Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

кафедра Информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Шифрование данных методом RSA

БГУИР КП 1-53 01 07 10724 ПЗ

Студент: гр. 953501 Жамойдик Н.С.

Руководитель: ассистент кафедры информатики Удовин И.А.

Минск 2020

Содержание

Введение…………………………………………………………………………

1. Анализ предметной области…………………………………………………

1.1 История возникновения……………………………………………...

1.2 Постановка задачи……………………………………………………

2. Описание алгоритма………………………………………………………….

2.1 Введение……………………………………………….……………...

2.2 Алгоритм………………………………………………………………

2.3 Шифрование и дешифрование……………………………………….

2.4 Цифровая подпись…………………………………………………….

3. Методика работы с программой……………………………………………..

Заключение……………………………………………………………………….

Литература………………………………………………………………………..

Исходный код приложения……………………………………………………...

**Введение**

Криптогра́фия (от [др.-греч.](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B5%D0%B2%D0%BD%D0%B5%D0%B3%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Древнегреческий язык) κρυπτός «скрытый» + γράφω «пишу») — наука о методах обеспечения [конфиденциальности](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C) (невозможности прочтения информации посторонним), [целостности данных](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) (невозможности незаметного изменения информации), [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) (проверки подлинности авторства или иных свойств объекта).

Изначально криптография изучала методы шифрования информации — обратимого преобразования открытого (исходного) текста на основе секретного алгоритма или [ключа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) в шифрованный текст (шифротекст). Традиционная криптография образует раздел [симметричных криптосистем](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B), в которых зашифровывание и расшифровывание проводится с использованием одного и того же секретного ключа. Помимо этого раздела современная криптография включает в себя [асимметричные криптосистемы](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), системы [электронной цифровой подписи](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C) (ЭЦП), [хеш-функции](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [управление ключами](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BB%D1%8E%D1%87%D0%B0%D0%BC%D0%B8), [получение скрытой информации](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B9_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8), [квантовую криптографию](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F).

Криптография не занимается защитой от [обмана](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D0%B0%D0%BD), [подкупа](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BA%D1%83%D0%BF) или [шантажа](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%BD%D1%82%D0%B0%D0%B6) законных абонентов, кражи ключей и других угроз информации, возникающих в защищённых системах [передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Криптография — одна из старейших наук, [её история](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%B8) насчитывает несколько тысяч лет.

Для современной криптографии характерно использование открытых алгоритмов шифрования, предполагающих использование вычислительных средств. Известно более десятка проверенных [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) шифрования, которые при использовании ключа достаточной длины и корректной реализации алгоритма [криптографически стойки](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%BE%D0%B9%D0%BA%D0%BE%D1%81%D1%82%D1%8C" \o "Криптографическая стойкость). Распространённые алгоритмы:

* симметричные [DES](https://ru.wikipedia.org/wiki/DES), [AES](https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard), [ГОСТ 28147-89](https://ru.wikipedia.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2_28147-89), [Camellia](https://ru.wikipedia.org/wiki/Camellia_(%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)" \o "Camellia (алгоритм)), [Twofish](https://ru.wikipedia.org/wiki/Twofish" \o "Twofish), [Blowfish](https://ru.wikipedia.org/wiki/Blowfish" \o "Blowfish), [IDEA](https://ru.wikipedia.org/wiki/IDEA), [RC4](https://ru.wikipedia.org/wiki/RC4) и др.;
* асимметричные [RSA](https://ru.wikipedia.org/wiki/RSA) и [Elgamal](https://ru.wikipedia.org/wiki/Elgamal" \o "Elgamal) (*Эль-Гамаль*);
* [хеш-функций](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [MD4](https://ru.wikipedia.org/wiki/MD4), [MD5](https://ru.wikipedia.org/wiki/MD5), [MD6](https://ru.wikipedia.org/wiki/MD6), [SHA-1](https://ru.wikipedia.org/wiki/SHA-1), [SHA-2](https://ru.wikipedia.org/wiki/SHA-2), [ГОСТ Р 34.11-2012 («Стрибог»)](https://ru.wikipedia.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2_%D0%A0_34.11-2012) .

Криптографические методы стали широко использоваться частными лицами в электронных коммерческих операциях, телекоммуникациях и многих других средах.

* 1. **Анализ предметной области**
  2. История возникновения

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — [криптографический алгоритм с открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), основывающийся на [вычислительной сложности](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [задачи факторизации](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) больших целых чисел.

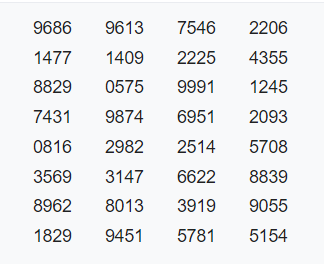
Криптосистема RSA стала первой системой, пригодной и для [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), и для [цифровой подписи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C). Алгоритм используется в большом числе криптографических приложений,

включая [PGP](https://ru.wikipedia.org/wiki/PGP), [S/MIME](https://ru.wikipedia.org/wiki/S/MIME), [TLS](https://ru.wikipedia.org/wiki/TLS)/[SSL](https://ru.wikipedia.org/wiki/SSL), [IPSEC](https://ru.wikipedia.org/wiki/IPSEC)/[IKE](https://ru.wikipedia.org/wiki/IKE) и других.

Опубликованная в ноябре 1976 года статья [Уитфилда Диффи](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B8,_%D0%A3%D0%B8%D1%82%D1%84%D0%B8%D0%BB%D0%B4" \o "Диффи, Уитфилд) и [Мартина Хеллмана](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD,_%D0%9C%D0%B0%D1%80%D1%82%D0%B8%D0%BD) «Новые направления в криптографии» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *New Directions in Cryptography*)[[4]](https://ru.wikipedia.org/wiki/RSA#cite_note-_08b07f1facd8ea80-4) перевернула представление о криптографических системах, заложив основы [криптографии с открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC). Разработанный впоследствии [алгоритм Диффи — Хеллмана](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B8%D1%84%D1%84%D0%B8_%E2%80%94_%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0) позволял двум сторонам получить общий секретный ключ, используя незащищенный канал связи. Однако этот алгоритм не решал проблему [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F). Без дополнительных средств пользователи не могли быть уверены, с кем именно они сгенерировали общий секретный ключ.

Изучив эту статью, трое учёных [Рональд Ривест](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%B2%D0%B5%D1%81%D1%82,_%D0%A0%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4_%D0%9B%D0%B8%D0%BD%D0%BD), [Ади Шамир](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%BC%D0%B8%D1%80,_%D0%90%D0%B4%D0%B8" \o "Шамир, Ади) и [Леонард Адлеман](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%BB%D0%B5%D0%BC%D0%B0%D0%BD,_%D0%9B%D0%B5%D0%BE%D0%BD%D0%B0%D1%80%D0%B4_%D0%9C%D0%B0%D0%BA%D1%81) из [Массачусетского технологического института](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82) (MIT) приступили к поискам математической функции, которая бы позволяла реализовать сформулированную [Уитфилдом Диффи](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B8,_%D0%A3%D0%B8%D1%82%D1%84%D0%B8%D0%BB%D0%B4" \o "Диффи, Уитфилд) и [Мартином Хеллманом](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD,_%D0%9C%D0%B0%D1%80%D1%82%D0%B8%D0%BD) модель [криптографической системы с открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC). После работы над более чем 40 возможными вариантами им удалось найти алгоритм, основанный на различии в том, насколько легко находить большие простые числа и насколько сложно раскладывать на множители произведение двух больших простых чисел, получивший впоследствии название RSA. Система была названа по первым буквам фамилий её создателей.

В августе [1977 года](https://ru.wikipedia.org/wiki/1977_%D0%B3%D0%BE%D0%B4) в колонке «Математические игры» Мартина Гарднера в журнале [Scientific American](https://ru.wikipedia.org/wiki/Scientific_American" \o "Scientific American) с разрешения Рональда Ривеста[[5]](https://ru.wikipedia.org/wiki/RSA" \l "cite_note-5) появилось первое описание криптосистемы RSA[[6]](https://ru.wikipedia.org/wiki/RSA#cite_note-_aa6c7f30194f3484-6). Читателям также было предложено дешифровать английскую фразу, зашифрованную описанным алгоритмом:



В качестве открытых параметров системы были использованы числа n=1143816...6879541 (129 десятичных знаков, 425 [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82), также известно как [RSA-129](https://ru.wikipedia.org/wiki/RSA-%D1%87%D0%B8%D1%81%D0%BB%D0%B0#RSA-129)) и e=9007. За расшифровку была обещана награда в 100 долларов США. По заявлению Ривеста, для факторизации числа потребовалось бы более 40 квадриллионов лет. Однако чуть более, чем через 15 лет, 3 сентября [1993 года](https://ru.wikipedia.org/wiki/1993_%D0%B3%D0%BE%D0%B4) было объявлено о запуске проекта [распределённых вычислений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) с координацией через электронную почту по нахождению сомножителей числа RSA-129 и решению головоломки. На протяжении полугода более 600 добровольцев из 20 стран жертвовали процессорное время 1600 машин (три из которых были факс-машинами). В результате были найдены простые множители и расшифровано исходное сообщение, которое представляет собой фразу «[THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE](https://ru.wikipedia.org/w/index.php?title=THE_MAGIC_WORDS_ARE_SQUEAMISH_OSSIFRAGE&action=edit&redlink=1) ([англ.](https://en.wikipedia.org/wiki/The_Magic_Words_are_Squeamish_Ossifrage))» («Волшебные слова — это брезгливый [ягнятник](https://ru.wikipedia.org/wiki/%D0%AF%D0%B3%D0%BD%D1%8F%D1%82%D0%BD%D0%B8%D0%BA)»)[[8]](https://ru.wikipedia.org/wiki/RSA#cite_note-8)[[9]](https://ru.wikipedia.org/wiki/RSA#cite_note-9). Полученную награду победители пожертвовали в [фонд свободного программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BD%D0%B4_%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F).

