Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин (СПОВМ)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«Защищённый диск»

Студент: гр.350501 Милько А. В.

Руководитель: Яночкин А.Л.

Минск 2015

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: нет.

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2015 г.

ЗАДАНИЕ

по курсовому проекту студента

Милько Андрея Владимировича

1. Тема проекта: «Защищённый диск».
2. Срок сдачи студентом законченного проекта: 19 мая 2015 г.
3. Исходные данные к проекту:
   1. Операционная система: Windows.
   2. Язык программирования: C/C++.
   3. Алгоритм шифрования RSA.
   4. Библиотека Qt для графического интерфейса.
4. Содержание пояснительной записки:

Введение.

Лист задания

1. Обзор литературы.

2. Системное проектирование.

3. Функциональное проектирование.

4. Руководство пользователя.

5. Программа и методика испытаний.

Заключение.

Список литературы.

1. Консультант по проекту: А. Л. Яночкин
2. Дата выдачи задания: 24 февраля 2015 г.
3. Календарный график работы над проектом на весь период проектирования:
4. *разделы 1,2 к 24.03 –*
5. *раздел 3,4 к 22.04*
6. *оформление пояснительной записки к 19.05*

*Защита курсового проекта 19.05 ––––––––––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ*– –––––\_\_\_\_\_– –––\_– А. Л. Яночкин*

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_–– А. В. Милько*

(дата и подпись студента)

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Разработать систему для хранения информации в зашифрованном виде с использованием алгоритма шифрования с открытым ключом RSA. Данная программа предполагает создание и использование защищенного хранилища.

Требования к программе или программному изделию:

* язык программирования – C++;
* работа на операционной системе семейства Windows;
* для шифрования данных использовать защищенное хранилище файлов;
* алгоритм шифрования RSA;
* графический пользовательский интерфейс для создания и доступа к защищенному хранилищу.

Содержание

[ВВЕДЕНИЕ 6](#_Toc419821379)

[1. ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc419821380)

[1.1. Шифрование/дешифрование 7](#_Toc419821381)

[1.1.1. RSA 7](#_Toc419821382)

[1.1.2 Имитовставка 8](#_Toc419821383)

[2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 9](#_Toc419821384)

[3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 10](#_Toc419821385)

[3.1. Генерирование ключей 10](#_Toc419821386)

[3.2 Шифрование 10](#_Toc419821387)

[3.3 Дешифрование 12](#_Toc419821388)

[4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 14](#_Toc419821389)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc419821390)

[ЗАКЛЮЧЕНИЕ 19](#_Toc419821391)

[ПРИЛОЖЕНИЕ 20](#_Toc419821392)

# ВВЕДЕНИЕ

Прогресс подарил человечеству великое множество достижений, но тот же прогресс породил и массу проблем. Человеческий разум, разрешая одни проблемы, непременно сталкивается при этом с другими, новыми, и этот процесс обречен на бесконечность в своей последовательности. Вечная проблема - защита информации. На различных этапах своего развития человечество решало эту проблему с присущей для данной эпохи характерностью. Изобретение компьютера и дальнейшее бурное развитие информационных технологий во второй половине 20 века сделали проблему защиты информации настолько актуальной и острой, насколько актуальна сегодня информатизация для всего общества.

Сегодня, наверное, никто не сможет с уверенностью назвать точную цифру суммарных потерь от компьютерных преступлений, связанных с несанкционированных доступом к информации. Это объясняется, прежде всего, нежеланием пострадавших компаний обнародовать информацию о своих потерях, а также тем, что не всегда потери от хищения информации можно точно оценить в денежном эквиваленте. Однако по данным, опубликованным в сети, общие потери от несанкционированного доступа к информации в компьютерных системах в 1997 году оценивались в 20 миллионов долларов, а уже в 1998 года в 53,6 миллионов долларов.

В связи с этим, было принято решение, разработать приложение, позволяющее защитить информацию на диске.

# 1. ОБЗОР ЛИТЕРАТУРЫ

## 1.1. Шифрование/дешифрование

### 1.1.1. RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Криптографические системы с открытым ключом используют так называемые односторонние функции, которые обладают следующим свойством:

* если известно x, то f(x) вычислить относительно просто;
* если известно y=f(x), то для вычисления 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 произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования за разумное время (обратной операции) необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложения числа на простые множители.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом (англ. public key), так и закрытым ключом (англ. private key). 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.

Алгоритм шифрования сеансового ключа

Наиболее используемым в настоящее время является смешанный алгоритм шифрования, в котором сначала шифруется сеансовый ключ, а потом уже с его помощью участники шифруют свои сообщения симметричными системами. После завершения сеанса сеансовый ключ как правило уничтожается.[1]

### 1.1.2 Имитовставка

Имитовставка (MAC, англ. message authentication code — код аутентификации сообщения) — средство обеспечения имитозащиты в протоколах аутентификации сообщений с доверяющими друг другу участниками — специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных. MAC обычно применяется для обеспечения целостности и защиты от фальсификации передаваемой информации.

Для проверки целостности (но не аутентичности) сообщения на отправляющей стороне к сообщению добавляется значение хеш-функции от этого сообщения, на приемной стороне также вырабатывается хеш от полученного сообщения. Выработанный на приёмной стороне и полученный хеш сравниваются, если они равны то считается, что полученное сообщение дошло без изменений.

Для защиты от фальсификации (имитации) сообщения применяется имитовставка, выработанная с использованием секретного элемента (ключа), известного только отправителю и получателю. [2]

# 2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Графически принцип работы приложения SecureDisk изображён на рисунках 2.1 и 2.2.

Зашифрованные данные

Закрытый ключ

Открытый ключ

Данные

Рис. 2.1

Рис. 2.2

SecureDisk

Расшифрованные

Зашифрованные данные

SecureDisk

# 3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Программа фактически состоит из 2-х окон:

* класс “SecureDisk”;
* класс “GenerateKey”.

Класс “SecureDisk” отвечает за шифрование и дешифрование данных, а класс “GenerateKey” за генерирование пары ключей.

