вариант №14. Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамала, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Алгоритм Диффи-Хеллмана:

Алгоритм назван по фамилиям его создателей Диффи (Diffie) и Хеллмана (Hellman).

Метод помогает обмениваться секретным ключом для симметричных криптосистем, но использует метод, очень похожий на асимметричный алгоритм RSA. Это не симметричный алгоритм, так как для шифрования и дешифрования используются различные ключи. Так же это не схема с открытым ключом, потому что ключи легко получаются один из другого, и ключ шифрования и ключ дешифрования должны храниться в секрете.

Определим круг его возможностей. Предположим, что двум абонентам необходимо провести конфиденциальную переписку, а в их распоряжении нет первоначально оговоренного секретного ключа. Однако, между ними существует канал, защищенный от модификации, то есть данные, передаваемые по нему, могут быть прослушаны, но не изменены (такие условия имеют место довольно часто). В этом случае две стороны могут создать одинаковый секретный ключ, ни разу не передав его по сети, по следующему алгоритму.

Предположим, что обоим абонентам известны некоторые два числа v и q . Они, впрочем, известны и всем остальным заинтересованным лицам. Например, они могут быть просто фиксированно «защиты» в программное обеспечение. Далее один из партнеров $P1$ генерирует случайное или псевдослучайное простое число x и посылает другому участнику будущих обменов $P2$ значение $A = qx \bmod n$

По получении A партнер $P2$ генерирует случайное или псевдослучайное простое число y и посылает $P2$ вычисленное значение $B = qy \bmod n$

Партнер P1, получив B, вычисляет $K_x = Bx \bmod n$, а партнер P2 вычисляет $K_y = Ay \bmod n$. Алгоритм гарантирует, что числа K_y и K_x равны и могут быть использованы в качестве секретного ключа для шифрования. Ведь даже перехватив числа A и B, трудно вычислить K_x или K_y .

Например, по вычисленным $K_x = K_y = K$ абоненты могут зашифровать сообщение $M=123$ по следующему алгоритму: к каждому символу сообщения M добавить K => сообщение $C=234$, при $K=1$. Соответственно алгоритмом расшифрования будет разность ключа K из каждого символа сообщения C.

Пример:

Пусть

$n=3$; $q=5$;

$x=5$; $y=7$;

тогда $A = q^x \bmod n = 1$, а $B = q^y \bmod n = 2$, то вычислив $K_x = B^x \bmod n$ и $K_y = A^y \bmod n$ получим $K_x = K_y = 1$. Зашифруем приведенное выше сообщение $M=123$ по приведенному выше алгоритму => сообщение $C=234$, расшифровав сообщение C по обратному алгоритму получим сообщение $M=123$.

Необходимо еще раз отметить, что алгоритм Диффи-Хеллмана работает только на линиях связи, надежно защищенных от модификации. Если бы он был применим на любых открытых каналах, то давно снял бы проблему распространения ключей и, возможно, заменил собой всю асимметричную криптографию. Однако, в тех случаях, когда в канале возможна модификация данных, появляется очевидная возможность вклинивания в процесс генерации ключей «злоумышленника-посредника» по той же самой схеме, что и для асимметричной криптографии.

Алгоритм Эль-Гамала:

Алгоритм Эль-Гамала может использоваться для формирования электронной подписи или для шифрования данных. Он базируется на трудности вычисления дискретного логарифма. Для генерации пары ключей сначала берется простое число p и два случайных простых числа g и x, каждое из которых меньше p. Затем вычисляется:

$$y = g^x \bmod p$$

Общедоступными ключами являются y, g и p, а секретным ключом является x. Для подписи сообщения M выбирается случайное число k, которое является простым по отношению к p-1. После этого вычисляется $a = g^k \bmod p$. Далее из уравнения $M = (xa + kb) \bmod (p-1)$ находим b. Электронной подписью для сообщения M будет служить пара a и b.

Случайное число k следует хранить в секрете. Для верификации подписи необходимо проверить равенство:

$$y^a \cdot a^b \bmod p = g^M \bmod p.$$

Пример:

Выберем $p=11$, $g=2$, а закрытый ключ $x=8$.

Вычислим $y = g^x \bmod p = 3$.