После публикации Мартина Гарднера полное описание новой криптосистемы любой желающий мог получить, выслав по почте запрос Рональду Ривесту с приложенным конвертом с обратным адресом и марками на 35 центов. Полное описание новой криптосистемы было опубликовано в журнале «[Communications of the ACM](https://ru.wikipedia.org/wiki/Communications_of_the_ACM" \o "Communications of the ACM)» в феврале 1978 года.

Заявка на патент была подана 14 декабря 1977 года, в качестве владельца был указан MIT. Патент 4405829 был выдан 20 сентября [1983 года](https://ru.wikipedia.org/wiki/1983_%D0%B3%D0%BE%D0%B4), а 21 сентября 2000 года срок его действия истёк. Однако за пределами США у изобретателей патента на алгоритм не было, так как в большинстве стран его необходимо было получить до первой публикации.

* 1. Постановка задачи

Основной задачей курсовой работы является разработка приложения, способного зашифровать данные, расшифровать данные, создать цифровую подпись. В качестве среды разработки приложения была выбрана Microsoft Visual Studio. Язык программирования – C#.

**2. Описание алгоритма**

* 1. Введение

Как и было сказано ранее, RSA является методом шифрования с открытым ключом. Это значит, что данные будут зашифровываться одним ключом (открытым), который может быть известен каждому, а расшифровываться данные будут другим ключом (закрытым), который будет известен только вам.

Криптографические системы с открытым ключом используют так называемые [односторонние функции](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%BD%D1%8F%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F), которые обладают следующим свойством:

* если известно {\displaystyle x}*x*, то *F(x)* {\displaystyle f(x)} вычислить относительно просто;
* если известно {\displaystyle y=f(x)}*y* = *F(x)* , то для вычисления {\displaystyle x}*x* нет простого (эффективного) пути.

Под односторонностью понимается не теоретическая однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность [задачи факторизации](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) произведения двух больших простых чисел. Для шифрования используется операция [возведения в степень по модулю](https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E) большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять [функцию Эйлера](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от данного большого числа, для чего необходимо знать разложение числа на простые множители.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом, так и закрытым ключом. В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными, то есть:

“Для всяких допустимых пар открытого и закрытого ключей (*p,s*) существуют соответствующие функции шифрования (*x*) и расшифрования (x) такие, что всякое сообщение *m* принадлежит *m* **∈** *M*, где *M* – множество допустимых сообщений, *m* = = .”

* 1. Алгоритм

Ключи шифрование генерируются следующим образом:

1. Выбираются 2 случайных простых числа *p* и *q*;
2. Вычисляется их произведение *n = p\*q*, которое называется модулем;
3. Вычисляется значение функции Эйлера от числа n
4. Выбирается целое число {\displaystyle e}*e* ({\displaystyle 1<e<\varphi (n)}1<*e*<, [взаимно простое](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) со значением функции {\displaystyle \varphi (n)}
5. Вычисляется число *d*, мультипликативно обратное числу *e* по модулю
6. Пара (*e, n*) публикуется в качестве открытого ключа шифрования
7. Пара (*d, n*) является закрытым ключом шифрования и держится в секрете

3.3 Шифрование и дешифрование

**Шифрование**

* Берётся открытый ключ (*e, n*)
* Берётся открытый текст *m*
* Сообщение шифруется с использованием открытого ключа

*с* =

**Дешифрование**

* Принимается зашифрованное сообщение *c*
* Берётся свой закрытый ключ (*d, n*)
* Сообщение дешифруется с использование закрытого ключа

Данная схема на практике не используется по причине того, что она не является практически надёжной. Действительно, односторонняя функция *E(m)* является детерминированной — при одних и тех же значениях входных параметров выдаёт одинаковый результат. Это значит, что не выполняется необходимое условие практической надёжности шифра.

* 1. Цифровая подпись

Система RSA может использоваться не только для шифрования, но и для [цифровой подписи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C).

**Алгоритм отправителя**

* Взять открытый текст *m*
* Создать цифровую подпись *s* с помощью своего секретного ключа (d,n)
* Передать пару (*m,s*)

**Алгоритм получателя**

* Принять пару (*m,s*)
* Взять открытый ключ отправителя (*e,n*)
* Вычислить прообраз сообщения из подписи
* Проверить подлинность подписи, сравнив *m* и *m’*

Поскольку [цифровая подпись](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C) обеспечивает как [аутентификацию](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) автора сообщения, так и подтверждение целостности содержимого подписанного сообщения, она служит аналогом подписи, сделанной от руки в конце рукописного документа.

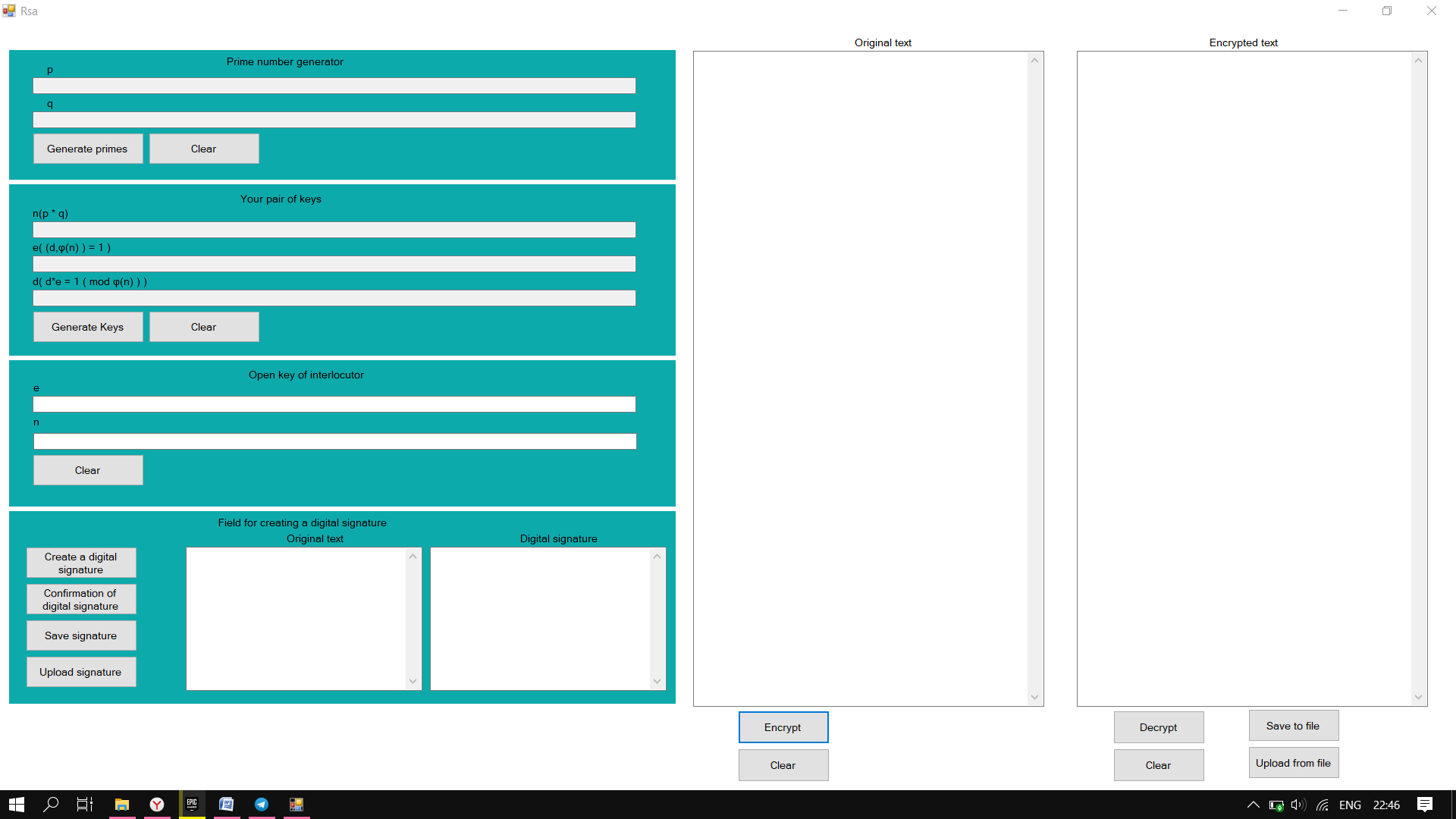
Важное свойство цифровой подписи заключается в том, что её может проверить каждый, кто имеет доступ к открытому ключу её автора. Один из участников обмена сообщениями после проверки подлинности цифровой подписи может передать подписанное сообщение ещё кому-то, кто тоже в состоянии проверить эту подпись.