## 3.1. Генерирование ключей

При нажатии кнопки “Генерировать ключи”, вызывается слот класса:

void GenerateKey::accept()

{

AutoSeededRandomPool rnd;

QString path = ui.keyPathLineEdit->text();

RSAES\_OAEP\_SHA\_Decryptor rsaPrivate;

rsaPrivate.AccessKey().GenerateRandomWithKeySize(rnd, 1024);

RSAES\_OAEP\_SHA\_Encryptor rsaPublic(rsaPrivate);

SavePrivateKey(path + QString("\\") + QString(PRIVATE\_KEY), rsaPrivate.AccessKey());

SavePublicKey(path + QString("\\") + QString(PUBLIC\_KEY), rsaPublic.AccessKey());

close();

}

Этот метод получает путь к папке, в которой будут сгенерированы случайные ключи. Сначала случайно генерируется закрытый ключ методом из библиотеки cryptopp:

void GenerateRandomWithKeySize(RandomNumberGenerator &rng, unsigned int keySize);

После этого создаётся публичный, принимая закрытый ключ как параметр. Методы SavePrivateKey и SavePublicKey, являющиеся функциями класса “GenerateKey” сохраняют ключи в указанной папке. Полный код см. в приложении.

## 3.2 Шифрование

Шифрование хранилища обеспечивают классы SecureDisk и cryptohandler. При нажатии на кнопку “Зашифровать” в окне “SecureDisk” во вкладке “Tab1” вызывается слот:

void SecureDisk::encryptFolder()

{

QString folderPath = ui.folderPathLineEdit->text();

QString storagePath = ui.storagePathLineEdit->text();

QString publickKeyPath = ui.publicKeyPathLineEdit->text();

QString tempFile = folderPath + "\\" + "temp.dat";

QDir directory(folderPath);

QStringList files = directory.entryList(QDir::NoDotAndDotDot | QDir::System | QDir::Hidden | QDir::AllDirs | QDir::Files, QDir::DirsFirst);

Storage storage;

storage.open(tempFile);

for each (QString file in files)

{

QFile\* fileHandler = new QFile(folderPath + "\\" + file);

fileHandler->open(QIODevice::ReadOnly);

storage.put(fileHandler, file);

fileHandler->close();

delete fileHandler;

}

storage.close();

CryptoHandler::EncryptFile(publickKeyPath, tempFile, storagePath);

QFile::remove(tempFile);

}

Этот метод получает следующие пути:

* к папке, которую нужно зашифровать;
* к папке, где будет находиться зашифрованное хранилище;
* к открытому ключу.

Результатом выполнения функции является зашифрованное

хранилище.

В данном методе используется временное хранилище, объект класса

“storage” из библиотеки фреймворка QT. В него помещаются все файлы из выбранной папки. Далее вызывается метод класса “ CryptoHandler” “EncryptFile” который шифрует хранилище, и записывает выбранную папку для зашифрованного хранилища. Временное хранилище удаляется.

void CryptoHandler::EncryptFile(QString publickeyfile,

QString filetoencrypt,

QString encryptedfile)

{

// Generate passphrase

string passphrase = randomStrGen(50);

string encrypted\_passphrase = RSAEncryptString(publickeyfile.toStdString(),

passphrase);

// Encrypt the file and save encrypted data to the string

string data;

FileSource f(filetoencrypt.toStdString().c\_str(), true,

new DefaultEncryptorWithMAC(

passphrase.c\_str(),

new HexEncoder(new StringSink(data))));

// Append encrypted passphrase to the beginning of the string

data = encrypted\_passphrase + data;

// Save string to the file

StringSource ss(data.c\_str(), true,

new FileSink(encryptedfile.toStdString().c\_str()));

}

На самом деле для шифрования используется гибридный алгоритм.

С помощью алгоритма RSA шифруется строка (первые 50 символов). А эта строка уже используется для шифрования файлов с помощью алгоритма MAC.

## 3.3 Дешифрование

Дешифрование хранилища также обеспечивают классы SecureDisk и cryptohandler. При нажатии на кнопку “Расшифровать” в окне “SecureDisk” во вкладке “Tab2” вызывается слот:

void SecureDisk::decryptStorage()

{

QString folderPath = ui.folderPathLineEdit\_2->text();

QString storagePath = ui.storagePathLineEdit\_2->text();

QString privateKeyPath = ui.privateKeyPathLineEdit->text();

QString tempFile = folderPath + "\\temp.dat";

CryptoHandler::DecryptFile(privateKeyPath, storagePath, tempFile);

Storage storage;

storage.open(tempFile);

QStringList files = storage.getNames();

for each (QString file in files)

{

QFile\* fileHandler = new QFile(folderPath + "\\" + file);

fileHandler->open(QIODevice::ReadOnly);

storage.out(fileHandler, file);

fileHandler->close();

delete fileHandler;

}

storage.close();

QFile::remove(tempFile);

}

Этот метод получает следующие пути:

* к хранилищу, которое нужно расшифровать;
* к папке, где будет находиться расшифрованная информация;
* к закрытому ключу.

Результатом выполнения этой функции является “распаковка”

зашифрованного хранилища в указанную папку. В данном методе вызывается функция класса “CryptoHandler” “ DecryptFile”.

void CryptoHandler::DecryptFile(QString privatekeyfile,

QString filetodecrypt,

QString decryptedfile)

{

// Read string from file

std::string data;

FileSource ff(filetodecrypt.toStdString().c\_str(), true, new StringSink(data));

// Grab passphrase from the string

string encrypted\_passphrase = data.substr(0, 256);

encrypted\_passphrase.resize(256);

string passphrase = RSADecryptString(privatekeyfile.toStdString(),

encrypted\_passphrase);

// Grab cipher from the string

std::string cipher = data.substr(256, data.length()).c\_str();

// Decrypt string and save to file

StringSource s(cipher.c\_str(), true,

new HexDecoder(new DefaultDecryptorWithMAC(

passphrase.c\_str(),

new FileSink(decryptedfile.toStdString().c\_str()))));

}

Сначала функция ‘RSADecryptString” алгоритмом RSA расшифровывает строку, которая была записана функцией “RSAEncryptString” при шифровании. А далее хранилище расшифровывается алгоритмом MAC.

# 4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для шифрования и дешифрования нужны соответственно открытый и закрытый ключи. Чтобы создать пару ключей необходимо выбрать вкладку “Tab1” и кликнуть на кнопку “Cгенерировать ключи” (см. рис. 4.1).