Открытым ключом являются $p=11$, $g=2$, $y=3$, чтобы подписать $M=5$ сначала выберем случайное число $k=9$. Вычисляем $a = g^k \bmod p = 2^9 \bmod 11 = 6$ и с помощью расширенного алгоритма Эвклида находим:

$$M = (xa + kb) \bmod (p - 1)$$

$$5 = (8 \cdot 6 + 9 \cdot b) \bmod 10$$

Решение: $b=3$, а подпись представляет собой пару $a=6$ и $b=3$.

Шифрование Эль-Гамала:

Модификация алгоритма позволяет шифровать сообщения. Для шифрования сообщения M сначала выбирается случайное число k , взаимно простое с $p-1$, затем вычисляются

$$a = g^k \bmod p$$

$$b = y^k M \bmod p$$

Пара a и b представляют собой зашифрованный текст. Следует заметить, что зашифрованный текст имеет размер в два раза больше исходного. Для дешифрования производится вычисление:

$$M = b/a^x \bmod p.$$

Пример:

Вернемся к предыдущему примеру:

$$a=6, b = y^k M \bmod p = 3.$$

Расшифруем: $M = b/a^x \bmod p = 5$.

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 по запросу ключей, необходимых для цифровой подписи методов Эль-Гамала, в браузер.

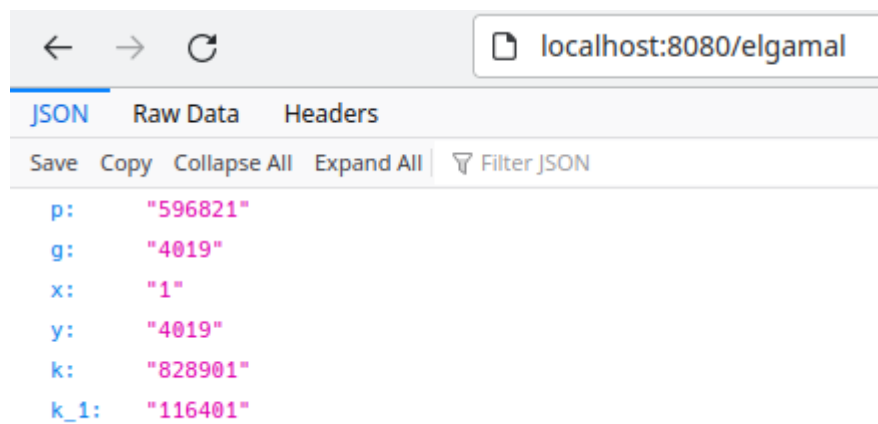


Рисунок 1 – Вывод результата запроса к серверу в браузере

На рисунке 2 представлен графический интерфейс десктопного приложения.