Заметим, что подписанное сообщение *{\displaystyle m}m* не [зашифровано](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Оно пересылается в исходном виде и его содержимое не защищено от нарушения конфиденциальности. Путём совместного применения представленных выше схем шифрования и цифровой подписи в системе RSA можно создавать сообщения, которые будут и зашифрованы, и содержать цифровую подпись. Для этого автор сначала должен добавить к сообщению свою цифровую подпись, а затем — зашифровать получившуюся в результате пару (состоящую из самого сообщения и подписи к нему) с помощью открытого ключа, принадлежащего получателю. Получатель расшифровывает полученное сообщение с помощью своего секретного ключа. Если проводить аналогию с пересылкой обычных бумажных документов, то этот процесс похож на то, как если бы автор документа поставил под ним свою печать, а затем положил его в бумажный конверт и запечатал, с тем чтобы конверт был распечатан только тем человеком, кому адресовано сообщение.

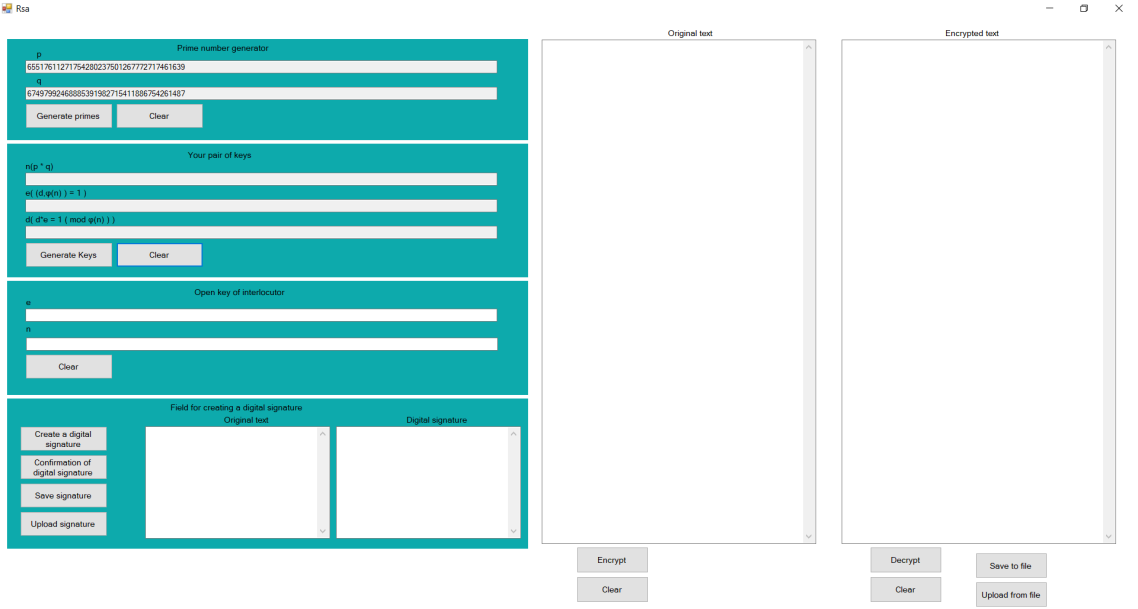
**3.Методика работы с программой**

Интерфейс программы состоит из нескольких блоков:

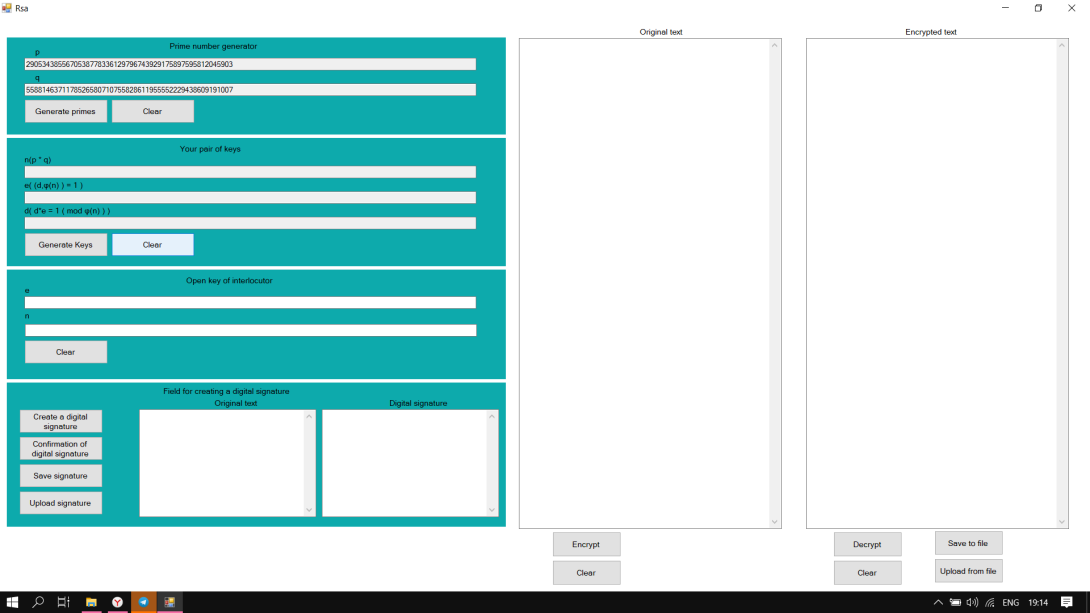
* Поле генерации простых чисел
* Поле генерации собственных ключей
* Поле открытого ключа собеседника
* Поле создания и подтверждения цифровой подписи
* Поле текста предназначенного для шифрования
* Поле текста предназначенного для дешифрования



Изначально собеседники генерируют простые числа.

