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-Гамаля»

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 несимметричных алгоритмов шифрования, на примере использования алгоритма Диффи-Хеллмана и Эль-Гамаля, а так же по применению электронных подписей на примере метода Эль-Гамаля.

**ЗАДАНИЕ**

**Вариант №14**. Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамаля, 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 «зашиты» 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 mod n

По получении А партнер P2 генерирует случайное или псевдослучайное простое число у и посылает P2 вычисленное значение B = qy mod n

Партнер P1, получив В, вычисляет Kx = Bx mod n, а партнер P2 вычисляет Ky = Ay mod n. Алгоритм гарантирует, что числа Ky и Kx равны и могут быть использованы в качестве секретного ключа для шифрования. Ведь даже перехватив числа А и В, трудно вычислить Kx или Ky.

Например, по вычисленным Kx =Ky=K абоненты могут зашифровать сообщение M=123 по следующему алгоритму: к каждому символу сообщения M добавить K => сообщение С=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 mod n = 1, а B = q^y mod n = 2, то вычислив Kx = B^x mod n и Ky = A^y mod n получим Kx= Ky=1. Зашифруем приведенное выше сообщение M=123 по приведенному выше алгоритму => сообщение С=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-Гамаля:**

Алгоритм Эль-Гамаля 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 mod p

Общедоступными ключами являются y, g и p, а секретным ключом является х. Для подписи сообщения M выбирается случайное число k, которое является простым по отношению к p-1. После этого вычисляется a = gk mod p. Далее из уравнения M = (xa + kb) mod (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 \* a^b mod p = g^M mod p.

Пример:

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

Вычислим y = g^x mod p = 3.

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

M = (xa + kb) mod (p - 1)

5 = (8\*6 + 9 \* b) mod 10

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

**Шифрование Эль-Гамаля:**

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

a = g^k mod p

b = y^kM dom 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 mod p.

Пример:

Вернемся к предыдущему примеру:

a=6, b = y^kM mod p = 3.

Расшифруем: M = b/a^x mod 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-Гамаля, в браузер.

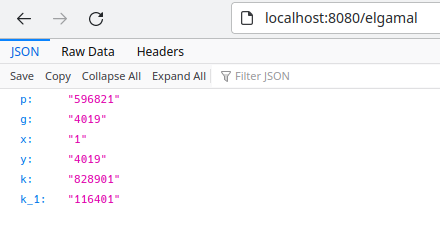
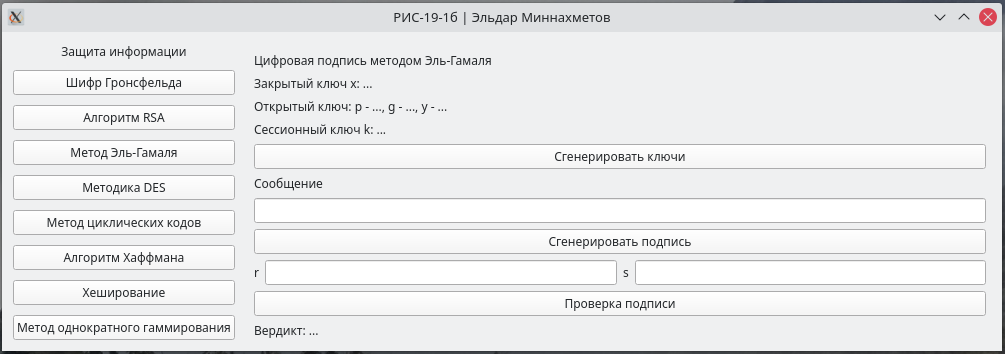
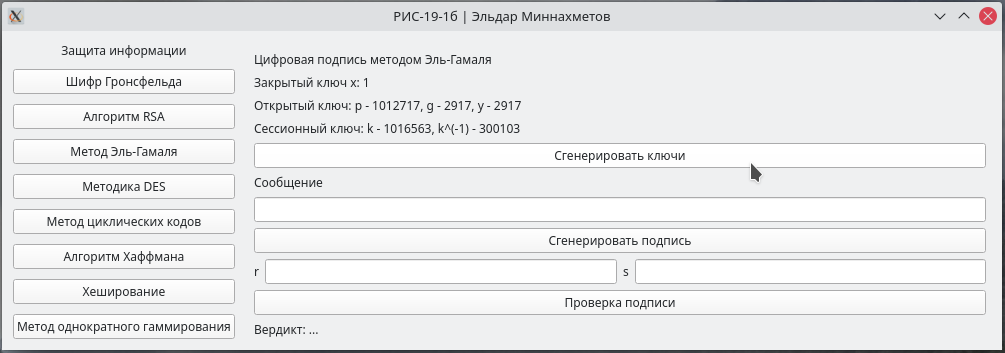