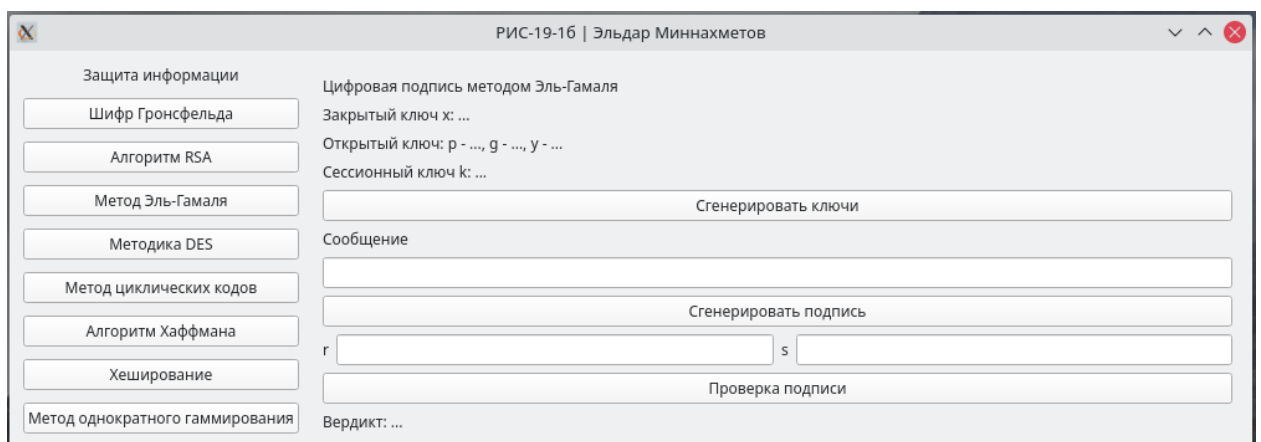
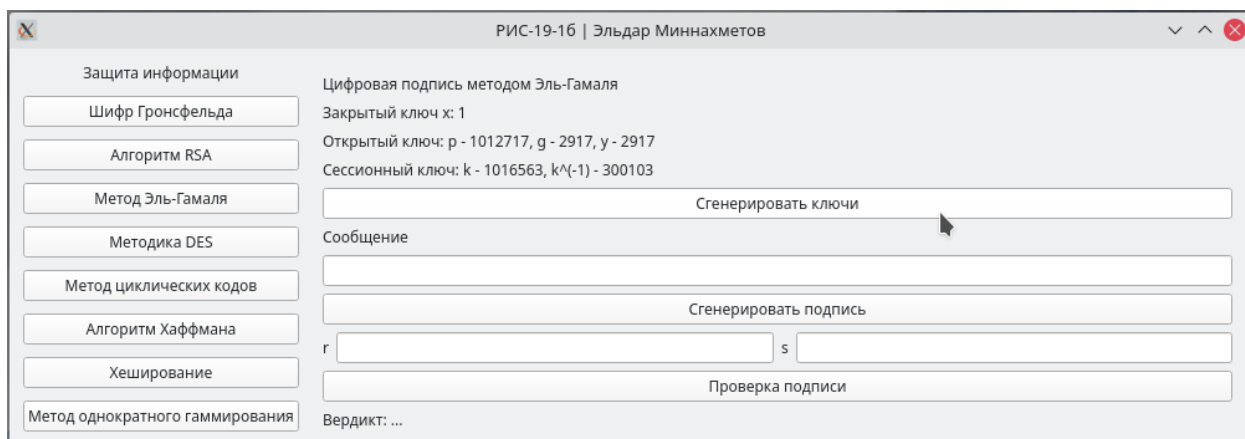


Рисунок 2 – Графический пользовательский интерфейс

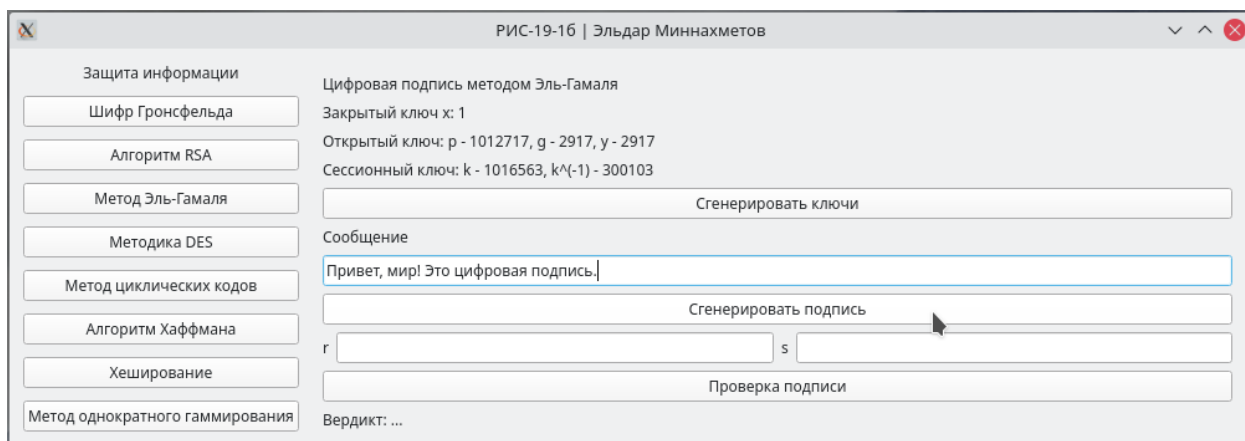
После нажатия на кнопку «Сгенерировать ключи» десктопное приложение отправляет запрос ключей. Полученные ключи вводятся в соответствующие поля на форме. Результат генерации представлен на рисунке 3.