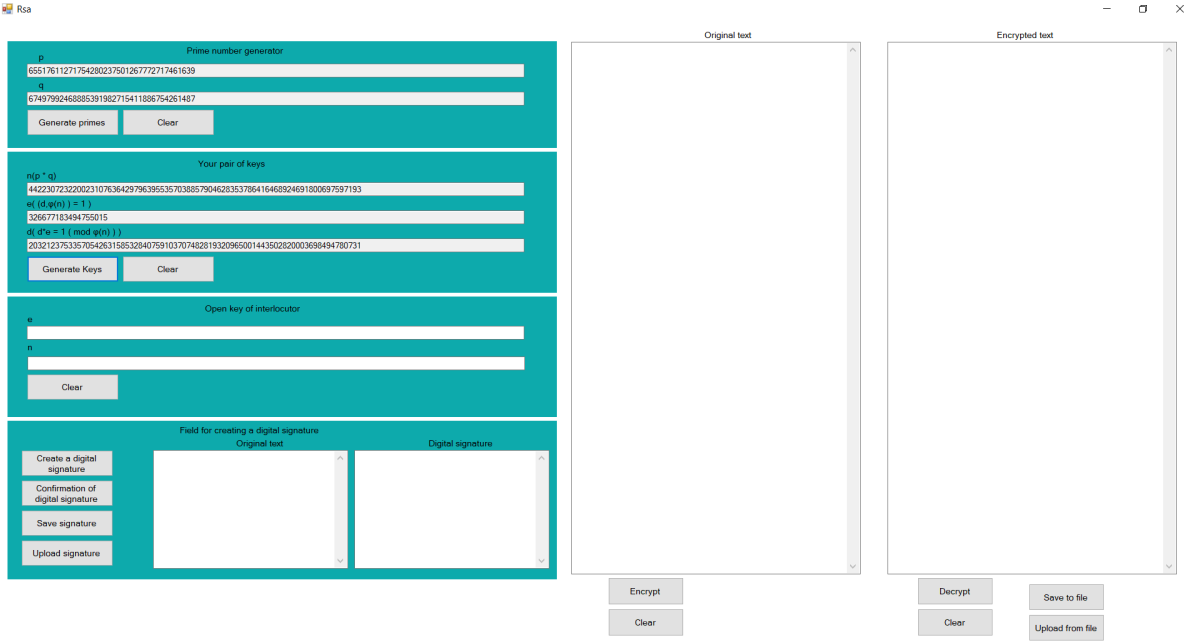


Собеседник A

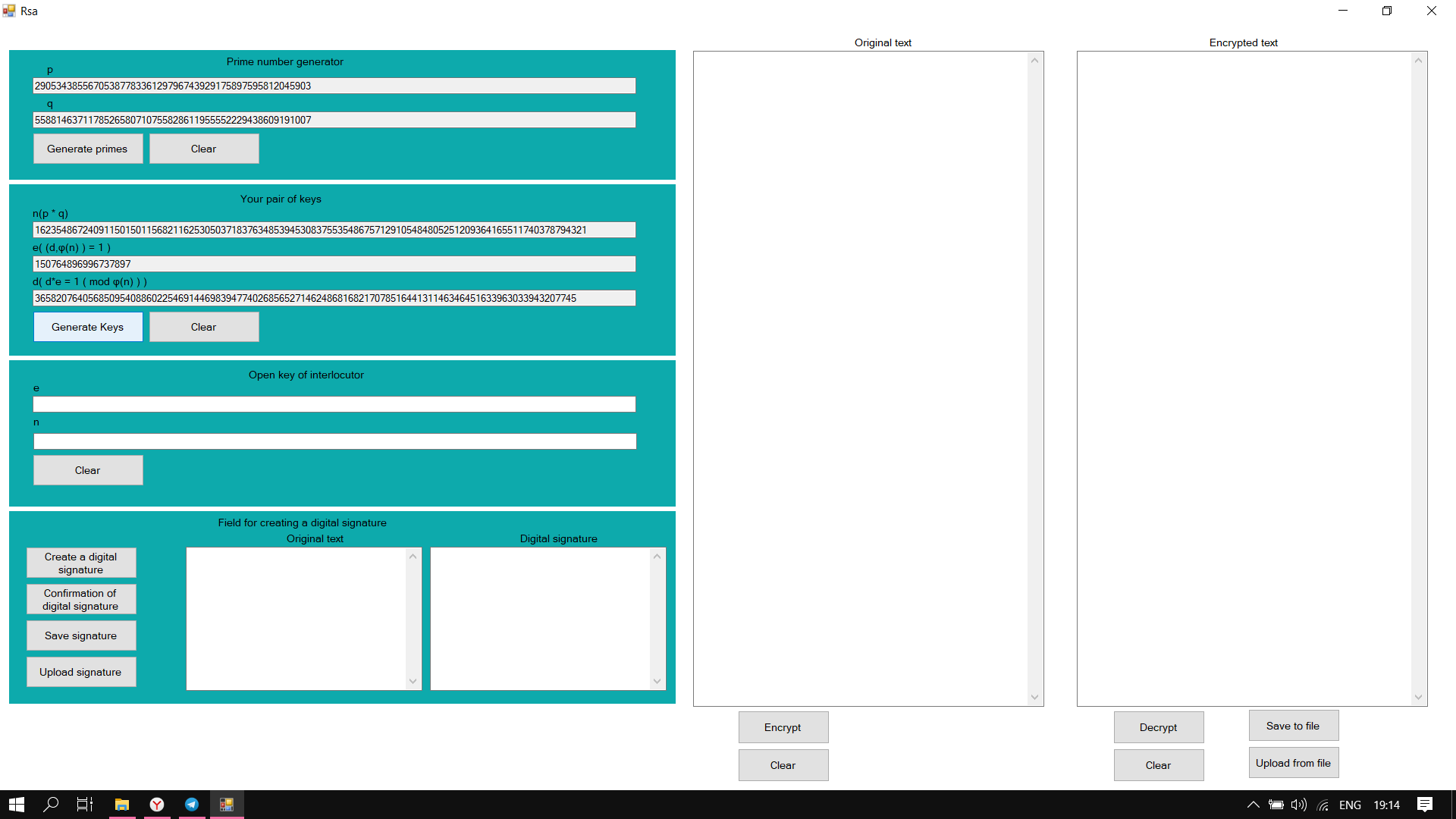


Собеседник Б

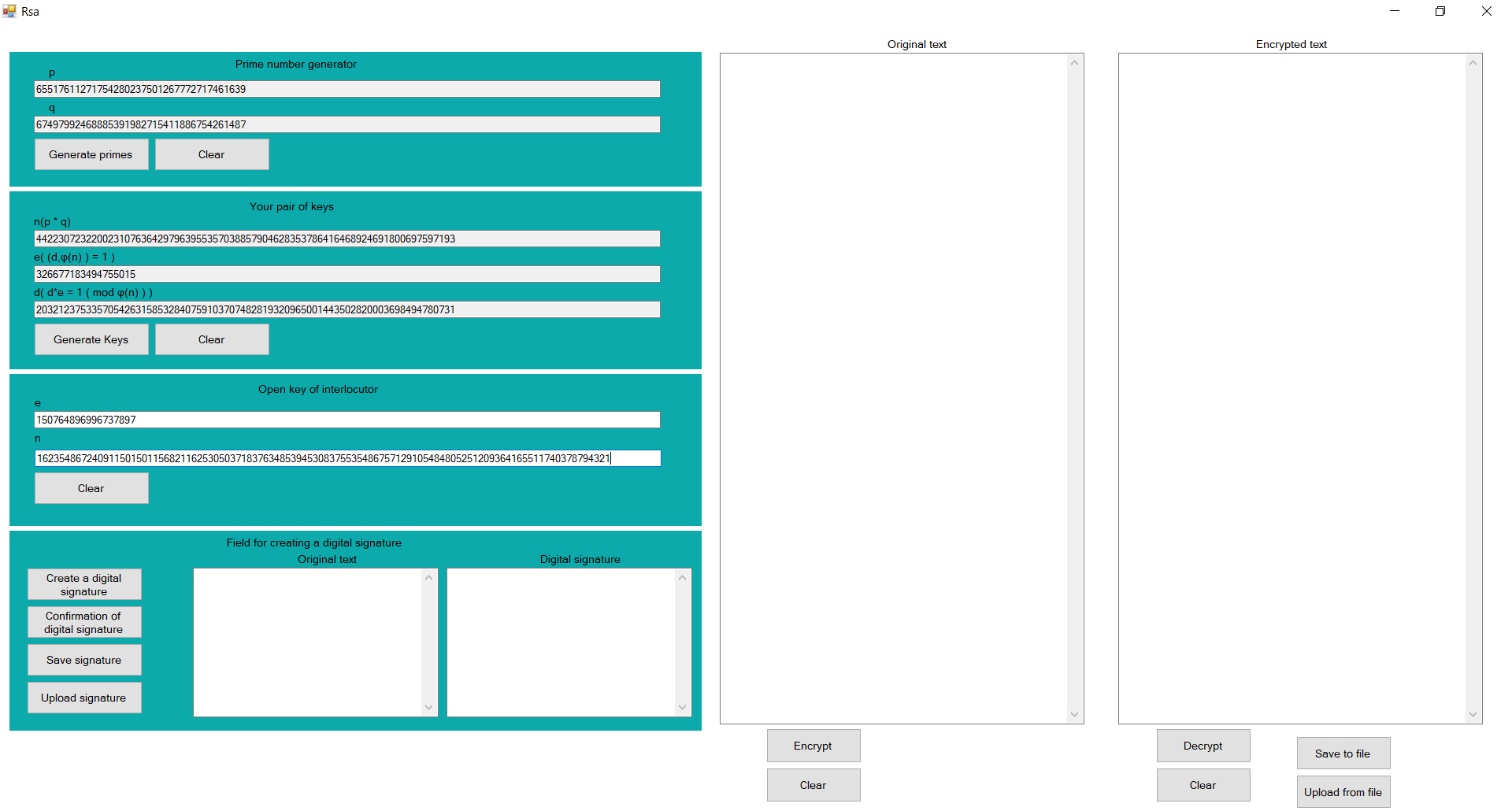
Затем собеседники генерируют свои пары открытого и закрытого ключей (поля с простыми числами и ключами имеют модификатор readonly, что значит что пользователь не может ввести числа сам)



