Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет Компьютерных Систем и Сетей

Кафедра Информатики

Курсовой проект по дисциплине «Программирование»:

Шифрование данных методом RSA

Выполнила:

студентка группы 753501

Самойлик Е. Н.

Научный руководитель:

ассистент кафедры информатики

Летохо А. С.

Минск 2018

**Содержание**

[Глава 1. Введение. Цели и задачи. 3](#_Toc515468356)

[Глава 2. Описание алгоритмов 4](#_Toc515468357)

2.1. Алгоритм шифрования RSA 7  
2.2 Возможные атаки на RSA............................................................................10

[2.3. Улучшения ...................................................................................................12](#_Toc515468359)

[Глава 3. Техническая реализация 13](#_Toc515468360)

[3.1. Структура программы: 13](#_Toc515468361)

[3.2. Организация данных 14](#_Toc515468362)

[3.3. Основные функции 15](#_Toc515468363)

[3.4. Интеграция с командной строкой Windows 16](#_Toc515468364)

[Глава 4. Выводы, возможности улучшения 18](#_Toc515468365)

[Глава 5. Список использованных источников 20](#_Toc515468366)

[Приложение 1. Исходный код 22](#_Toc515468367)

**Глава 1. Введение. Цели и задачи.**

Во многих отраслях достаточно остро стоит проблема сохранности информации, что порождает необходимость в ее защите. В большинстве случаев локально защитить какую-либо информацию от несанкционированного вмешательства помогает шифрование – обратимое преобразование информации в целях ее сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Тема шифрования была выбрана в качестве курсовой работы по следующим причинам:

1. Тема является интересной, полезной, применимой и развивающейся в настоящее время;
2. Тема позволяет реализовывать различные типы проектов: от небольшой библиотеки до огромных клиент-серверных приложений;
3. Тема предоставляет широкие возможности доработки уже существующих алгоритмов и возможности для улучшения приложений в дальнейшем;

В соответствии с выбранной темой были поставлены следующие цели:

1. Реализовать алгоритм шифрования методом RSA;
2. Предоставить возможность работать с модулями проекта как с библиотеками с целью применения их в других программах;
3. Интегрировать приложение с командной строкой Windows, сведя количество вводимых данных к минимуму;

**Глава 2. Описание алгоритмов**

Шифрование — обратимое преобразование информации в целях сокрытия от [неавторизованных](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) лиц, с предоставлением, в это же время, [авторизованным](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения [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) передаваемой информации. Важной особенностью любого алгоритма шифрования является использование [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)), который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма.

Пользователи являются авторизованными, если они обладают определённым [аутентичным](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) [ключом](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)). Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс.

В целом, шифрование состоит из двух составляющих — зашифровывание и расшифровывание.

С помощью шифрования обеспечиваются три состояния безопасности информации:

* Конфиденциальность.

Шифрование используется для скрытия информации от неавторизованных пользователей при передаче или при хранении.

* Целостность.

Шифрование используется для предотвращения изменения информации при передаче или хранении.

* Идентифицируемость.

Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.

Для того, чтобы прочитать зашифрованную информацию, принимающей стороне необходимы [ключ](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)) и дешифратор (устройство, реализующее алгоритм расшифровывания). Идея шифрования состоит в том, что злоумышленник, перехватив зашифрованные данные и не имея к ним ключа, не может ни прочитать, ни изменить передаваемую информацию. Кроме того, в современных [криптосистемах (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) для шифрования, расшифрования данных могут использоваться разные ключи. Однако, с развитием криптоанализа, появились методики, позволяющие дешифровать закрытый текст без ключа. Они основаны на математическом анализе переданных данных.

В зависимости от структуры используемых ключей методы шифрования подразделяются на:

• симметричное шифрование: посторонним лицам может быть известен алгоритм шифрования, но неизвестна небольшая порция секретной информации — ключа, одинакового для отправителя и получателя сообщения;

• асимметричное (открытое) шифрование: посторонним лицам может быть известен алгоритм шифрования, и, возможно открытый ключ, но неизвестен закрытый ключ, известный только получателю. Криптографические системы с открытым ключом в настоящее время широко применяются в различных сетевых протоколах;

Криптографические преобразования имеют цель обеспечить недоступность информации для лиц, не имеющих ключа, и поддержание с требуемой надежностью обнаружения несанкционированных искажений. Большинство средств защиты информации базируется на использовании криптографических шифров и процедур шифрования-расшифрования. В соответствии со стандартом ГОСТ 28147-89 под шифром понимают совокупность обратимых преобразований множества открытых данных на множество зашифрованных данных, задаваемых ключом и алгоритмом преобразования.

В криптографии используются следующие основные алгоритмы шифрования:

* алгоритм замены (подстановки) – символы шифруемого текста заменяются символами того же или другого алфавита в соответствии с заранее обусловленной схемой замены;
* алгоритм перестановки – символы шифруемого текста переставляются по определенному правилу в пределах некоторого блока этого текста;
* гаммирование – символы шифруемого текста складываются с символами некоторой случайной последовательности;
* аналитическое преобразование – преобразование шифруемого текста по некоторому аналитическому правилу (формуле).