The screenshot shows a desktop application window titled "РИС-19-16 | Эльдар Миннахметов". On the left, under "Защита информации", there is a list of encryption methods: Шифр Гронсфельда, Алгоритм RSA, Метод Эль-Гамала, Методика DES, Метод циклических кодов, Алгоритм Хаффмана, Хеширование, and Метод однократного гаммирования. The right side is titled "Цифровая подпись методом Эль-Гамала" and displays the generated keys: "Закрытый ключ x: 1", "Открытый ключ: p - 1012717, g - 2917, y - 2917", and "Сессионный ключ: k - 1016563, k^(-1) - 300103". Below this, there is a button "Сгенерировать ключи" which has been clicked. Further down, there is a "Сообщение" field, a "Сгенерировать подпись" button, and fields for the signature components "r" and "s". At the bottom, there is a "Проверка подписи" button and a "Вердикт: ..." label.

Рисунок 3 – Результат генерации ключей

Далее необходимо ввести сообщение, у которого будет вычисляться пара (r, s).



This screenshot is identical to the previous one, but the "Сообщение" field now contains the text "Привет, мир! Это цифровая подпись." The mouse cursor is positioned over the "Сгенерировать подпись" button.

Рисунок 4 – Ввод текстового сообщения

По нажатию на кнопку «Сгенерировать подпись» выполняется генерация пары (r, s) и затем она записывается в соответствующие поля, как показано на рисунке 5.

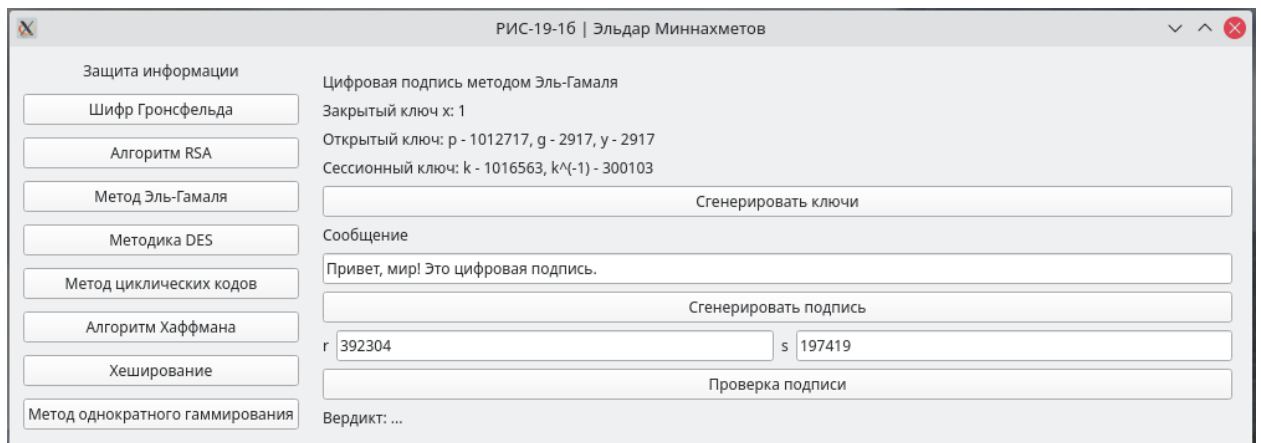


Рисунок 5 – Генерация пары (r, s)

После чего становится доступна функция проверки тройки <Сообщение, r, s> на достоверность, что представлено на рисунке 6.

