Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет Компьютерных Систем и Сетей

Кафедра Информатики

Курсовой проект

по дисциплине: *«Программирование»*

на тему: ***«Разработка утилиты шифрования файлов для***

***Windows»***

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. 653501  Специальности: ИиТП  Молчанов И. В. |  | Проверил: Козуб В. Н.  Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |

2017

**Содержание**

1. Введение. Цели и задачи.
2. Описание алгоритмов:
   1. Алгоритм шифрования XOR;
   2. Алгоритм шифрования RSA;
   3. Сравнение выбранных алгоритмов;
   4. Улучшения;
3. Техническая реализация:
   1. Структура программы;
   2. Организация данных;
   3. Основные функции;
   4. Интеграция с командной строкой Windows;
4. Выводы, возможности улучшения;
5. **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 два алгоритма шифрования: XOR и 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;
4. **Описание алгоритмов**

В зависимости от структуры используемых ключей методы шифрования подразделяются на:

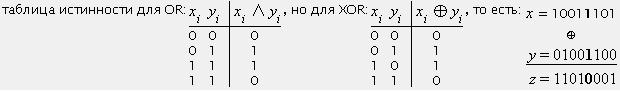
• симметричное шифрование: посторонним лицам может быть известен алгоритм шифрования, но неизвестна небольшая порция секретной информации — ключа, одинакового для отправителя и получателя сообщения;

• асимметричное 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;

**2.1. Алгоритм шифрования XOR**

Одним из самых простых и одним из самых эффективных при правильном использовании алгоритмов шифрования является шифрование с помощью логической операции побитового ИЛИ. Рассмотрим идею этого простейшего метода.

Как известно из булевой алгебры, операция «||» логического исключаещего ИЛИ имеет следующую семантику:



То есть, операция z = x || y является побитовой (поразрядной). Если ровно один из соответствующих битов равен 1, то результат равен 1. Если оба равны 0 или 1, то результат равен 0. В соответствии с приведенной таблицей истинности, операция побитового исключающего ИЛИ является обратимой (x || k = y, y || k = x);

Отсюда можно сделать следующие выводы: зная число k и применяя XOR к x и k, мы получим y. Затем, мы, опять же используя k, получим из y обратно число x. Таким образом, мы можем преобразовать последовательность чисел xi в последовательность yi, используя последовательность чисел ki. Рассматривая любую информацию как последовательность чисел, можно зашифровать любой набор данных. Таким образом можно назвать последовательность ki шифрующим ключом.

**2.2. 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.

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.

Рассмотрим алгоритм RSA с практической точки зрения.

Для начала необходимо сгенерировать открытый и секретные ключи:

* Возьмем два больших простых числа p and q.
* Определим n, как результат умножения p on q (n= p\*q).
* Выберем случайное число, которое назовем d. Это число должно быть взаимно простым 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.
* Hазовем 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.

**2.3. Сравнение выбранных алгоритмов**

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.

Достоинствами шифрования с помощью XOR являются:

• быстрота шифрования данных;

• простота реализации;

• высокая устойчивость шифра при регулярной смене ключа.

Основными его недостатками являются:

• при использовании чистого XOR c фиксированным ключом и достаточном количестве перехваченных сообщений ключ может быть получен путем анализа зашифрованных данных;

• при известной части текста ключ также может быть получен при перехвате сообщения;

• один и тот же ключ используется как для шифрования, так и для дешифровки сообщения.

**2.4. Улучшения**

В данной работе не только представлена реализация наиболее стандартных алгоритмов шифрования XOR и RSA, но и предложена их небольшая доработка с помощью реккурентных последовательностей.

Возьмем любую реккурентную 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.