**2.1. Алгоритм шифрования RSA**

На данный момент асимметричное шифрование на основе открытого ключа RSA (расшифровывается, как Rivest, Shamir and Aldeman - создатели алгоритма) использует большинство продуктов на рынке информационной безопасности. 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).

Его криптостойкость основывается на сложности разложения на множители больших чисел, а именно - на исключительной трудности задачи определить секретный ключ на основании открытого, так как для этого потребуется решить задачу о существовании делителей целого числа. Наиболее криптостойкие системы используют 2048-битовые большие числа.

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 , то  вычислить относительно просто.
* Если известно , то для вычисления  нет простого (эффективного) пути.

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

Как было сказано выше, 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 всех допустимых пар открытого и закрытого ключей существуют соответствующие функции шифрования  и расшифрования  такие, что все сообщения , где  — множество допустимых сообщений,

*.*

Рассмотрим алгоритм RSA с практической точки зрения.

Для начала необходимо сгенерировать открытый и секретные ключи:

* Возьмем два больших простых числа p и q заданного размера.
* Определим n, как результат умножения p на q (n= p\*q).
* Выберем случайное число, которое назовем d. Это число должно быть взаимно простым со значением функции Эйлера от числа n, т.е. с результатом умножения (p-1) \* (q-1).

Определим такое число е, для которого является истинным следующее соотношение (e\*d) mod ((p-1)\*(q-1)) = 1. Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в [двоичной записи](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), например, простые [числа Ферма](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B5%D1%80%D0%BC%D0%B0) 17, 257 или 65537.

* Число e называется открытой экспонентой.
* Время, необходимое для шифрования с использованием [быстрого возведения в степень](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE_%D0%B2%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C), пропорционально числу единичных бит в e.
* Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.
* Назовём открытым ключом числа {e; n}, а секретным – {d; n}.

Для того, чтобы зашифровать данные по открытому ключу {e, n}, необходимо следующее:

* разбить шифруемый текст на блоки, каждый из которых может быть представлен в виде числа Mi = 0,1,2..., n-1( т.е. только до n-1).
* зашифровать текст, рассматриваемый как последовательность чисел Mi) по формуле Ci = Mie mod n.

Чтобы расшифровать эти данные, используя секретный ключ {d,n}, необходимо выполнить следующие вычисления: Mi = Cid mod n. В результате будет получено множество чисел Mi, которые представляют собой исходный текст.

Таким образом, отметим, что:

Основными достоинствами шифра RSA являются:

* использование двух ключей, что позволяет использовать его в клиент-серверных приложениях;
* надежность шифрования при использовании достаточно длинных ключей (512 бит и более).

Основными его недостатками являются:

* необходимость генерации длинных простых чисел, что требует гигантских вычислительных мощностей;
* падение криптостойкости при небольших по длине ключах.

**2.2. Возможные атаки на RSA**

Главным и наиболее очевидным способом сломать RSA является отыскание закрытого ключа по данному открытому. Это позволило бы злоумышленнику читать все сообщения, зашифрованные этим ключом. Сделать это можно, например, разложив n на простые множители p и q. После этого отыскание d является тривиальной задачей. Надежность RSA основывается на трудности разложения n.

Лучшим алгоритмом, применяемым для факторизации на сегодняшний день, является NFS (Number Field Sieve - числовое решето). Алгоритм этот сравнительно новый - при разложении знаменитого числа RSA-129, полученного в марте 1994, еще использовалось менее эффективное квадратичное решето, а не NFS. RSA-129 — это 129-значное число, разложение которого считалось весьма желательным с 1977 года. Для осуществления этого проекта через Internet было задействовано 1600 компьютеров (в том числе два факса), работавших в течение восьми месяцев. По некоторым оценкам использование NFS могло бы сократить этот срок в три-четыре раза. Таким образом, если 130 знаков примерно составляют предел для квадратичного решета, то использование NFS поднимает планку до 140-150 знаков. Модуль длиной 512 бит содержит около 155 знаков, поэтому для практически большинства приложений этого достаточно. По мнению фирмы RSA Data Security разложение такого числа будет стоить порядка $1,000,000 и займет около восьми месяцев. Также, по ее мнению, ключ длиной 768 бит останется неразложимым примерно до 2004 года.

Другим способом сломать RSA является отыскание способа вычисления корней степени e по модулю n. Так как, то даст нам исходное сообщение m. Этот способ не является эквивалентным разложению n, так как он не дает нам закрытого ключа d. Тем не менее само по себе вычисление корней над полями Галуа представляет собой сложную задачу. Известна теорема Рабина о том, что вычисление эквивалентно факторизации n, так что на самом деле практически не предпринималось попыток раскрытия шифра в этом направлении.