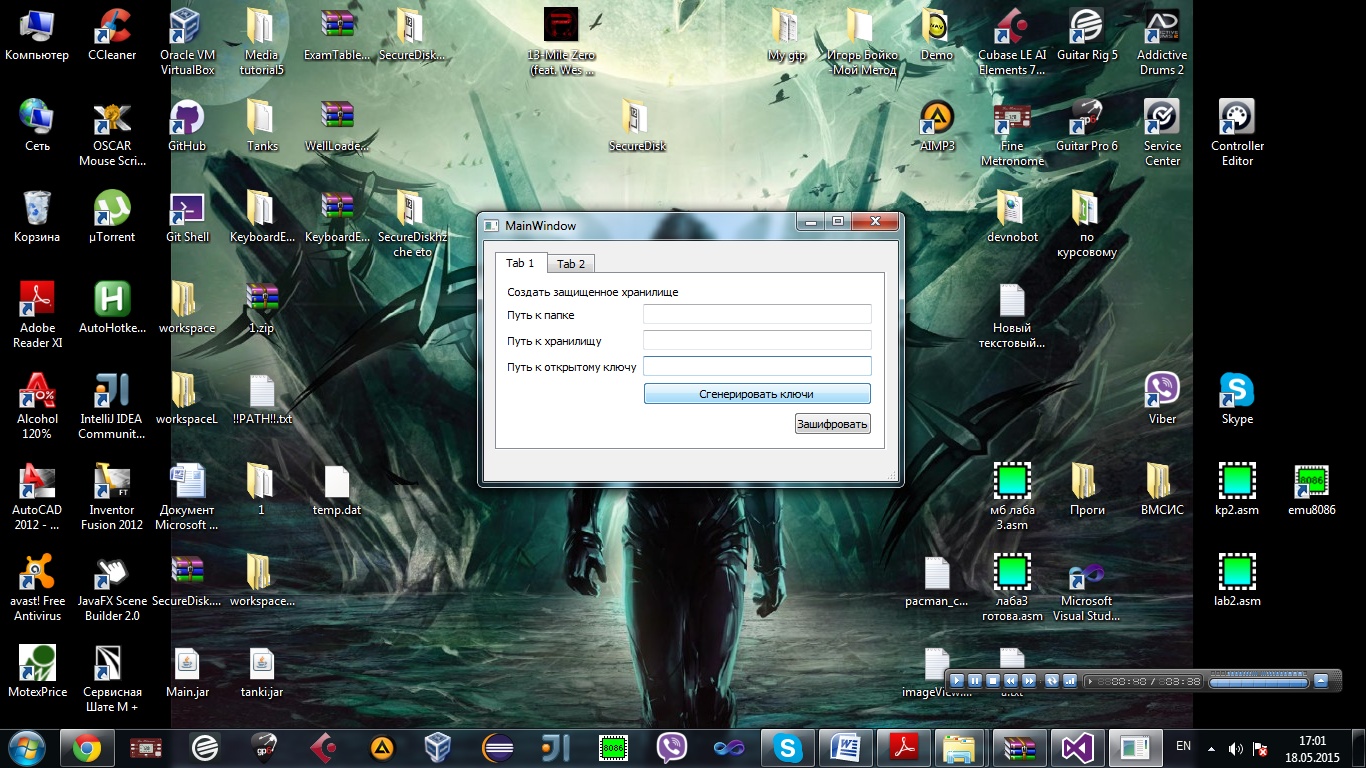


Рис. 4.1 Общий вид интерфейса

В появившемся окне в поле нужно вписать папку, в которой будут сгенерирована пара ключей (рис. 4.2). Результат показан на рис. 4.3.