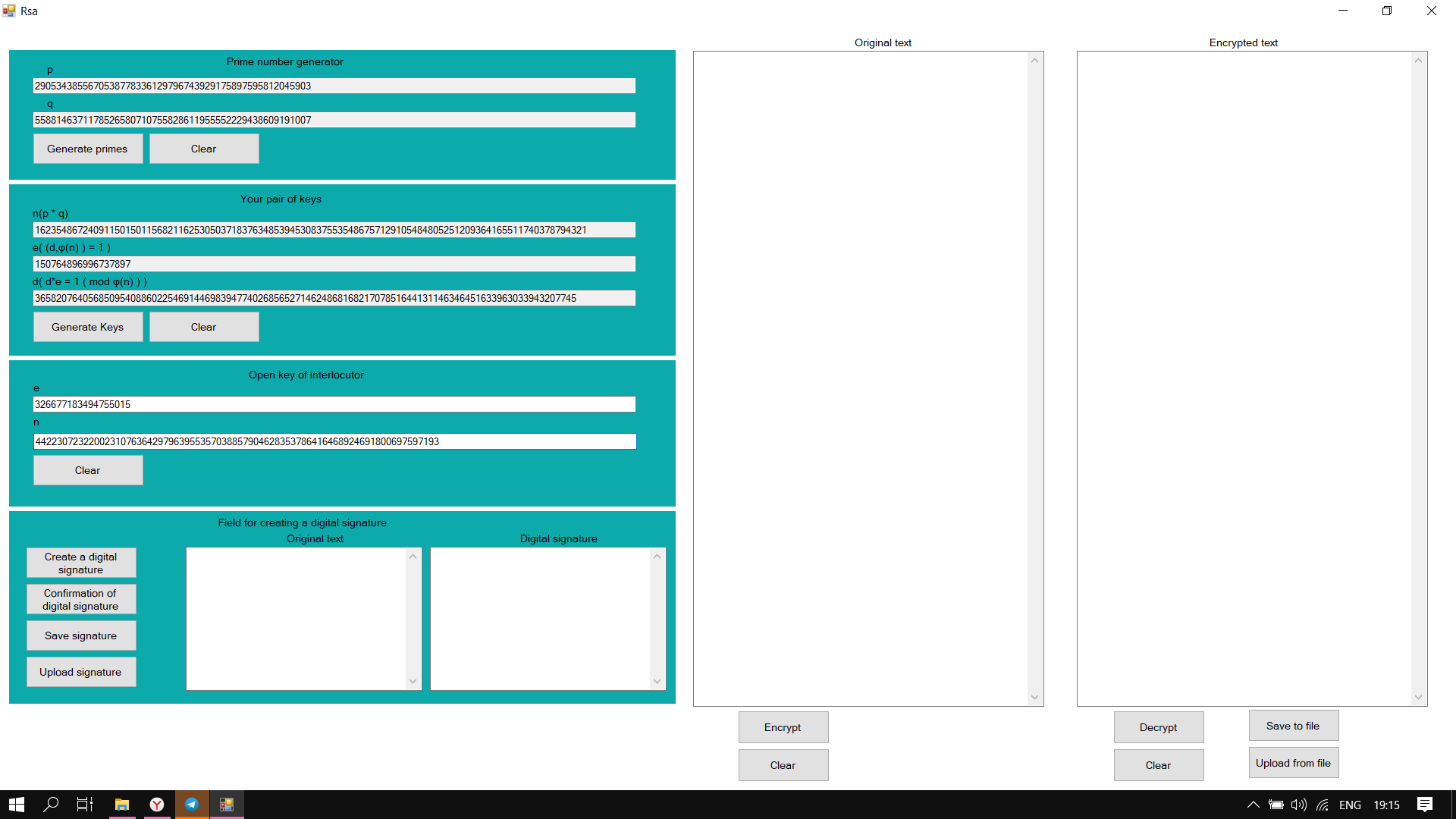
Собеседник А



Собеседник Б

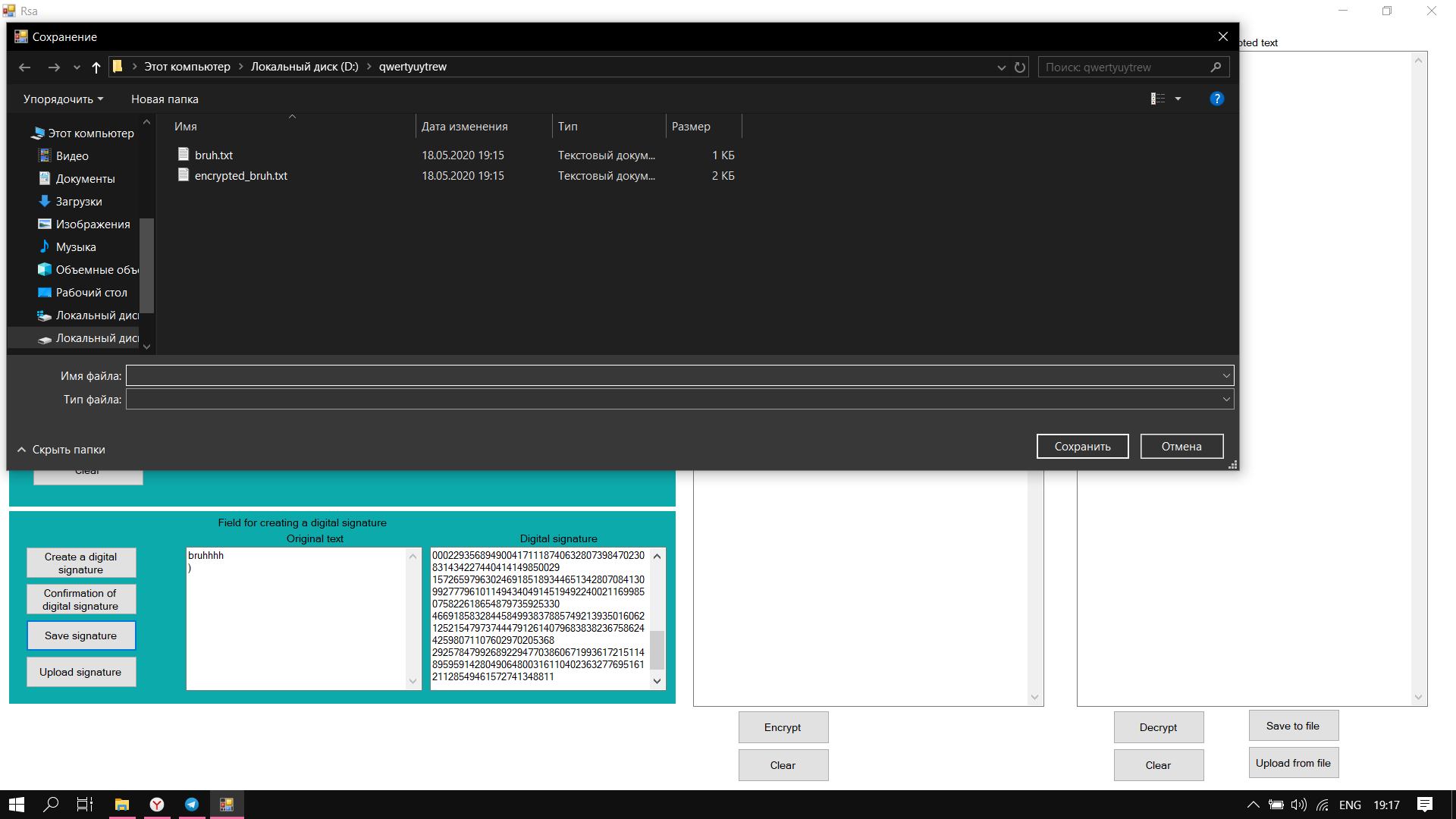
Затем обмениваются открытыми ключами и заполняют соответствующие поля

Собеседник А



Собеседник Б

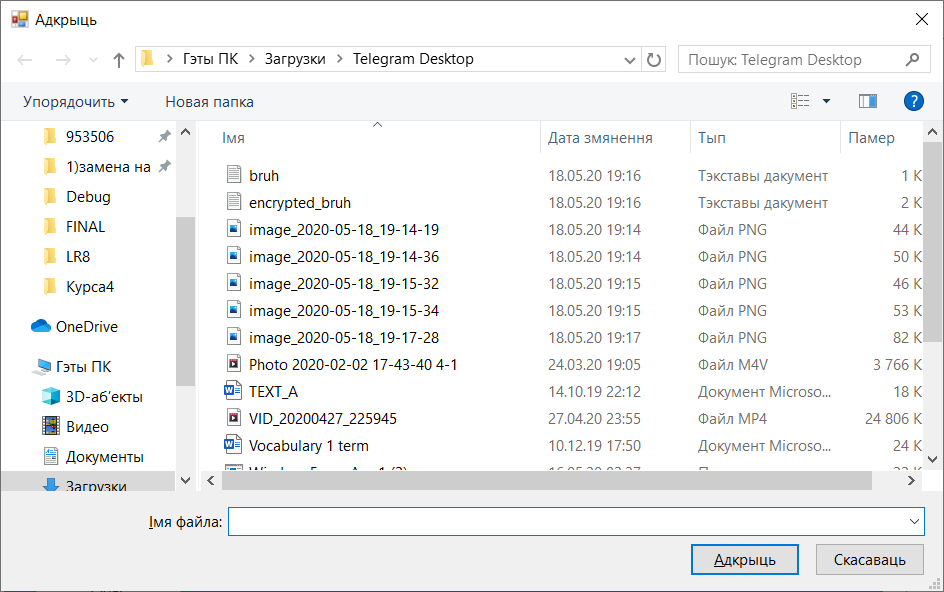
В целях безопасности Собеседник Б решил отправить Собеседнику А цифровую подпись. Порядок действий следующий:

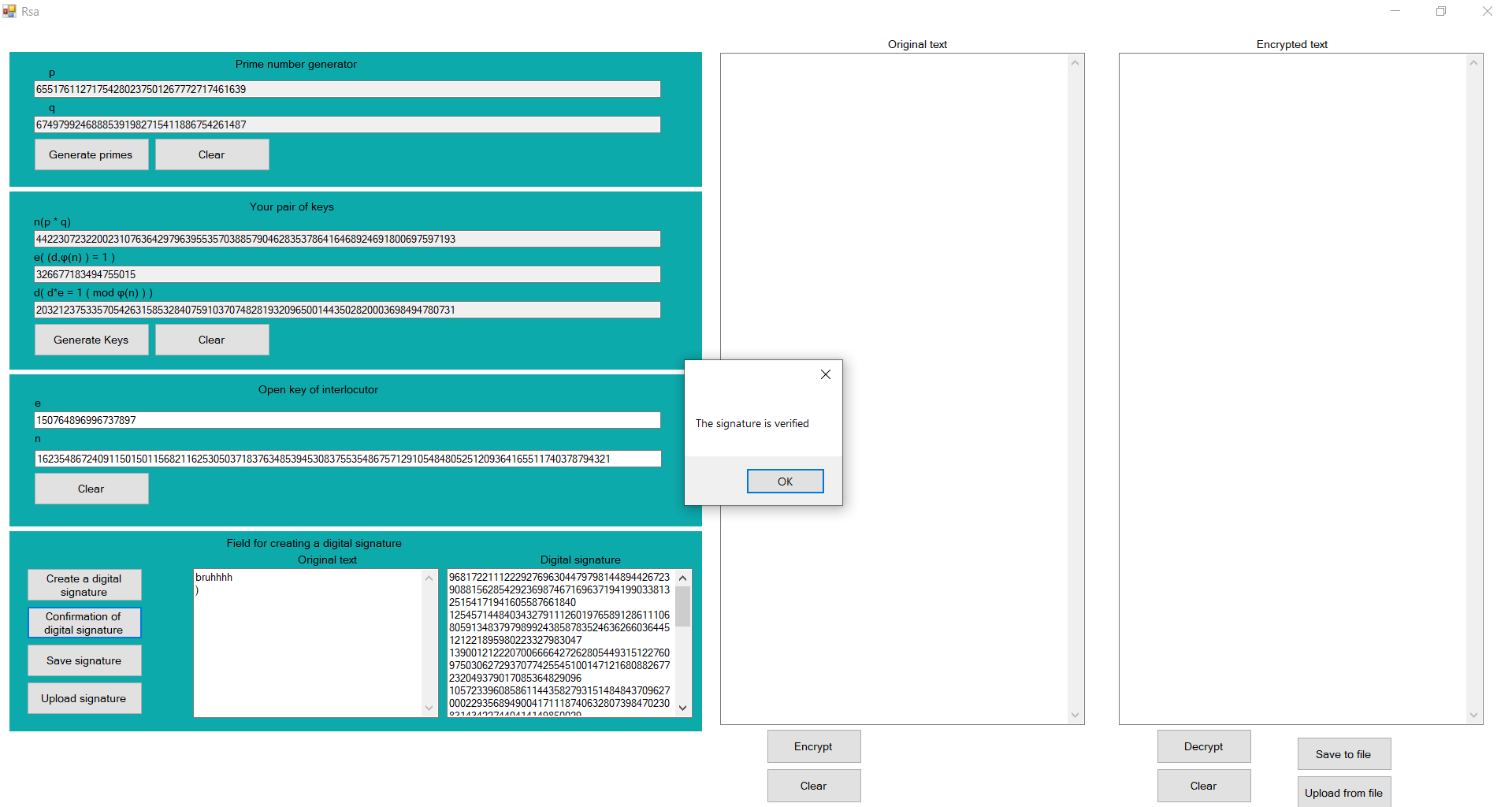
* Вводим текст в поле “Original text”
* Нажимаем кнопку “Create a digital signature”
* Затем текст и шифровка текста может быть отправлена путём копирования из полей ввода или, нажав на кнопку “Save signature”, мы можем сохранить и текст и шифровку в файлы.