Все прочие способы не дают возможности раскрытия всех сообщений, зашифрованных данным ключом. Однако существуют подходы, иногда дающие возможность взломать данное конкретное сообщение. Простейший из них - угадывание оригинала. Злоумышленник видит шифр, догадывается, что в сообщении написано “Стреляй в Бориса”, зашифровывает свою версию известным открытым ключом и сравнивает с шифровкой. Еще один подход основан на свойствах преобразования RSA. Если одно и то же сообщение посылается e разным получателям (где е не основание натурального логарифма, а открытая экспонента), то злоумышленник, перехвативший все е шифровок, сможет восстановить исходное сообщение.

**2.3. Улучшения**

В данной работе не только представлена реализация стандартного алгоритма шифрования RSA, но и предложена его небольшая доработка с помощью рекуррентных последовательностей.

Возьмем любую рекуррентную последовательность вида:

a1 = A > 0; // псевдослучайное число

a2 = B > 0; // псевдослучайное число

ai+1 = α • ai + β • ai-1; // α и β – также псевдослучайны

На позиции j = ai добавим псевдослучайные символы или заранее заготовленную информацию, сгенерированную по какому-либо алгоритму. Таким образом, попытка узнать длину ключа путем циклического сдвига сильно затруднена. Таким образом, при надлежащем использовании, данный алгоритм может быть использован для шифрования файлов. При разработке утилиты особое внимание уделено не только реализации данного алгоритма, но и попытке распространения такой системы на асинхронный алгоритм шифрования.

**Глава 3. Техническая реализация**

**3.1. Структура программы**

Main.cpp

Encryption\_RSA\_Advanced.h

Encryption\_RSA\_Advanced.cpp

RSA.h

Primes.h

RSA.cpp

Primes.cpp

Файл, содержащий функции шифрования/ дешифрования, это **Encryption\_RSA\_Advanced.cpp.** Вместе с его заголовочным файлом, содержащим прототипы функций, данный файл может быть использован в качестве небольшой и достаточно простой библиотеки шифрования. С этой целью в проект необходимо добавить все файлы, которые включены данными файлами. В файле **Main.cpp** реализованы детали интерфейса и интеграции с командной строкой Windows.

**3.2. Организация данных**

В процессе разработки были созданы два класса: **RSA** и **Sequence**. Из стандартных классов часто использовался стандартный класс **string**.

Класс **RSA** служит для генерации и хранения ключей шифрования RSA, а также содержит функции для шифрования/дешифрования информации с помощью этих ключей. Закрытыми данными являются поля **p**, **q** и **PrivateKey**, а также две функции. Первая функция генерирует ключи, а вторая является служебной функцией, оболочками для которой являются функции **EncryptChar** и **DecryptChar**.

-------------------------------RSA.cpp-------------------------------

// prototype of the class defined in "Encryption\_RSA.cpp"

class RSA {

private:

\_\_int32 p;

\_\_int32 q;

\_\_int32 PrivateKey;

// private functions

void GenerateKeys();

\_\_int32 BinaryPower(\_\_int32 arg, \_\_int32 power, \_\_int32 mod);

public:

\_\_int32 PublicKey;

\_\_int32 Key;

// input/output

void Display();

void FromFile(const char\* FileName);

void ToFile(const char\* FileName);

// encryption/decryption

\_\_int32 EncryptChar(char arg);

char DecryptChar(\_\_int32 arg);

// for random sequence

\_\_int32 Random();

// constructors

RSA();

RSA(const char\* FileName);

RSA(\_\_int32 FirstPrimeNumber, \_\_int32 SecondPrimeNumber);

};

Класс **Sequence** служит для генерации и хранения индексов последовательности, а также содержит функции для генерации следующего элемента, чтения и записи структуры. Все данные закрыты, для получения следующего элемента используется функция **Next**.

------------------------------Sequence.h------------------------------

// prototype of the class defined in "Sequence.cpp"

class Sequence {

private:

\_\_int64 Start[2];

\_\_int64 Current[2];

\_\_int64 Alpha;

\_\_int64 Beta;

public:

\_\_int64 Next();

void Write();

void ToFile(const char\* FileName);

Sequence();

Sequence(\_\_int64 arg\_a1, \_\_int64 arg\_a2, \_\_int64 arg\_Alpha, \_\_int64 arg\_Beta);

Sequence(const char\* FileName);

};

**3.3. Основные функции**

В файле **Primes.cpp** реализованы функции работы с числами, которые позволяют сгенерировать простое число для шифрования RSA.

-------------------------------Primes.h-------------------------------

// prototypes of the functions defined in "Primes.cpp"

bool IsPrime(\_\_int32 Number);

\_\_int32 GeneratePrime(\_\_int32 MaxValue);

\_\_int32 GCD(\_\_int32 First, \_\_int32 Second);

В файле **Encryption\_RSA.cpp** реализованы стандартные функции шифрования, в **Advanced**-версии функции шифрования улучшается так, как изложено в п. 2.4.