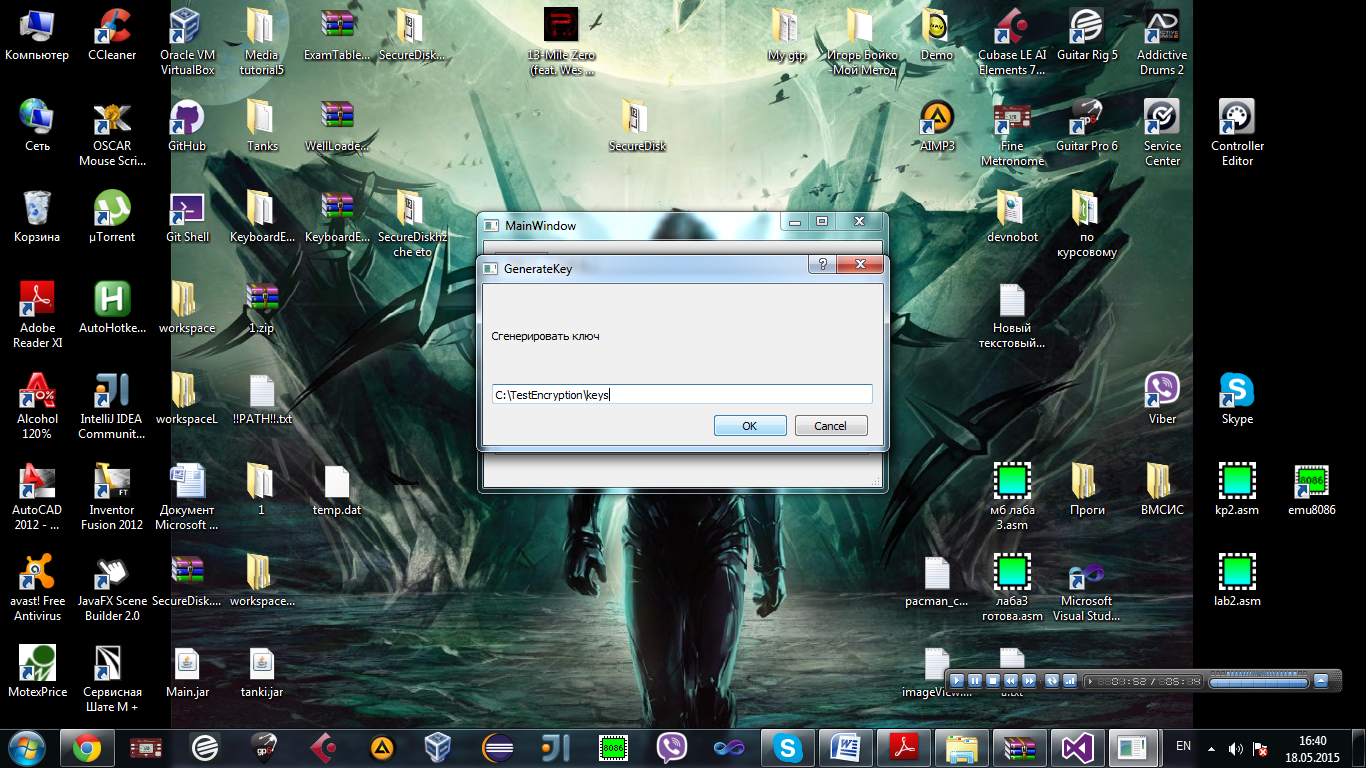


Рис. 4.2 Генерация ключей

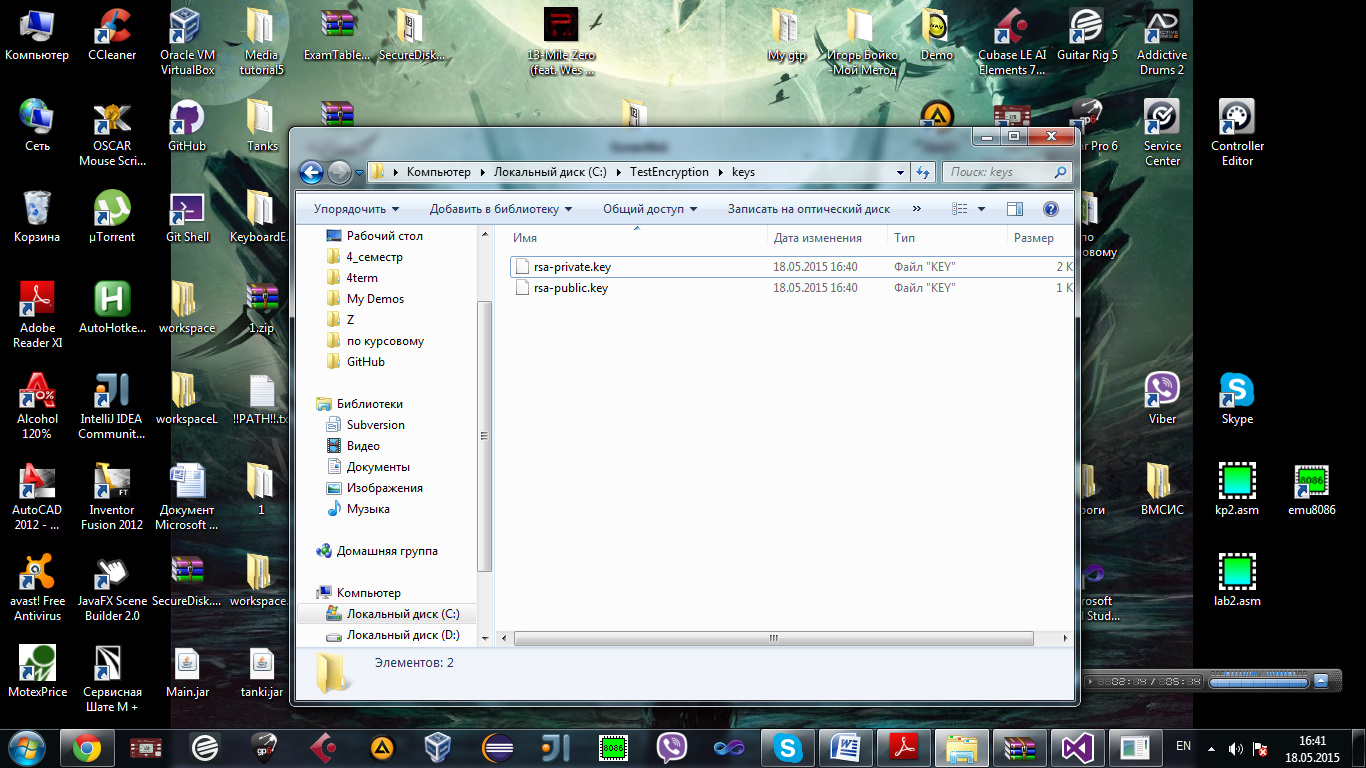


Рис. 4.3 Сгенерированные ключи

В папке “inFolder” имеем файлы, которые нужно зашифровать (см. рис. 4.4). В папке “keys” имеем закрытый и открытый ключи (см. рис. 4.3).. Выбираем папку для создания зашифрованного хранилища (папка “encryptedStorage”). Вводим пути к папке, хранилищу и ключу в соответствующие поля (см. рис 4.5).