Чтобы проверить цифровую подпись, собеседник A должен выполнить следующий порядок:

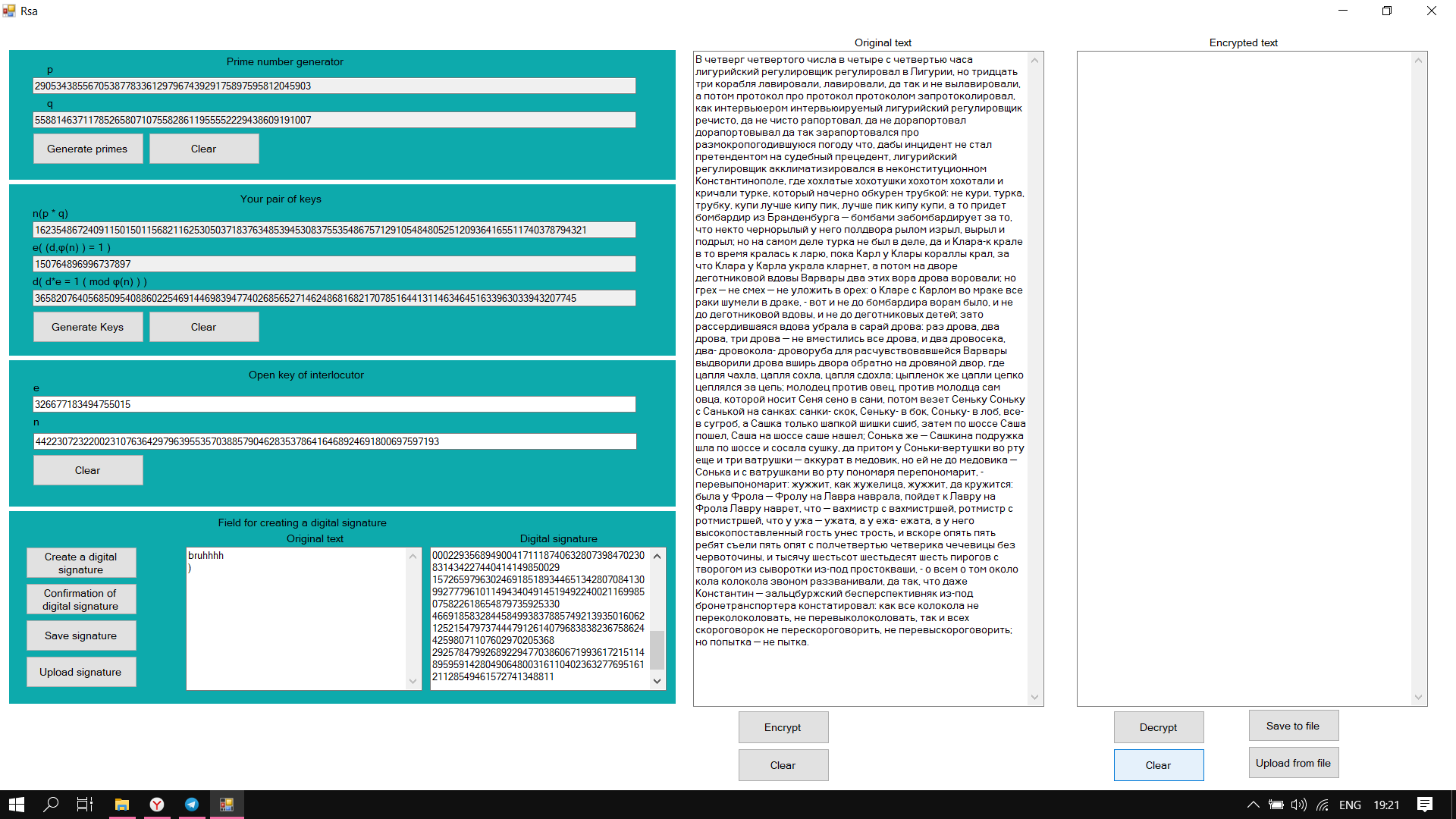
* Либо вставить скопированные тексты в соответствующие поля, либо нажать на кнопку “Upload signature” и выбрать соответствующие файлы.
* Нажать на кнопку “Confirmation of digital signature”

Результатом нажатия кнопки будет уведомление о соответствии или несоответствии подписи