---------------------------Encryption\_RSA.h---------------------------

// prototypes of functions defined in "Encryption\_RSA.cpp"

void RSA\_Encrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile);

void RSA\_Decrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile);

-----------------------Encryption\_RSA\_Advanced.h----------------------

// prototypes of functions defined in "Encryption\_RSA\_Advanced.cpp"

void RSA\_Advanced\_Encrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile, const char\* SequenceFile);

void RSA\_Advanced\_Decrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile, const char\* SequenceFile);

Можно видеть, что в каждом из них реализованы две функции **Encrypt/Decrypt**, принимающие следующие аргументы:

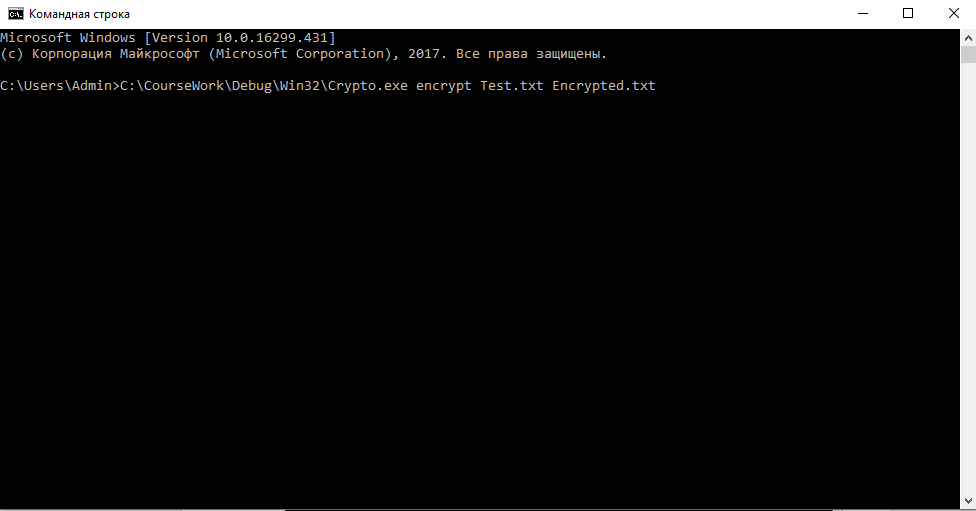
* 1. Название шифруемого файла;
  2. Название файла с выводом;
  3. Название файла, содержащего ключ;
  4. Название файла, хранящего структуру Sequence (в **Advanced**-версии);

Каждая функция шифрования выполняет следующие действия:

1. Сгенерировать ключ;
2. Сохранить его в файл;
3. Сгенерировать последовательность (для **Advanced**-версии);
4. Открыть файлы;
5. Если шифруемого файла не существует, то выйти из функции с сообщением об ошибке;
6. Зашифровать файл необходимым образом;
7. Закрыть файлы.

**3.4. Интеграция с командной строкой Windows**

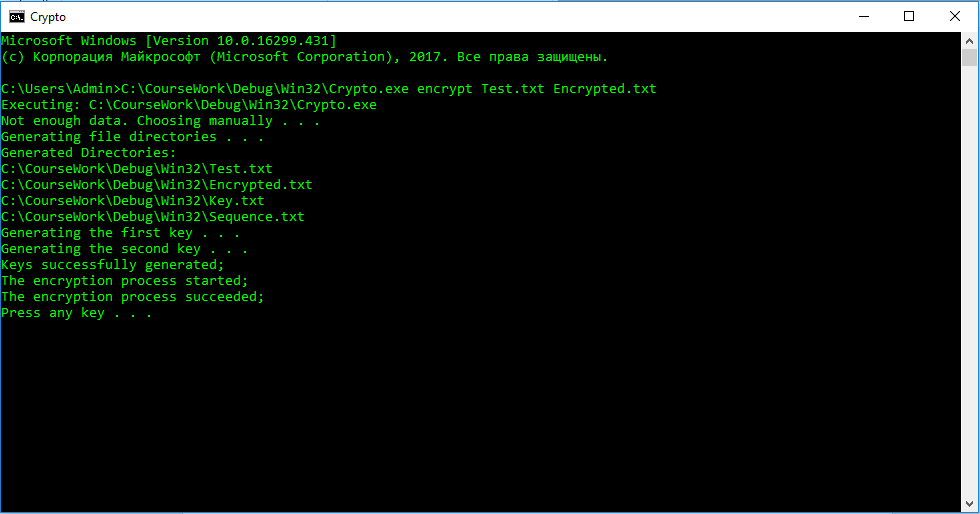
Для взаимодействия с пользователем было решено интегрировать данную программу с командной строкой Windows (**cmd.exe**). Очевидными предпосылками для этого служит то, что функции шифрования/дешифрования принимают строго определенное количество аргументов, что позволяет использовать программу «в одну строчку». Чтобы работать с программой таким образом, необходимо написать в командной строке **<Путь к программе> <encrypt|decrypt> <Название файла ввода> <Название файла вывода>** без скобок и кавычек, причем все файлы (ввод, вывод и ключи) хранятся в папке программы.