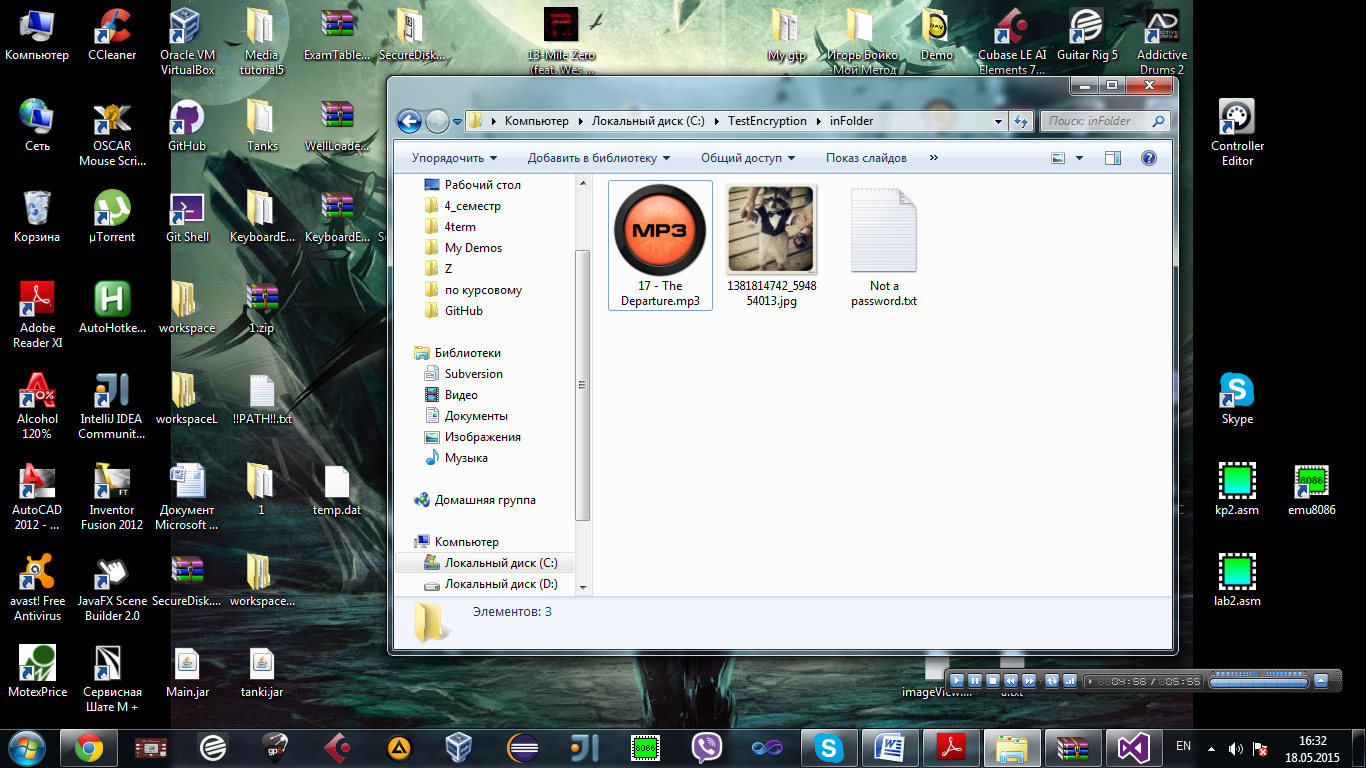


Рис. 4.4 Папка с входными файлами



Рис. 4.5 Окно шифрования

Нажимаем кнопку “Зашифровать”. Зашифрованное хранилище появится по выбранному пути (см. рис 4.6).

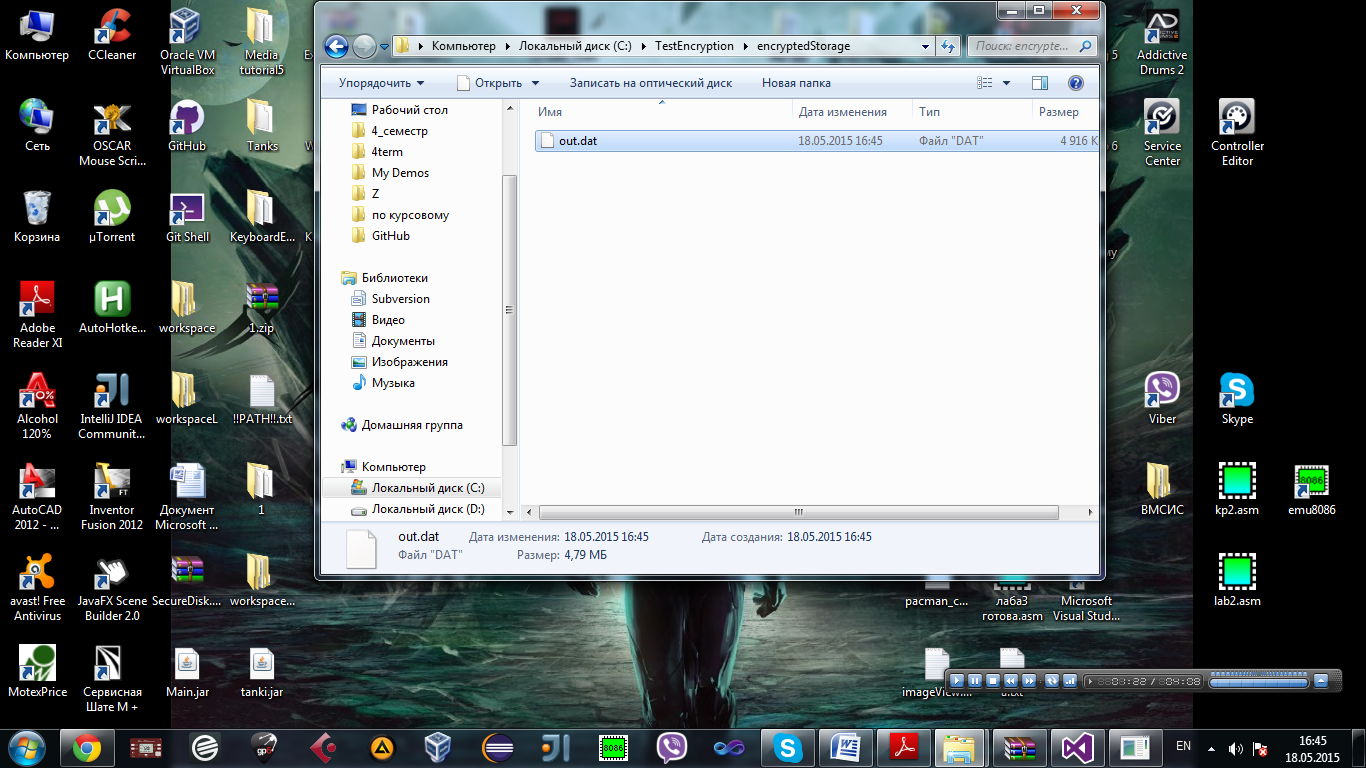


Рис. 4.6 Зашифрованное хранилище

Поля вкладки “Tab2” похожи на поля “Tab1”. Вводим пути к хранилищу, закрытому ключу и папке, в которую мы хотим поместить расшифрованное хранилище в соответствующие поля (см. рис. 4.7).