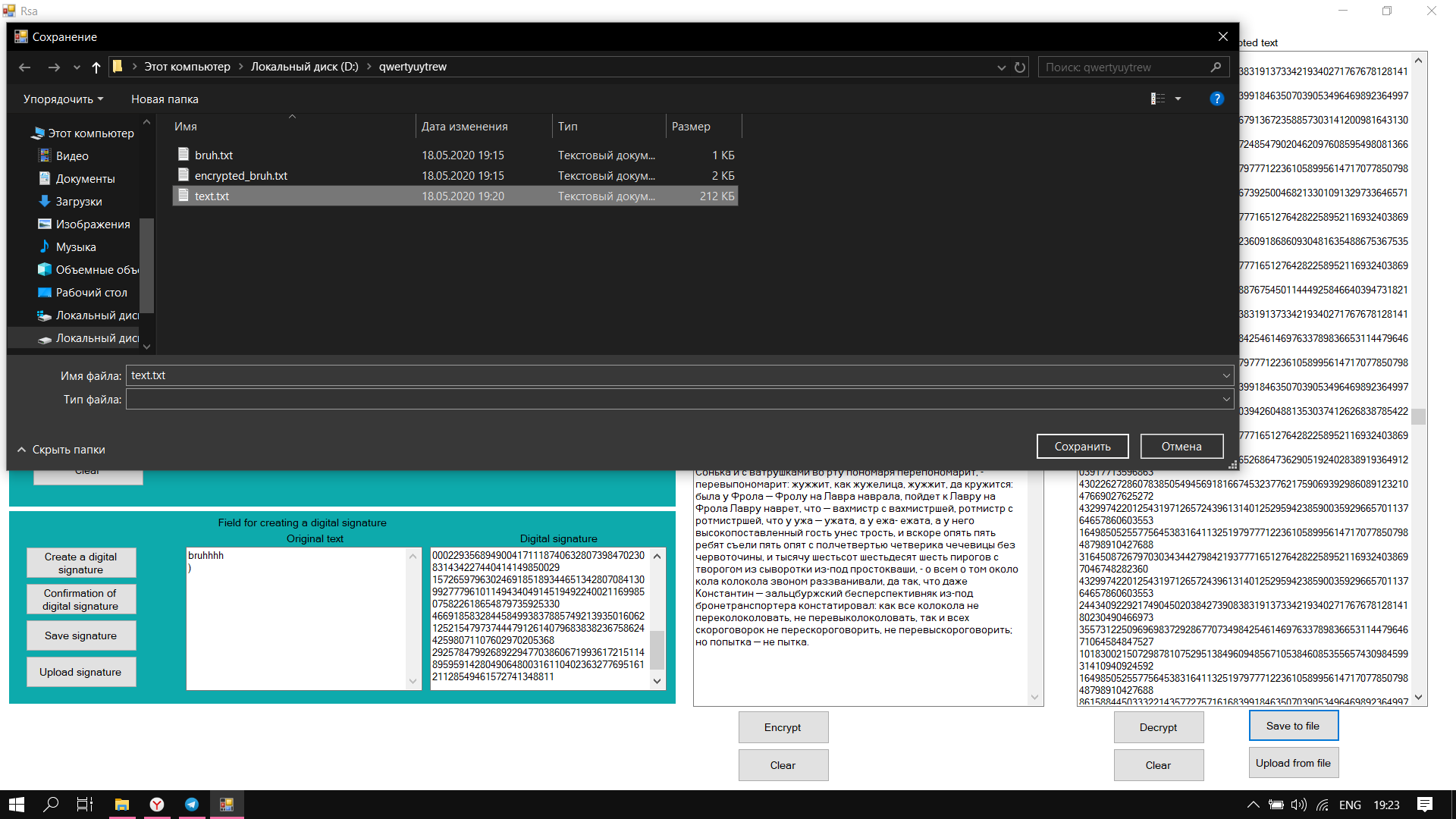




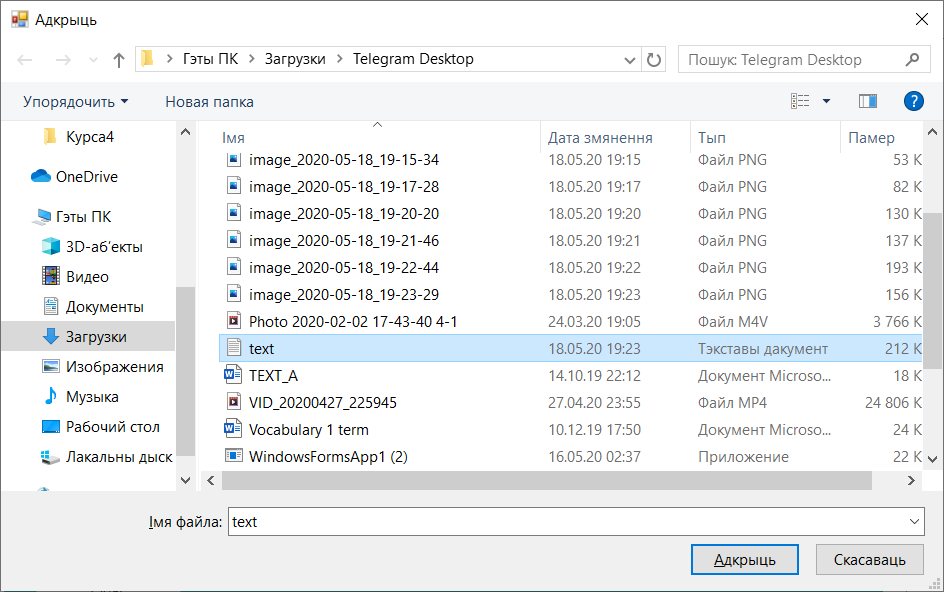
Собеседник Б хочет отправить Собеседнику А сообщение. Он вводит исходный текст в поле “Original text”, после чего нажимает на кнопку “Encrypt” и получает зашифрованный текст. Зашифрованные текст Собеседник Б может либо скопировать, либо сохранить в файл.

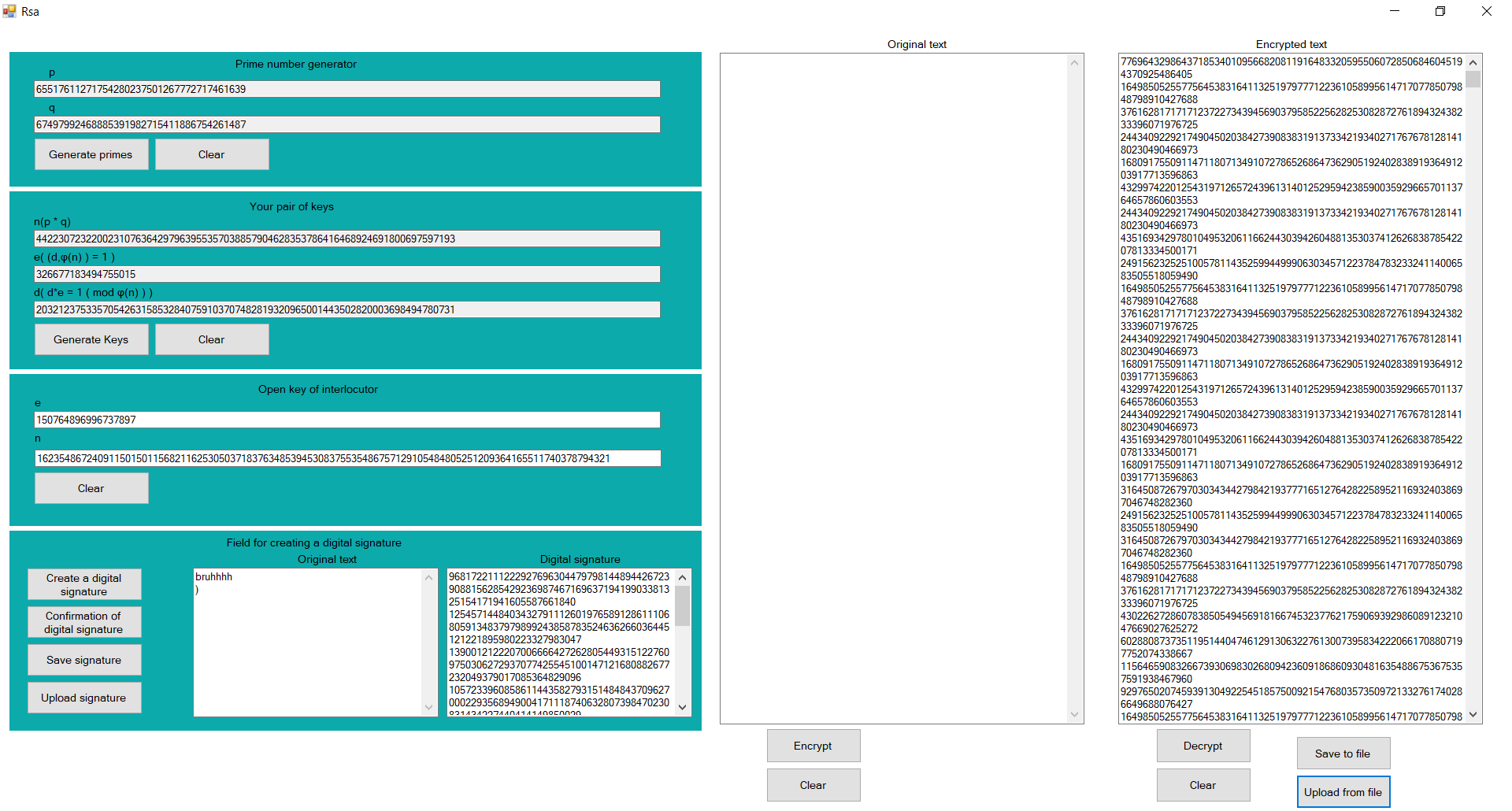






Собеседник А получает зашифрованный текст или файл с зашифрованным текстом. Текст вставляется в поле “Encrypted text” или загружается из файла после нажатия кнопки “Upload from file”. После нажатия кнопки “Decrypt” Собеседник А получает расшифрованный текст





****

**Заключение**

Метод шифрования RSA на сегодняшний день довольно популярен и много где используется. Но он все же имеет свой недостаток.

Метод имеет относительно простую математическую модель, но практическая модель показывает довольно низкую скорость шифрования

Из-за низкой скорости шифрования, сообщения обычно шифруют с помощью более производительных симметричных алгоритмов со случайным сеансовым ключом (например, [AES](https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard), [IDEA](https://ru.wikipedia.org/wiki/IDEA), [Serpent](https://ru.wikipedia.org/wiki/Serpent" \o "Serpent), [Twofish](https://ru.wikipedia.org/wiki/Twofish" \o "Twofish)), а с помощью RSA шифруют лишь этот ключ, таким образом реализуется [гибридная криптосистема](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D1%80%D0%B8%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Такой механизм имеет потенциальные уязвимости ввиду необходимости использовать [криптографически стойкий генератор псевдослучайных чисел](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8_%D1%81%D1%82%D0%BE%D0%B9%D0%BA%D0%B8%D0%B9_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB" \o "Криптографически стойкий генератор псевдослучайных чисел) для формирования случайного сеансового ключа симметричного шифрования.

**Список использованных источников**

1. Сайт <https://ru.wikipedia.org> – теория
2. Сайт <https://habr.com/> - информация о генерация простых чисел

**Исходный код приложения**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Numerics;

using System.Text.RegularExpressions;

using System.Runtime.InteropServices.WindowsRuntime;

using System.Runtime.Remoting;

using System.Text;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

//------------------------------------------------------------------------------Часть кода отвечающая за генерацию простых чисел-------------------------------------------------

private bool Check(int a)//Функция проверки числа на простотоу

{

for(int i=2; i<a; i++)

{

if (a % i == 0)

return false;

}

return true;

}

private static int LenghtOfBin(BigInteger a)//Функция определяющая кол-во бит в числе

{

int lenght = 0;

while (a != 0)

{

a = BigInteger.Divide(a, 2);

lenght++;

}

return lenght;

}

private void generatePrimes(List<int> primes, out BigInteger firstPrime, out BigInteger secondPrime)//Функция генерирующая 2 простых числа

{

firstPrime = 0;

secondPrime = 0;

List<BigInteger> mod3\_1 = new List<BigInteger>();

List<BigInteger> l = new List<BigInteger>();

BigInteger three = new BigInteger(3), two = new BigInteger(2);

for (int k = 0; k < primes.Count() - 1; k++)

{

BigInteger seed = new BigInteger(primes[k]);

BigInteger s2 = BigInteger.Multiply(seed, 2);

BigInteger r0;

BigInteger.DivRem(seed, three, out r0);

// Проверка на остаток = 1

if (r0 == BigInteger.One) mod3\_1.Add(seed);

for (int i = k + 1; i < primes.Count(); i++)

{

BigInteger p = new BigInteger(primes[i]);

BigInteger r;

BigInteger.DivRem(p, three, out r);

if (r == r0) continue;

else addIfPrime(p, seed, s2, two, l);

}

}

List<BigInteger> ps = l;

do

{

l = new List<BigInteger>();

int size = ps.Count();

for (int k = 0; k < size; k++)

{

BigInteger seed = ps[k];

BigInteger s2 = BigInteger.Multiply(seed, 2);

for (int i = 0; i < mod3\_1.Count(); i++)

addIfPrime(mod3\_1[i], seed, s2, two, l);

int n = 100000;

if (l.Count() > 0)

if (LenghtOfBin(l[0]) < 700) n = 10;

else if (LenghtOfBin(l[0]) < 800) n = 20;

else if (LenghtOfBin(l[0]) < 900) n = 40;

if (l.Count() > n) break;