Например, если пользователю необходимо зашифровать файл **«Test.txt»** в файл **«Encrypted.txt»**, то необходимо написать в командной строке **«Crypto.exe encrypt Test.txt Encrypted.txt»**, после чего программа, найдя в папке программы файл **Test.txt**, зашифрует его шифром RSA, выведя данные в файл **Encrypted.txt**. 

*запуск приложения через командную строку*

Однако при работе программы могут возникнуть непредвиденные ситуации, связанные с корректностью введенных данных:

1. Пользователь не ввел часть данных => программа запросит недостающие;
2. Пользователь ввел неправильно **«encrypt|decrypt»** => по умолчанию используется **encrypt**;
3. Пользователь неверно указал имена файлов => при шифровании процесс оборвется с сообщением об ошибке в консоль;



*корректная работа программы*

**Глава 4. Выводы, возможности улучшения**

Данная работа включила в себя изучение стандартного алгоритма шифрования методом RSA, небольшое его улучшение, реализацию утилиты шифрования и ее адаптацию для работы через командную строку Windows. Работа была проделана полностью, и конечная реализация включает в себя рабочую утилиту **Crypto.exe**, а также файлы с кодом, которые можно использовать в других приложениях в качестве библиотек.

В процессе разработки приложения были получены знания о разнообразных системах шифрования, включая реализованную, а также получен ценный опыт работы с защитой данных. Тема является достаточно интересной, чтобы продолжить работу в том же направлении.

Как и любое приложение, данный проект можно улучшить. В качестве ближайших улучшений можно рассматривать:

1. Подключение к RSA длинной арифметики;
2. Реализация архивации (сжатия) файлов до их шифрования;
3. Подключение или реализация собственного генератора случайных чисел (т. к. стандартный генератор не считается криптостойким);
4. Добавление других, более сложных и криптостойких алгоритмов шифрования.

**Глава 5. Список использованных источников**

1. Баричев С., Криптография без секретов

2. Е.Бубнова, Системы с открытым ключом: алгоритм дешифрования RSA

3. RSA Laboratories, Frequently Asked Questions About Today's Cryptography, 1995

4. Д.Сяо, Д. Керр, С.Мэдник, Защита ЭВМ

5. Л.Дж.Хоффман, Современные методы защиты информации

6. James Nechvatal, Public key cryptography, 1991

7. Chris K.Caldwell, Finding Primes and proving primality

8. Сайт <http://www.e-nigma.ru/stat/rsa>

9. Сайт <https://ru.wikipedia.org/wiki/RSA>

**Приложение 1. Исходный код**

Main.cpp

#pragma hdrstop

#pragma argsused

#ifdef \_WIN32

#include <tchar.h>

#else

typedef char \_TCHAR;

#define \_tmain main

#endif

//---------------------------------------------------------------------------

#include <windows.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

#include <string>

#include <string.h>

#include <conio.h>

#include <limits.h>

#include <time.h>

#include <dos.h>

#include <vcl.h>

//---------------------------------------------------------------------------

#include "Encryption\_RSA.h"

#include "Encryption\_RSA\_Advanced.h"

//---------------------------------------------------------------------------

int \_tmain(int argc, \_TCHAR\* argv[])

{

system("color 0a");

system("title Crypto");

randomize();

// the address of the .exe file

std::cout << "Executing: " << argv[0] << endl;

// creating the path to the key files

std::string ProgramPath(argv[0]);

ProgramPath.erase(ProgramPath.length() - 10, 10);

// input data

std::string Mode;

std::string Input;

std::string Output;

// getting arguments

if (argc >= 2) {

Mode = argv[1];

if (argc >= 3) {

Input = argv[2];

if (argc >= 4) {

Output = argv[3];

}

}

}

// If not enough data found

if (argc < 5) {

std::cout << "Not enough data. Choosing manually . . .\n";

}

if (argc < 2) {

std::cout << "Choose mode: encrypt/decrypt: ";

std::getline(std::cin, Mode);

}

if (argc < 3) {

std::cout << "Enter the input file path: ";

std::cout << ProgramPath;

std::getline(std::cin, Input);

}

if (argc < 4) {

std::cout << "Enter the output file name: ";

std::cout << ProgramPath;

std::getline(std::cin, Output);

}

// generating file paths

std::cout << "Generating file directories . . .\n";

std::string InputPath = ProgramPath;

std::string OutputPath = ProgramPath;

std::string KeyPath = ProgramPath;

std::string SequencePath = ProgramPath;

InputPath.append(Input);

OutputPath.append(Output);

KeyPath.append("Key.txt");

SequencePath.append("Sequence.txt");

std::cout << "Generated Directories:\n" << InputPath << endl << OutputPath << endl << KeyPath << endl << SequencePath << endl;