Рисунок 1 – Вывод результата запроса к серверу в браузере

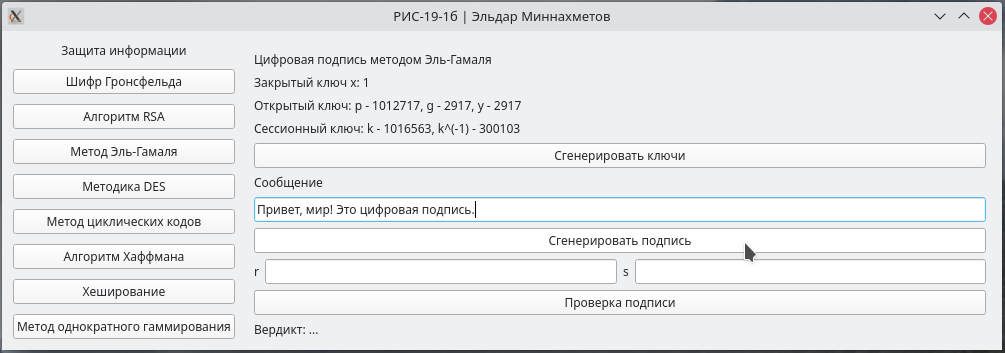
На рисунке 2 представлен графический интерфейс десктопного приложения.

Рисунок 2 – Графический пользовательский интерфейс

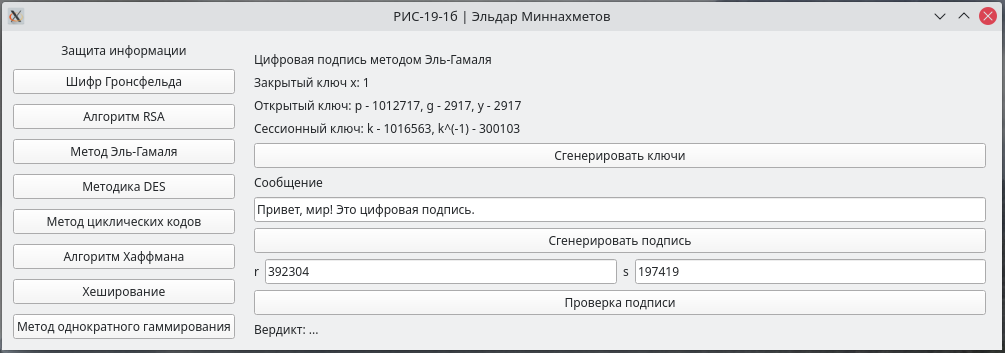
После нажатия на кнопку «Сгенерировать ключи» десктопное приложение отправляет запрос ключей. Полученные ключи вводятся в соответствующие поля на форме. Результат генерации представлен на рисунке 3.

Рисунок 3 – Результат генерации ключей

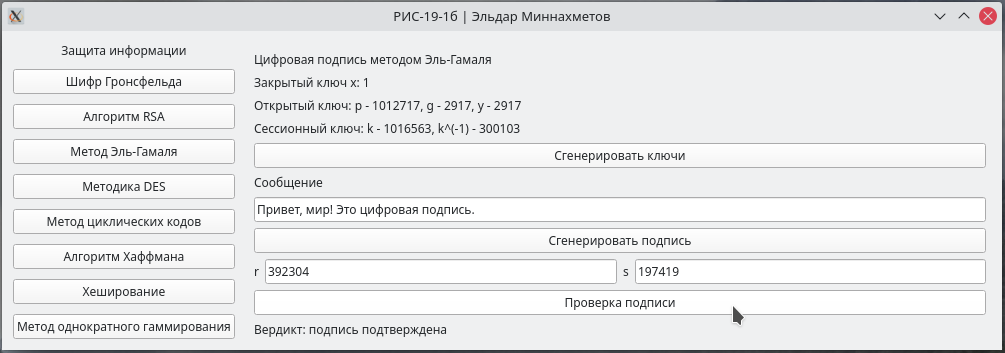
Далее необходимо ввести сообщение, у которого будет вычисляться пара (r, s).

Рисунок 4 – Ввод текстового сообщения

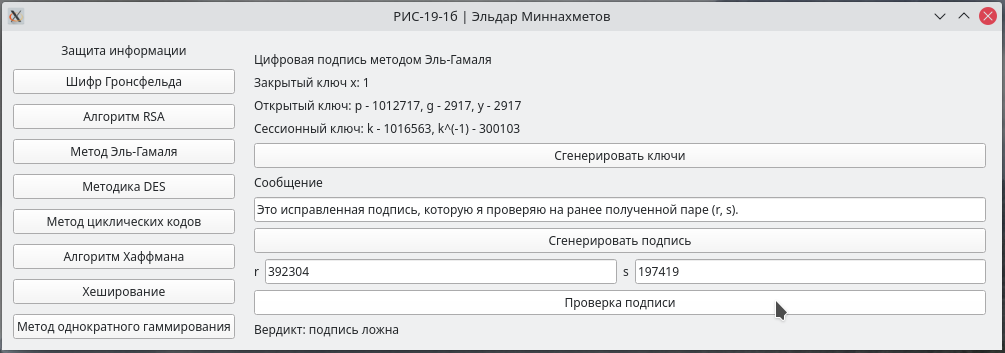
По нажатию на кнопку «Сгенерировать подпись» выполняется генерация пары (r, s) и затем она записывается в соответствующие поля, как показано на рисунке 5.

 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.RequestParam  
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-Гамаля");  
 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;  
}