}

ps = l;

Random rnd = new Random();

if (rnd.Next(1, 12) == 1)

{

while (true)

{

firstPrime = l[rnd.Next(0, l.Count() - 1)];

secondPrime = l[rnd.Next(0, l.Count() - 1)];

if (firstPrime.ToString().Length == secondPrime.ToString().Length)

{

if(firstPrime!=secondPrime)

return;

}

else

continue;

}

}

}

while (l.Count() > 0);

}

private static void addIfPrime(BigInteger a, BigInteger b, BigInteger b2, BigInteger two, List<BigInteger> l)

{

BigInteger a2 = BigInteger.Multiply(a, 2), fp = BigInteger.Multiply(b, a2), n = BigInteger.Add(fp, 1);

BigInteger r = new BigInteger();

if (BigInteger.Compare(a2, b) < 0)

r = BigInteger.ModPow(two, a2, n);

else if (BigInteger.Compare(a, b2) < 0)

r = BigInteger.ModPow(two, a, n);

if (r != null && BigInteger.Compare(r, 1) == 0) return;

r = new BigInteger();

if (BigInteger.Compare(b2, a) < 0)

r = BigInteger.ModPow(two, b2, n);

else if (BigInteger.Compare(b, a2) < 0)

r = BigInteger.ModPow(two, b, n);

if (r != null && BigInteger.Compare(r, 1) == 0) return;

r = BigInteger.ModPow(two, fp / 2, n);

if (BigInteger.Compare(r, 1) != 0) return;

l.Add(n);

}

//------------------------------------------------------------------------------Конец генерции чисел----------------------------------------------------------------------------------------

public Form1()

{

InitializeComponent();

}

//------------------------------------------------------------------------------Шифрование и дешифрование----------------------------------------------------------------------------------------

private List<string> RSA\_Endoce(string s, BigInteger e, BigInteger n)//Функция выполняющая шифрование

{

List<string> result = new List<string>();

BigInteger bi;

int size = s.Length;

for(int i=0; i<size; i++)

{

bi = s[i];

result.Add(BigInteger.ModPow(bi, e, n).ToString());

}

return result;

}

private string RSA\_Dedoce(string[] input, BigInteger d, BigInteger n)//Функция выполняющая дешифрование

{

string result = "";

BigInteger bi;

foreach (string item in input)

{

bi = BigInteger.Parse(item);

bi = BigInteger.ModPow(bi, d, n);

try

{

result += (char)bi;

}

catch(Exception)

{

MessageBox.Show("Problem of decryption. Check your keys");

break;

}

}

return result;

}

//------------------------------------------------------------------------------Конец шифрования и дешифрования----------------------------------------------------------------------------------------

private BigInteger EuclidAlgorithm(BigInteger a, BigInteger b)//Алгоритм Евклида для нахождения НОД

{

BigInteger reminder;

BigInteger.DivRem(a, b, out reminder);

while(reminder!=BigInteger.Zero)

{

a = b;

b = reminder;

BigInteger.DivRem(a, b, out reminder);

}

return b;

}

//------------------------------------------------------------------------------Высчитывание публичного и приватного ключей-----------------------------------------------------------------------

private BigInteger Calculate\_e(BigInteger m)//Вычисление открытого ключа

{

Random rnd = new Random();

while(true)

{

BigInteger d1 = rnd.Next(1000000000);

BigInteger d2 = rnd.Next(1000000000);

BigInteger d =BigInteger.Multiply(d1, d2);

if (d > m)

continue;

else

{

if (EuclidAlgorithm(m, d) == 1)

return d;

}

}

}

private BigInteger Calculate\_d(BigInteger a, BigInteger b)//вычисление закрытого ключа

{

BigInteger tempA = a, tempB = b;

BigInteger u1 = BigInteger.One, u2 = BigInteger.Zero, v1 = BigInteger.Zero, v2 = BigInteger.One;

BigInteger temp = new BigInteger(), remainder = new BigInteger(), tempU = new BigInteger(), tempV = new BigInteger();

while(b!=1)

{

BigInteger.DivRem(a, b, out remainder);

temp = BigInteger.Divide(a, b);

tempU = BigInteger.Subtract(u1, BigInteger.Multiply(u2, temp));

tempV = BigInteger.Subtract(v1, BigInteger.Multiply(v2, temp));

u1 = u2; u2 = tempU;

v1 = v2; v2 = tempV;

a = b;

b = remainder;

}

if (v2 < 0)

v2 = BigInteger.Add(v2, tempA);

return v2;

}

//------------------------------------------------------------------------------Конец высчитывания публичного и приватного ключей-----------------------------------------------------------------------

private void EncryptionButton\_Click(object sender, EventArgs e)//Кнопка шифрования

{

textBox\_encrypted.Clear();

if ((textBox\_n\_interlocutor.Text.Length > 0) && (textBox\_e\_interlocutor.Text.Length > 0))

{

if (textBox\_original.Text.Length > 0)

{

string s = textBox\_original.Text;

BigInteger n = BigInteger.Parse(textBox\_n\_interlocutor.Text);

BigInteger e\_ = BigInteger.Parse(textBox\_e\_interlocutor.Text);

List<string> result = RSA\_Endoce(s, e\_, n);

foreach (var text in result)

{

textBox\_encrypted.AppendText(text + "\n");

}

}

else

MessageBox.Show("The text field is empty");

}

else

MessageBox.Show("The public key is missing!");

}

private void DecryptionButton\_Click(object sender, EventArgs e)//Кнопка дешифрования

{

textBox\_original.Clear();

if ((textBox\_d.Text.Length > 0) && (textBox\_n.Text.Length > 0))

{

if (textBox\_encrypted.Text.Length > 0)

{

BigInteger d = BigInteger.Parse(textBox\_d.Text);

BigInteger n = BigInteger.Parse(textBox\_n.Text);

string[] input = textBox\_encrypted.Text.Split(new char[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries);

string result = RSA\_Dedoce(input, d, n);

textBox\_original.AppendText(result);

}

else

MessageBox.Show("The text field is empty");

}

else

MessageBox.Show("The private key is missing!");

}

private void PrimesGen\_Click(object sender, EventArgs e)//Кнопка генирации простых чисел

{

BigInteger p = new BigInteger();

BigInteger q = new BigInteger();

List<int> temp = new List<int>();

for (int i = 0; i < 10000; i++)

{

if (Check(i))

temp.Add(i);

}

generatePrimes(temp, out p, out q);

textBox\_p.Text = p.ToString();

textBox\_q.Text = q.ToString();

}

private void KeysGen\_Click(object sender, EventArgs e)//Кнопка вычисления ключей

{

if ((textBox\_p.Text.Length > 0) && (textBox\_q.Text.Length > 0))

{

BigInteger p = BigInteger.Parse(textBox\_p.Text);

BigInteger q = BigInteger.Parse(textBox\_q.Text);

BigInteger n = BigInteger.Multiply(p, q);

BigInteger m = BigInteger.Multiply((p - 1), (q - 1));

BigInteger e\_ = Calculate\_e(m);

BigInteger d = Calculate\_d(m, e\_);

textBox\_d.Text = d.ToString();

textBox\_n.Text = n.ToString();

textBox\_e.Text = e\_.ToString();

}

else

MessageBox.Show("Generate Prime numbers!");

}

//Кнопки очистки полей

private void clear\_original\_textbox\_Click(object sender, EventArgs e)

{

textBox\_original.Clear();

}

private void clear\_encrypted\_textbox\_Click(object sender, EventArgs e)

{

textBox\_encrypted.Clear();

}

private void clear\_primes\_Click(object sender, EventArgs e)

{

textBox\_p.Clear();

textBox\_q.Clear();

}

private void clear\_keys\_Click(object sender, EventArgs e)

{

textBox\_n.Clear();

textBox\_e.Clear();

textBox\_d.Clear();

}

private void digital\_signature\_button\_Click(object sender, EventArgs e)

{

textBox\_signature\_encrypted.Clear();

if ((textBox\_n.Text.Length > 0) && (textBox\_d.Text.Length > 0))

{

if (textBox\_signature\_origin.Text.Length > 0)

{

string s = textBox\_signature\_origin.Text;

BigInteger n = BigInteger.Parse(textBox\_n.Text);

BigInteger d = BigInteger.Parse(textBox\_d.Text);

List<string> result = RSA\_Endoce(s, d, n);

foreach (var text in result)

{

textBox\_signature\_encrypted.AppendText(text + "\n");

}

}

else

MessageBox.Show("The text field is empty");

}

else

MessageBox.Show("The private key is missing!");

}

private void confirming\_signature\_button\_Click(object sender, EventArgs e)

{

if ((textBox\_e\_interlocutor.Text.Length > 0) && (textBox\_n\_interlocutor.Text.Length > 0))

{

if (textBox\_signature\_origin.Text.Length > 0 && textBox\_signature\_encrypted.Text.Length > 0)

{

BigInteger e\_ = BigInteger.Parse(textBox\_e\_interlocutor.Text);

BigInteger n = BigInteger.Parse(textBox\_n\_interlocutor.Text);

string[] input = textBox\_signature\_encrypted.Text.Split(new char[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries);

string result = RSA\_Dedoce(input, e\_, n);

if(result == textBox\_signature\_origin.Text)

MessageBox.Show("The signature is verified");

else

MessageBox.Show("The signature is not verified");

}

else

MessageBox.Show("The text field is empty");

}

else

MessageBox.Show("The public key is missing!");

}

}

}