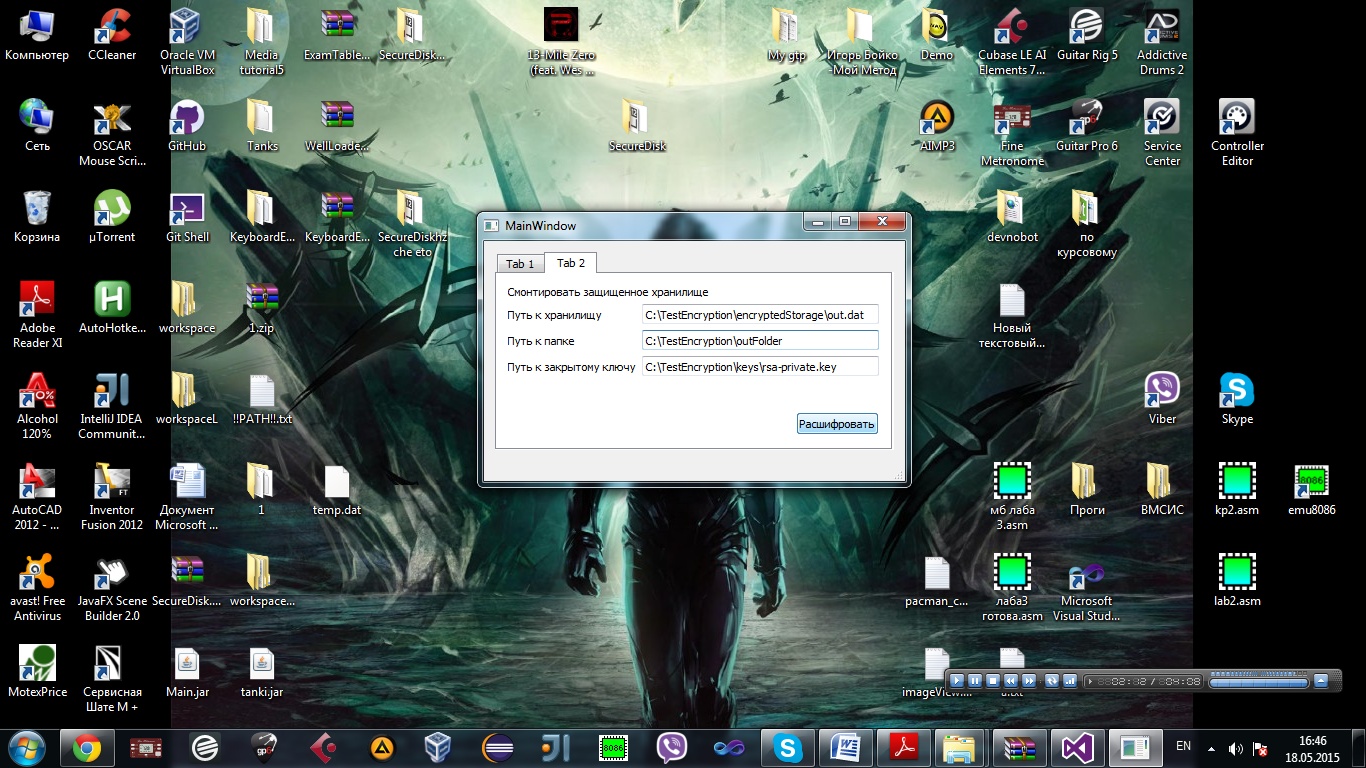


Рис. 4.7 Окно дешифрования

В выбранной папке появились расшифрованные файлы (см. рис 4.7).