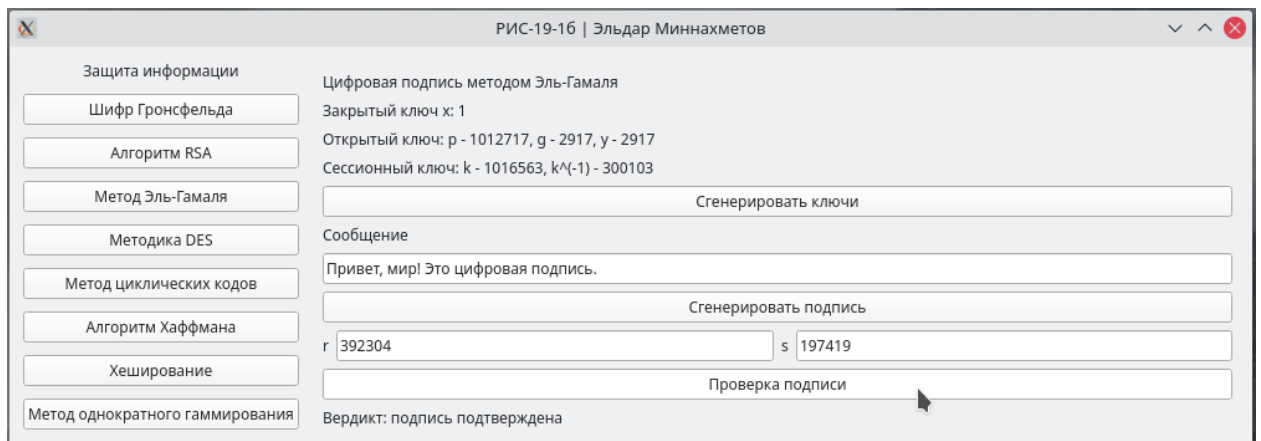


Рисунок 6 – Проверка достоверности цифровой подписи

Если же изменить текст подписи или поля r и s, то такая тройка <Сообщение, r, s> уже не будет проходить проверку подлинности – это продемонстрировано на рисунке 7.

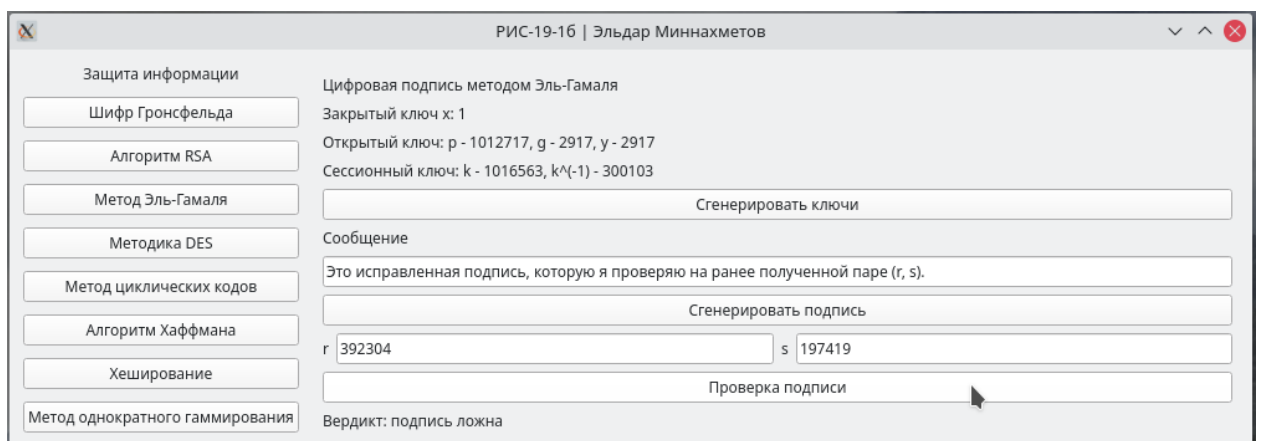


Рисунок 7 – Проверка достоверности цифровой подписи

ПРИЛОЖЕНИЕ А

Листинг класса 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.RequestMapping
import org.springframework.web.bind.annotation.RestController

@SpringBootApplication
class BackApplication

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

@RestController
class LController {
    @GetMapping("/elgamal")
    fun elgamal(): ElgamalResponse {
        return generateElgamal()
    }
}
```


Листинг файла common.kt

```
package org.eldarian.back

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

data class ElgamalResponse (
    val p: String,
    val g: String,
    val x: String,
    val y: String,
    val k: String,
    val k_1: String,
) {
    constructor(
        p: BigInteger,
        g: BigInteger,
        x: BigInteger,
        y: BigInteger,
        k: BigInteger,
        k_1: BigInteger,
    ) : this(
        p.toString(),
        g.toString(),
        x.toString(),
        y.toString(),
        k.toString(),
        k_1.toString(),
    )
}

fun generateElgamal(): ElgamalResponse {
    val PK = 20
    val XG = 12
    val rnd = Random()
    val p = BigInteger.probablePrime(PK, rnd)
    val g = p.coprime(rnd, XG);
    val x = randBetween(rnd, XG, false, BigInteger.ONE, p)
    val y = g.modPow(x, p)
    val k = randBetween(rnd, PK, false, BigInteger.ONE, p - BigInteger.ONE)
    val k_1 = k.multiplicative(p - BigInteger.ONE)
    return ElgamalResponse(p, g, x, y, k, k_1)
}