// choosing the mode and starting the process

if (Mode == "decrypt") {

RSA\_Advanced\_Decrypt(InputPath.c\_str(), OutputPath.c\_str(), KeyPath.c\_str(), SequencePath.c\_str());

}

else {

// standard mode - encrypt

RSA\_Advanced\_Encrypt(InputPath.c\_str(), OutputPath.c\_str(), KeyPath.c\_str(), SequencePath.c\_str());

}

std::cout << "Press any key . . . ";

getch();

return 0;

}

//---------------------------------------------------------------------------

Primes.h

// prototypes of the functions defined in "Primes.cpp"

bool IsPrime(\_\_int32 Number);

\_\_int32 GeneratePrime(\_\_int32 MaxValue);

\_\_int32 GCD(\_\_int32 First, \_\_int32 Second);

Primes.cpp

//---------------------------------------------------------------------------

#include <windows.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <limits.h>

#include <time.h>

#include <dos.h>

//---------------------------------------------------------------------------

#include "Primes.h"

//---------------------------------------------------------------------------

bool IsPrime(\_\_int32 Value) {

if (Value % 2 == 0) {

return false;

}

else {

\_\_int32 Max = (\_\_int32) sqrt((double) Value);

for (\_\_int32 i = 3; i <= Max; i += 2)

if (Value % i == 0)

return false;

return true;

}

}

\_\_int32 GeneratePrime(\_\_int32 MaxValue) {

if (MaxValue % 2 == 0)

MaxValue--;

for (\_\_int32 i = MaxValue; i > 0; i -= 2) {

if (IsPrime(i))

return i;

}

return 0;

}

\_\_int32 GCD(\_\_int32 First, \_\_int32 Second) {

if (First % Second) {

First %= Second;

return GCD(Second, First);

}

else return std::min(First, Second);

}

RSA.h

// prototype of the class defined in "RSA.cpp"

class RSA {

private:

\_\_int32 p;

\_\_int32 q;

\_\_int32 PrivateKey;

// private functions

void GenerateKeys();

\_\_int32 BinaryPower(\_\_int32 arg, \_\_int32 power, \_\_int32 mod);

public:

\_\_int32 PublicKey;

\_\_int32 Key;

// input/output

void Display();

void FromFile(const char\* FileName);

void ToFile(const char\* FileName);

// encryption/decryption

\_\_int32 EncryptChar(char arg);

char DecryptChar(\_\_int32 arg);

// for random sequence

\_\_int32 Random();

// constructors

RSA();

RSA(const char\* FileName);

RSA(\_\_int32 FirstPrimeNumber, \_\_int32 SecondPrimeNumber);

};

RSA.cpp

//---------------------------------------------------------------------------

#include <windows.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <limits.h>

#include <math.h>

#include <time.h>

#include <dos.h>

//---------------------------------------------------------------------------

#include "RSA.h"

#include "Primes.h"

//---------------------------------------------------------------------------

#define HALF LONG\_MAX / 4 + LONG\_MAX / 8 + LONG\_MAX / 16 + LONG\_MAX / 32

// keys generation

void RSA::GenerateKeys() {

\_\_int32 Temp = (p - 1) \* (q - 1);

std::cout << "Generating the first key . . .\n";

\_\_int32 Start = 1000 + random(1000);

for (\_\_int32 d = Start; ; d++) {

if (GCD(Temp, d) == 1) {

PrivateKey = d;

break;

}

}

std::cout << "Generating the second key . . .\n";

for (\_\_int32 e = 2; ; e++) {

\_\_int64 ED = e \* PrivateKey;

if ((ED % Temp) == 1) {

PublicKey = e;

break;

}

}

std::cout << "Keys successfully generated;\n";

Key = p \* q;

}

// arg ^ power % mod

\_\_int32 RSA::BinaryPower(\_\_int32 arg, \_\_int32 power, \_\_int32 mod) {

if (power == 0)

return (1 % mod);

if (power % 2 == 1) {

\_\_int64 temp = BinaryPower(arg, power - 1, mod) \* (\_\_int64) (arg % mod);

return (\_\_int32) (temp % mod);

}

else {

\_\_int64 temp = BinaryPower(arg, power / 2, mod);

temp \*= temp;

return (\_\_int32) (temp % mod);

}

}

// output

void RSA::Display() {

std::cout << "p: " << p << endl;

std::cout << "q: " << q << endl;

std::cout << "PrivateKey: " << PrivateKey << endl;

std::cout << "Public Key: " << PublicKey << endl;

}

// getting from the file

void RSA::FromFile(const char\* FileName) {

FILE\* InputFile = fopen(FileName, "rb");

fread(this, sizeof(RSA), 1, InputFile);

fclose(InputFile);

}

// saving to the file

void RSA::ToFile(const char\* FileName) {

FILE\* OutputFile = fopen(FileName, "wb");

fwrite(this, sizeof(RSA), 1, OutputFile);

fclose(OutputFile);

}

// encryption

\_\_int32 RSA::EncryptChar(char arg) {

return BinaryPower(arg, PublicKey, Key);