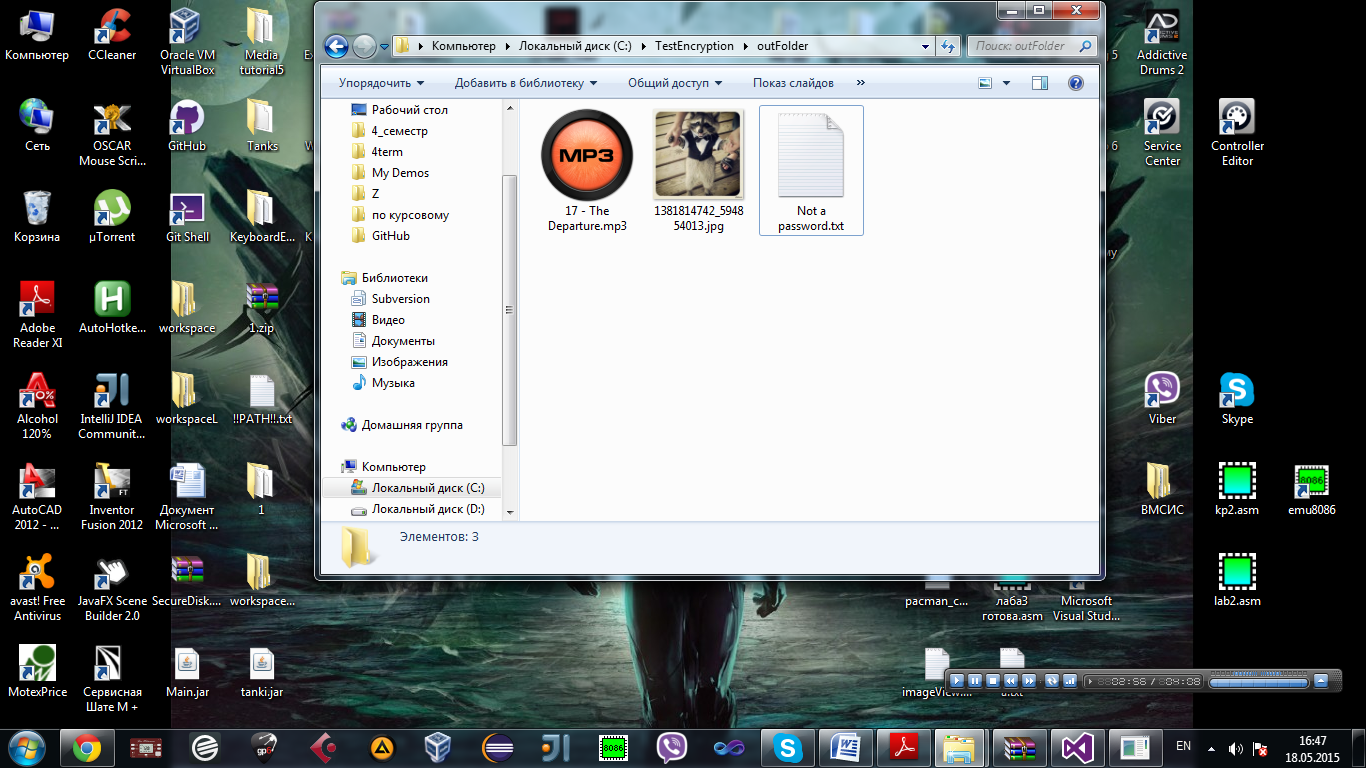


Рис. 4.7 Папка с выходными файлами

# СПИСОК ЛИТЕРАТУРЫ

1. https://ru.wikipedia.org/wiki/RSA

2. https://ru.wikipedia.org/wiki/Message\_authentication\_code

3. <http://www.cryptopp.com/>

4. http://en.wikipedia.org/wiki/Public-key\_cryptography

5. https://en.wikipedia.org/wiki/Message\_authentication\_code

# ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является приложение SecureDisk, предназначенное для шифрования и дешифрования файлов гибридным методом: RSA + MAC. Приложение позволяет предотвратить утечку важной информации (от паролей, до секретных разработок).

Зашифрованное хранилище занимает значительно больше пространства жёсткого диска, чем исходные файлы. Поэтому в перспективе проекта улучшение алгоритма шифрования или последующая архивация хранилища.

# ПРИЛОЖЕНИЕ

Здесь представлены ключевые фрагменты файлов с «исходными текстами.

* Файл securedisk.h:

#ifndef SECUREDISK\_H

#define SECUREDISK\_H

#include <QtWidgets/QMainWindow>

#include "ui\_securedisk.h"

class SecureDisk : public QMainWindow

{

Q\_OBJECT

public:

SecureDisk(QWidget \*parent = 0);

~SecureDisk();

private slots:

void encryptFolder();

void decryptStorage();

void generateKeys();

private:

Ui::SecureDiskClass ui;

};

#endif // SECUREDISK\_H

* Файл securedisk.cpp:

#include "securedisk.h"

#include "cryptohandler.h"

#include "storage.h"

#include "generatekey.h"

#include <qdir.h>

SecureDisk::SecureDisk(QWidget \*parent)

: QMainWindow(parent)

{

ui.setupUi(this);

connect(ui.encryptPushButton, SIGNAL(clicked()), this, SLOT(encryptFolder()));

connect(ui.decryptPushButton, SIGNAL(clicked()), this, SLOT(decryptStorage()));

connect(ui.generateKeysPushButton, SIGNAL(clicked()), this, SLOT(generateKeys()));

}

SecureDisk::~SecureDisk()

{

}

void SecureDisk::encryptFolder()

{

QString folderPath = ui.folderPathLineEdit->text();

QString storagePath = ui.storagePathLineEdit->text();

QString publickKeyPath = ui.publicKeyPathLineEdit->text();

QString tempFile = folderPath + "\\" + "temp.dat";

QDir directory(folderPath);

QStringList files = directory.entryList(QDir::NoDotAndDotDot | QDir::System | QDir::Hidden | QDir::AllDirs | QDir::Files, QDir::DirsFirst);

Storage storage;

storage.open(tempFile);

for each (QString file in files)

{

QFile\* fileHandler = new QFile(folderPath + "\\" + file);

fileHandler->open(QIODevice::ReadOnly);

storage.put(fileHandler, file);

fileHandler->close();

delete fileHandler;

}

storage.close();

CryptoHandler::EncryptFile(publickKeyPath, tempFile, storagePath);

QFile::remove(tempFile);

}

void SecureDisk::decryptStorage()

{

QString folderPath = ui.folderPathLineEdit\_2->text();

QString storagePath = ui.storagePathLineEdit\_2->text();

QString privateKeyPath = ui.privateKeyPathLineEdit->text();

QString tempFile = folderPath + "\\temp.dat";

CryptoHandler::DecryptFile(privateKeyPath, storagePath, tempFile);

Storage storage;

storage.open(tempFile);

QStringList files = storage.getNames();

for each (QString file in files)

{

QFile\* fileHandler = new QFile(folderPath + "\\" + file);

fileHandler->open(QIODevice::ReadOnly);

storage.out(fileHandler, file);

fileHandler->close();

delete fileHandler;

}

storage.close();

QFile::remove(tempFile);

}

void SecureDisk::generateKeys()

{

GenerateKey dialog;

dialog.exec();

}

* Файл cryptohandler.h:

#ifndef CRYPTOHANDLER\_H

#define CRYPTOHANDLER\_H

#include <QObject>

using namespace std;

class CryptoHandler : public QObject

{

Q\_OBJECT

public:

CryptoHandler(QObject \*parent);

~CryptoHandler();

static string randomStrGen(int length);

static string RSAEncryptString(string publickeyfile,

string message);

static string RSADecryptString(string privatekeyfile,

string ciphertext);

static void EncryptFile(QString publickeyfile,

QString filetoencrypt,

QString encryptedfile);

static void DecryptFile(QString privatekeyfile,

QString filetodecrypt,

QString decryptedfile);

private:

};

#endif // CRYPTOHANDLER\_H

* Файл cryptohandler.cpp:

#include "cryptohandler.h"

#include "cryptocpp\config.h"

#include "cryptocpp\cryptlib.h"

#include "cryptocpp\default.cpp"

#include "cryptocpp\files.h"

#include "cryptocpp\rsa.h"

#include "cryptocpp\randpool.h"

#include "cryptocpp\hex.h"

#include <cryptocpp\osrng.h>

#include <cryptocpp\modes.h>

using namespace CryptoPP;

static OFB\_Mode<AES>::Encryption s\_globalRNG;

RandomNumberGenerator & GlobalRNG()

{

return s\_globalRNG;

}

CryptoHandler::CryptoHandler(QObject \*parent)

: QObject(parent)

{

}

CryptoHandler::~CryptoHandler()

{

}

string CryptoHandler::randomStrGen(int length)

{

static string charset

= "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";

string result;

result.resize(length);

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

{

result[i] = charset[rand() % charset.length()];

}

return result;

}

string CryptoHandler::RSAEncryptString(string publickeyfile,

string message)

{

string seed = randomStrGen(50);

FileSource pubFile(publickeyfile.c\_str(), true, new HexDecoder());

RSAES\_OAEP\_SHA\_Encryptor pub(pubFile);