**3. Техническая реализация**

* 1. **Структура программы:**

Sequence.cpp

Keys.cpp

RSA.cpp

Primes.cpp

Keys.h

Encryption\_RSA\_Advanced.cpp

RSA.h

Primes.h

Encryption\_XOR\_Advanced.cpp

Main.cpp

Encryption\_RSA\_Advanced.h

Encryption\_XOR\_Advanced.h

Sequence.h

Файлы, содержащие функции шифрования/дешифрования, это **Encryption\_XOR\_Advanced.cpp** и **Encryption\_RSA\_Advanced.cpp.** Вместе с их заголовочными файлами, содержащими прототипы функций, данные файлы могут быть использованы в качестве небольших и достаточно простых библиотек 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.

* 1. **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);

};

* 1. **Основные функции:**

В файле **Keys.cpp** реализованы функции работы с ключами шифрования XOR. Функция **CountSymbols** открывает файл, который необходимо зашифровать, и считывает количество символов в нем, чтобы сгенерировать ключ достаточной длины. Функция **GenerateKey** случайным образом генерирует ключ шифрования. Последние две функции написаны для работы с файлами, в них реализовано сохранение ключа в файл и получение его из файла.

--------------------------------Keys.h--------------------------------

long long CountSymbols(const char\* FileName);

string GenerateKey(long long NumberOfSymbols);

void KeyToFile(string Key, const char\* KeyFile);

string KeyFromFile(const char\* KeyFile);

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 файлах **Encryption\_XOR.cpp**, **Encryption\_RSA.cpp**, **Encryption\_XOR\_Advanced.cpp**, **Encryption\_RSA\_Advanced.cpp** реализованы функции шифрования/дешифрования определенным методом. В файлах **Encryption\_XOR.cpp**, **Encryption\_RSA.cpp** реализованы стандартные функции шифрования, в **Advanced**-версиях функций шифрование улучшается так, как изложено в п. 2.4.

---------------------------Encryption\_XOR.h---------------------------

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

void XOR\_Perform(const char\* InputFile, const char\* OutputFile, string Key);

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

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

---------------------------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\_XOR\_Advanced.h----------------------

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

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

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

-----------------------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**-версиях);

Каждая функция шифрования выполняет следующие действия:

1. Сгенерировать ключ;
2. Сохранить его в файл;
3. Сгенерировать последовательность (для **Advanced**-версий);
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.

В качестве примера рассмотрим функцию шифрования с помощью улучшенного XOR (**XOR\_Advanced\_Encrypt**):

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

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

// generating Key

string Key = GenerateKey(CountSymbols(InputFile));

KeyToFile(Key, KeyFile);

// generating Sequence

Sequence\* Object = new Sequence();

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;

}

// initializing additional variables

char CurrentSymbol = '\0'; // Current symbol before XOR

char CurrentEncrypted = '\0'; // Current symbol after XOR

\_\_int64 i = 0; // Number of key symbol under XOR

\_\_int64 missing = Object -> Next(); // Number of next symbol to be missed

\_\_int64 end = Key.length() - 1; // Number of last Key symbol

// encrypting using our Key and Sequence

while (true) {

// if next symbol is to be missed

if (i == missing) {

putc(random(256), FileEncrypted);

missing = Object -> Next();

}

else {

CurrentSymbol = getc(FileToEncrypt);

if (!feof(FileToEncrypt)) {

CurrentEncrypted = CurrentSymbol ^ Key[i % end];

putc(CurrentEncrypted, FileEncrypted);

i++;

}

else {

// success report

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

break;

}

}

}

// awaiting

Sleep(200);