}

// decryption

char RSA::DecryptChar(\_\_int32 arg) {

return (char) BinaryPower(arg, PrivateKey, Key);

}

// for random sequence

\_\_int32 RSA::Random() {

return random(p \* q);

}

// constructor - standard

RSA::RSA() {

\_\_int32 Max = (\_\_int32) sqrt(sqrt((double) LONG\_MAX));

p = GeneratePrime(Max / 2 + random(Max / 2));

q = GeneratePrime(Max / 2 + random(Max / 2));

GenerateKeys();

}

// constructor - from file

RSA::RSA(const char\* FileName) {

FromFile(FileName);

}

// constructor - user values

RSA::RSA(\_\_int32 FirstPrimeNumber, \_\_int32 SecondPrimeNumber) {

if (IsPrime(FirstPrimeNumber) && IsPrime(SecondPrimeNumber)) {

\_\_int64 N = FirstPrimeNumber \* SecondPrimeNumber;

if (N < LONG\_MAX) {

p = FirstPrimeNumber;

q = SecondPrimeNumber;

}

else {

RSA();

}

}

else {

RSA();

}

GenerateKeys();

}

Sequence.h

// prototype of the class defined in "Sequence.cpp"

class Sequence {

private:

\_\_int64 Start[2];

\_\_int64 Current[2];

\_\_int64 Alpha;

\_\_int64 Beta;

public:

\_\_int64 Next();

void Write();

void ToFile(const char\* FileName);

Sequence();

Sequence(\_\_int64 arg\_a1, \_\_int64 arg\_a2, \_\_int64 arg\_Alpha, \_\_int64 arg\_Beta);

Sequence(const char\* FileName);

};

Sequence.cpp

//---------------------------------------------------------------------------

#include <windows.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <limits.h>

#include <time.h>

#include <dos.h>

//---------------------------------------------------------------------------

#include "Sequence.h"

//---------------------------------------------------------------------------

#define ERROR 1000000000000000000

//---------------------------------------------------------------------------

// next number

\_\_int64 Sequence::Next() {

if (Current[0] > ERROR) {

return 0;

}

else {

\_\_int64 Result = Alpha \* Current[0] + Beta \* Current[1];

Current[0] = Current[1];

Current[1] = Result;

return Result;

}

}

// output

void Sequence::Write() {

std::cout << Start[0] << " " << Start[1] << " " << Alpha << " " << Beta << endl;

}

// saving the key to file

void Sequence::ToFile(const char\* FileName) {

FILE\* FileWithKey = fopen(FileName, "wb");

fprintf(FileWithKey, "%lld %lld %lld %lld", Start[0], Start[1], Alpha, Beta);

fclose(FileWithKey);

}

// random generation

Sequence::Sequence() {

randomize();

Current[0] = Start[0] = random(3) + 1;

Current[1] = Start[1] = random(3) + 1;

Alpha = random(3) + 1;

Beta = random(3) + 1;

}

// manual numbers

Sequence::Sequence(\_\_int64 arg\_a1, \_\_int64 arg\_a2, \_\_int64 arg\_Alpha, \_\_int64 arg\_Beta) {

Current[0] = Start[0] = arg\_a1;

Current[1] = Start[1] = arg\_a2;

Alpha = arg\_Alpha;

Beta = arg\_Beta;

}

// from file

Sequence::Sequence(const char\* FileName) {

FILE\* FileWithKey = fopen(FileName, "rb");

fscanf(FileWithKey, "%lld %lld %lld %lld", &Start[0], &Start[1], &Alpha, &Beta);

Current[0] = Start[0];

Current[1] = Start[1];

fclose(FileWithKey);

}

Encryption\_RSA.h

// prototypes of functions defined in "Encryption\_RSA.cpp"

void RSA\_Encrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile);

void RSA\_Decrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile);

Encryption\_RSA.cpp

//---------------------------------------------------------------------------

#include <windows.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <limits.h>

#include <time.h>

#include <dos.h>

//---------------------------------------------------------------------------

#include "Encryption\_RSA.h"

#include "RSA.h"

//---------------------------------------------------------------------------

// standard RSA encryption

void RSA\_Encrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile) {

// creating RSA structure

RSA\* Key = new RSA();

// saving it to the file

Key -> ToFile(KeyFile);

// console comments

std::cout << "The encryption process started;\n";

// opening files

FILE\* FileToEncrypt = fopen(InputFile, "rb");

FILE\* FileEncrypted = fopen(OutputFile, "wb");

// catching NULL

if (FileToEncrypt == NULL) {

std::cout << "NULL reference. The process terminated incorrectly;\n";

return;

}

// encrypting

char CurrentSymbol = '\0';

\_\_int32 CurrentEncrypted = 0;

while (true) {

CurrentSymbol = getc(FileToEncrypt);

if (!feof(FileToEncrypt)) {

CurrentEncrypted = Key -> EncryptChar(CurrentSymbol);

fprintf(FileEncrypted,"%d\n", CurrentEncrypted);