fun randBetween(rnd: Random, bits: Int, prime: Boolean, low: BigInteger? = null, high: BigInteger? = null) :
BigInteger {
    return rand(rnd, bits, prime) { (low == null || low < it) && (high == null || it < high) }
}

fun rand(rnd: Random, bits: Int, prime: Boolean, f: (BigInteger) -> Boolean): BigInteger {
    var result: BigInteger
    do {
        if(prime) {
```

```

        result = BigInteger.probablePrime(bits, rnd)
    } else {
        result = BigInteger(bits, rnd)
    }
} while(f(result))
return result
}

fun BigInteger.coprime(rnd: Random, bits: Int): BigInteger {
    var r: BigInteger
    var gcd: BigInteger
    do {
        r = BigInteger.probablePrime(bits, rnd)
        gcd = this.gcd(r)
    } while (gcd != BigInteger.ONE)
    return r
}

fun BigInteger.multiplicative(mod: BigInteger): BigInteger {
    val r: BigInteger
    var i = BigInteger.ONE
    while(true) {
        val top = i.multiply(mod).add(BigInteger.ONE)
        if(top > this && this / top.gcd(this) == BigInteger.ONE) {
            r = top.divide(this)
            break
        }
        ++i
    }
    return r
}

```

ПРИЛОЖЕНИЕ Б

Листинг файла ElgamalTask.h

```
#pragma once

#include <QWidget>

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

class QLabel;
class QLineEdit;
class QPushButton;
class QVBoxLayout;
class QHBoxLayout;

class ElgamalClient;

class ElgamalTask: public QObject, public Task {
    Q_OBJECT

private:
    ElgamalClient* client = nullptr;

    QVBoxLayout *lytMain;
    QHBoxLayout *lytPair;

    QLabel *lblName;
    QLabel *lblPrivate;
    QLabel *lblPublic;
    QLabel *lblSession;
    QLabel *lblMessage;
    QLabel *lblR;
    QLabel *lblS;
    QLabel *lblVerdict;

    QLineEdit *leMessage;
    QLineEdit *leR;
    QLineEdit *leS;

    QPushButton *btnGenerate;
    QPushButton *btnCrypt;
    QPushButton *btnCheck;

public:
    ElgamalTask(): Task("Метод Эль-Гамаля") {}

    void initWidget(QWidget *wgt) override;
    void setElgamal(const BigInt &p, const BigInt &g, const BigInt &x, const BigInt &y, const BigInt &k, const
```

```
    BigInt &k_1);
```

```
public slots:
```

```
    void getElgamal();
```

```
    void crypt();
```

```
    void check();
```

```
};
```

```
class ElgamalLoader : public LoadTask {
```

```
private:
```

```
    ElgamalTask* task;
```

```
public:
```

```
    explicit ElgamalLoader(ElgamalTask* task) : task(task) {}
```

```
    QString query() override { return "elgamal"; }
```

```
    void done(QJsonObject& json) override;
```

```
};
```

```
class ElgamalClient {
```

```
private:
```

```
    BigInt p, g, x, y, k, k_1;
```

```
public:
```

```
    ElgamalClient(const BigInt &p, const BigInt &g, const BigInt &x, const BigInt &y, const BigInt &k, const  
    BigInt &k_1)
```

```
        : p(p), g(g), x(x), y(y), k(k), k_1(k_1) {}
```

```
    void generate(const QString &m, BigInt &r, BigInt &s);
```

```
    bool check(const QString &m, const BigInt &r, const BigInt &s);
```

```
};
```

Листинг файла ElgamalTask.cpp

```
#include "ElgamalTask.h"
#include "BigInt.h"