// closing the files

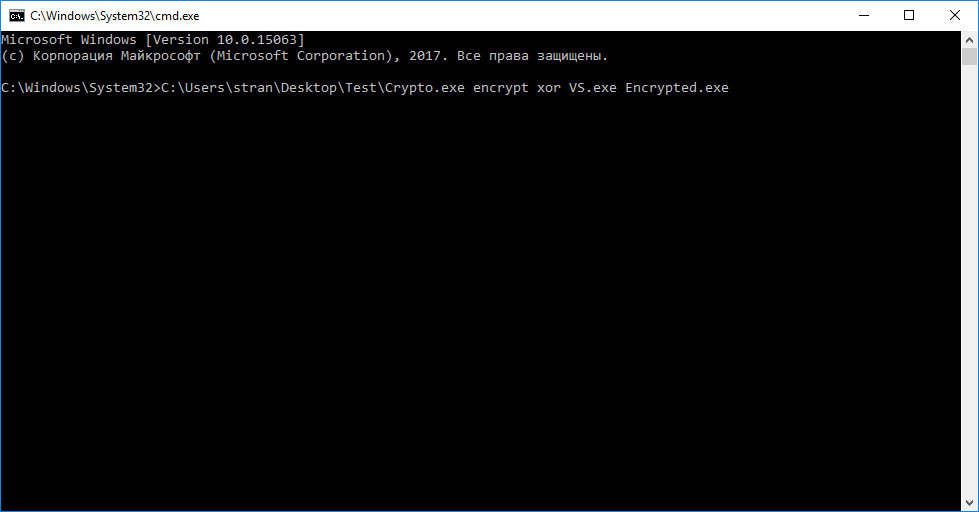
fclose(FileToEncrypt);

fclose(FileEncrypted);

* 1. **Интеграция с командной строкой Windows**

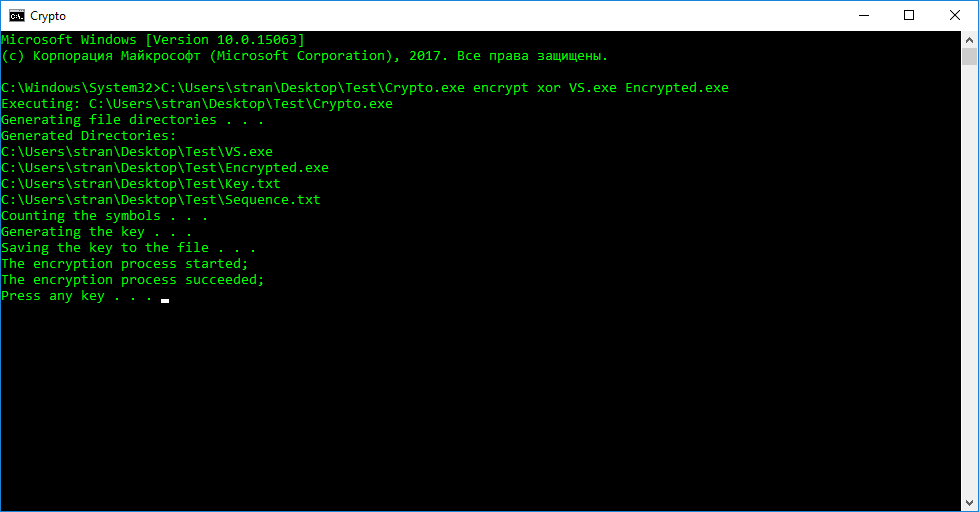
Для взаимодействия с пользователем было решено интегрировать данную программу с командной строкой Windows (**cmd.exe**). Очевидными предпосылками для этого служит то, что функции шифрования/дешифрования принимают строго определенное количество аргументов, что позволяет использовать программу «в одну строчку». Чтобы работать с программой таким образом, необходимо написать в командной строке **<Путь к программе> <encrypt|decrypt> <xor|rsa> <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»** шифрованием RSA, то необходимо написать в командной строке **«Crypto.exe encrypt rsa 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 напечатал неправильно **«xor|rsa»** => по умолчанию используется **xor**;
4. Пользователь неверно указал имена файлов => при шифровании процесс оборвется с сообщением об ошибке в консоль;

*корректная работа программы*

**4. Выводы, возможности улучшения**

Данная работа включила в себя изучение стандартных алгоритмов шифрования, небольшое их 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 реализованные, 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.