}

else {

// success report

std::cout << "The encryption process succeeded;\n";

break;

}

}

// awaiting

Sleep(200);

// closing the files

fclose(FileToEncrypt);

fclose(FileEncrypted);

}

// standard RSA decryption

void RSA\_Decrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile) {

// creating RSA structure

RSA\* Key = new RSA(KeyFile);

// console comments

std::cout << "The decryption process started;\n";

// opening files

FILE\* FileToDecrypt = fopen(InputFile, "rb");

FILE\* FileDecrypted = fopen(OutputFile, "wb");

// catching NULL

if (FileToDecrypt == NULL) {

std::cout << "NULL reference. The process terminated incorrectly;\n";

return;

}

// decryption

\_\_int32 CurrentNumber = 0;

char CurrentDecrypted = '\0';

while (true) {

fscanf(FileToDecrypt, "%d", &CurrentNumber);

if (!feof(FileToDecrypt)) {

CurrentDecrypted = Key -> DecryptChar(CurrentNumber);

putc(CurrentDecrypted, FileDecrypted);

}

else {

// success report

std::cout << "The decryption process succeeded;\n";

break;

}

}

// awaiting

Sleep(200);

// closing the files

fclose(FileToDecrypt);

fclose(FileDecrypted);

}

Encryption\_RSA\_Advanced.h

// prototypes of functions defined in "Encryption\_RSA\_Advanced.cpp"

void RSA\_Advanced\_Encrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile, const char\* SequenceFile);

void RSA\_Advanced\_Decrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile, const char\* SequenceFile);

Encryption\_RSA\_Advanced.cpp

//---------------------------------------------------------------------------

#include <windows.h>

#include <iostream.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <limits.h>

#include <time.h>

#include <dos.h>

//---------------------------------------------------------------------------

#include "Encryption\_RSA\_Advanced.h"

#include "RSA.h"

#include "Sequence.h"

//---------------------------------------------------------------------------

// the function of advanced RSA encryption using Key and Sequence

void RSA\_Advanced\_Encrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile, const char\* SequenceFile) {

// creating RSA structure

RSA\* Key = new RSA();

// saving it to the file

Key -> ToFile(KeyFile);

// creating Sequence

Sequence\* Object = new Sequence();

// saving it to the file

Object -> ToFile(SequenceFile);

// console comments

std::cout << "The encryption process started;\n";

// opening files

FILE\* FileToEncrypt = fopen(InputFile, "rb");

FILE\* FileEncrypted = fopen(OutputFile, "wb");

// catching NULL

if (FileToEncrypt == NULL) {

std::cout << "NULL reference. The process terminated incorrectly;\n";

return;

}

// encrypting

char CurrentSymbol = '\0';

\_\_int32 CurrentEncrypted = 0;

\_\_int64 i = 0, missing = Object -> Next();

while (true) {

// adding random number if necessary

if (i == missing) {

fprintf(FileEncrypted, "%d\n", Key -> Random());

missing = Object -> Next();

}

else {

CurrentSymbol = getc(FileToEncrypt);

if (!feof(FileToEncrypt)) {

CurrentEncrypted = Key -> EncryptChar(CurrentSymbol);

fprintf(FileEncrypted,"%d\n", CurrentEncrypted);

i++;

}

else {

// success report

std::cout << "The encryption process succeeded;\n";

break;

}

}

}

// awaiting

Sleep(200);

// closing the files

fclose(FileToEncrypt);

fclose(FileEncrypted);

}

// the decrypting function

void RSA\_Advanced\_Decrypt(const char\* InputFile, const char\* OutputFile, const char\* KeyFile, const char\* SequenceFile) {

// creating RSA structure

RSA\* Key = new RSA(KeyFile);

// creating Sequence

Sequence\* Object = new Sequence(SequenceFile);

// console comments

std::cout << "The decryption process started;\n";

// opening files

FILE\* FileToDecrypt = fopen(InputFile, "rb");

FILE\* FileDecrypted = fopen(OutputFile, "wb");

// catching NULL

if (FileToDecrypt == NULL) {

std::cout << "NULL reference. The process terminated incorrectly;\n";

return;

}

// decryption

\_\_int32 CurrentNumber = 0;

\_\_int64 i = 0, missing = Object -> Next();

char CurrentDecrypted = '\0';

while (true) {

if (i == missing) {

fscanf(FileToDecrypt, "%d", &CurrentNumber);

missing = Object -> Next();

}

else {

fscanf(FileToDecrypt, "%d", &CurrentNumber);

if (!feof(FileToDecrypt)) {

CurrentDecrypted = Key -> DecryptChar(CurrentNumber);

putc(CurrentDecrypted, FileDecrypted);

i++;

}

else {

// success report

std::cout << "The decryption process succeeded;\n";

break;

}

}

}

// awaiting

Sleep(200);

// closing the files

fclose(FileToDecrypt);

fclose(FileDecrypted);

}