#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QJsonObject>

void ElgamalTask::initWidget(QWidget *wgt) {
    lytMain = new QVBoxLayout;
    lytPair = new QHBoxLayout;

    lblName = new QLabel("Цифровая подпись методом Эль-Гамала");
    lblPrivate = new QLabel("Закрытый ключ x: ...");
    lblPublic = new QLabel("Открытый ключ: p - ..., g - ..., y - ...");
    lblSession = new QLabel("Сессионный ключ k: ...");
    lblMessage = new QLabel("Сообщение");
    lblR = new QLabel("r");
    lblS = new QLabel("s");
    lblVerdict = new QLabel("Вердикт: ...");

    leMessage = new QLineEdit;
    leR = new QLineEdit;
    leS = new QLineEdit;

    btnGenerate = new QPushButton("Сгенерировать ключи");
    btnCrypt = new QPushButton("Сгенерировать подпись");
    btnCheck = new QPushButton("Проверка подписи");

    wgt->setLayout(lytMain);
    lytMain->addWidget(lblName);
    lytMain->addWidget(lblPrivate);
    lytMain->addWidget(lblPublic);
    lytMain->addWidget(lblSession);
    lytMain->addWidget(btnGenerate);
    lytMain->addWidget(lblMessage);
    lytMain->addWidget(leMessage);
    lytMain->addWidget(btnCrypt);
    lytMain->addLayout(lytPair);
    lytPair->addWidget(lblR);
    lytPair->addWidget(leR);
    lytPair->addWidget(lblS);
    lytPair->addWidget(leS);
    lytMain->addWidget(btnCheck);
    lytMain->addWidget(lblVerdict);
    lytMain->setAlignment(Qt::Alignment::enum_type::AlignTop);

    connect(btnGenerate, SIGNAL(released()), this, SLOT(getElgamal()));
    connect(btnCrypt, SIGNAL(released()), this, SLOT(crypt()));
```

```

    connect(btnCheck, SIGNAL(released()), this, SLOT(check()));
}

void ElgamalTask::setElgamal(const BigInt &p, const BigInt &g, const BigInt &x, const BigInt &y, const BigInt
&k, const BigInt &k_1) {
    if(client) {
        delete client;
    }
    client = new ElgamalClient(p, g, x, y, k, k_1);
    lblPrivate->setText(("Закрытый ключ x: " + BigInt::to_string(x)).c_str());
    lblPublic->setText(("Открытый ключ: p - " + BigInt::to_string(p) +
        ", g - " + BigInt::to_string(g) +
        ", y - " + BigInt::to_string(y)).c_str());
    lblSession->setText(("Сессионный ключ: k - " + BigInt::to_string(k) +
        ", k^(-1) - " + BigInt::to_string(k_1)).c_str());
}

void ElgamalTask::getElgamal() {
    (new ElgamalLoader(this))->run();
}

void ElgamalTask::crypt() {
    BigInt r, s;
    client->generate(leMessage->text(), r, s);
    leR->setText(BigInt::to_string(r).c_str());
    leS->setText(BigInt::to_string(s).c_str());
}

void ElgamalTask::check() {
    BigInt r(leR->text().toStdString());
    BigInt s(leS->text().toStdString());
    lblVerdict->setText(QString("Вердикт: ") +
        (client->check(leMessage->text(), r, s) ? "подпись подтверждена" : "подпись ложна"));
}

void ElgamalLoader::done(QJsonObject& json) {
    BigInt p(json["p"].toString().toStdString());
    BigInt g(json["g"].toString().toStdString());
    BigInt x(json["x"].toString().toStdString());
    BigInt y(json["y"].toString().toStdString());
    BigInt k(json["k"].toString().toStdString());
    BigInt k_1(json["k_1"].toString().toStdString());
    task->setElgamal(p, g, x, y, k, k_1);
}

size_t hs(QString s) {
    return std::hash<std::string>{}(s.toStdString());
}

void ElgamalClient::generate(const QString &m, BigInt &r, BigInt &s) {
    size_t hm = hs(m);
    r = BigInt::modPow(g, k, p);
    s = ((hm - x * r) * k_1) % (p - 1);
}

```

```
}
```

```
bool ElgamalClient::check(const QString &m, const BigInt &r, const BigInt &s) {  
    bool result = 0 < r && r < p && 0 < s && s < (p - 1);  
    if(result) {  
        size_t hm = hs(m);  
        BigInt left = BigInt::mod2Pow(y, r, r, s, p);  
        BigInt right = BigInt::modPow(g, hm, p);  
        result = left == right;  
    }  
    return result;  
}
```