RandomPool randPool;

//randPool.Put((byte \*)seed.c\_str(), strlen(seed.c\_str()));

randPool.IncorporateEntropy((byte \*)seed.c\_str(),

strlen(seed.c\_str()));

string result;

StringSource(message.c\_str(), true,

new PK\_EncryptorFilter(

randPool,

pub,

new HexEncoder(new StringSink(result))));

return result;

}

string CryptoHandler::RSADecryptString(string privatekeyfile,

string ciphertext)

{

FileSource privFile(privatekeyfile.c\_str(), true, new HexDecoder);

RSAES\_OAEP\_SHA\_Decryptor priv(privFile);

std::string seed2 = IntToString(time(NULL));

seed2.resize(16);

s\_globalRNG.SetKeyWithIV((byte \*)seed2.data(), 16, (byte \*)seed2.data());

string result;

StringSource(ciphertext, true, new HexDecoder(new PK\_DecryptorFilter(GlobalRNG(), priv, new StringSink(result))));

return result;

}

void CryptoHandler::EncryptFile(QString publickeyfile,

QString filetoencrypt,

QString encryptedfile)

{

// Generate passphrase

string passphrase = randomStrGen(50);

string encrypted\_passphrase = RSAEncryptString(publickeyfile.toStdString(),

passphrase);

// Encrypt the file and save encrypted data to the string

string data;

FileSource f(filetoencrypt.toStdString().c\_str(), true,

new DefaultEncryptorWithMAC(

passphrase.c\_str(),

new HexEncoder(new StringSink(data))));

// Append encrypted passphrase to the beginning of the string

data = encrypted\_passphrase + data;

// Save string to the file

StringSource ss(data.c\_str(), true,

new FileSink(encryptedfile.toStdString().c\_str()));

}

void CryptoHandler::DecryptFile(QString privatekeyfile,

QString filetodecrypt,

QString decryptedfile)

{

// Read string from file

std::string data;

FileSource ff(filetodecrypt.toStdString().c\_str(), true, new StringSink(data));

// Grab passphrase from the string

string encrypted\_passphrase = data.substr(0, 256);

encrypted\_passphrase.resize(256);

string passphrase = RSADecryptString(privatekeyfile.toStdString(),

encrypted\_passphrase);

// Grab cipher from the string

std::string cipher = data.substr(256, data.length()).c\_str();

// Decrypt string and save to file

StringSource s(cipher.c\_str(), true,

new HexDecoder(new DefaultDecryptorWithMAC(

passphrase.c\_str(),

new FileSink(decryptedfile.toStdString().c\_str()))));

}

* Файл storage.h:

#ifndef STORAGE\_H

#define STORAGE\_H

#include <QObject>

#include <QFile>

#include <QStringList>

struct fileInfo

{

char name[32];

qint64 size;

fileInfo(void) {}

fileInfo(const char\* name, qint64 size)

{

strcpy(this->name, this->getName(name));

this->size = size;

}

const char\* getName(const char\* fileName)

{

const char\* str = fileName + strlen(fileName);

int sz = strlen(fileName);

while (\*str != '\\' && sz--)

--str;

return str;

}

int length(void) const

{

return sizeof(name)+sizeof(size);

}

};

class Storage : public QObject

{

Q\_OBJECT

public:

explicit Storage(QObject \*parent = 0);

QFile\* open(QString path);

void close();

void put(QFile\* file, QString filename);

void out(QFile\* file, QString filename);

QStringList getNames();

signals:

public slots :

private:

QFile\* \_storage;

};

#endif // STORAGE\_H

* Файл storage.cpp:

#include "storage.h"

Storage::Storage(QObject \*parent) :

QObject(parent), \_storage(new QFile())

{

}

QFile\* Storage::open(QString path)

{

\_storage = new QFile(path);

\_storage->open(QIODevice::ReadWrite | QIODevice::Append);

return \_storage;

}

void Storage::close()

{

\_storage->close();

}

void Storage::put(QFile\* file, QString filename)

{

char\* buf;

fileInfo info(filename.toStdString().c\_str(), file->size());

file->open(QIODevice::ReadOnly);

int fileSize = file->size();

if (file->isOpen())

{

\_storage->write((const char\*)&info, info.length());

buf = new char[fileSize];

file->read(buf, fileSize);

\_storage->write(buf, fileSize);

\_storage->flush();

file->close();

}

}

void Storage::out(QFile\* fp, QString fileName)

{

fileInfo info;

fp->open(QIODevice::WriteOnly);

char\* buf;

quint64 pos = 0;

while (\_storage->seek(pos))

{

memset(&info, 0, sizeof(fileInfo));

\_storage->read((char\*)&info, sizeof(fileInfo));

if (info.size == 0)

break;

else

{

if (fileName == QString(info.name))

{

buf = new char[info.size];

\_storage->read(buf, info.size);

fp->write(buf, info.size);

delete[] buf;

}

pos += (info.size + sizeof(fileInfo));

}

}

fp->close();

}

QStringList Storage::getNames()

{

QStringList fileList;

fileInfo info;

quint64 pos = 0;

\_storage->seek(0);

while(\_storage->seek(pos))

{

memset(&info, 0, sizeof(fileInfo));

\_storage->read((char\*)&info, sizeof(fileInfo));

if (info.size == 0)

break;

else

{

fileList << QString(info.name);

pos += (info.size + sizeof(fileInfo));

}

}

return fileList;

}

* Файл generatekey.h:

#ifndef GENERATEKEY\_H

#define GENERATEKEY\_H

#include <QDialog>

#include "ui\_generatekey.h"

#include "cryptocpp\rsa.h"

#include "cryptocpp\osrng.h"

#include "cryptocpp\files.h"

#include "cryptocpp\hex.h"

#define PUBLIC\_KEY "rsa-public.key"

#define PRIVATE\_KEY "rsa-private.key"

using namespace CryptoPP;

class GenerateKey : public QDialog

{

Q\_OBJECT

public:

GenerateKey(QWidget \*parent = 0);

~GenerateKey();

private:

void SavePublicKey(const QString& filename, const PublicKey& key);

void SavePrivateKey(const QString& filename, const PrivateKey& key);

void Save(const QString& filename, const BufferedTransformation& bt);

void SaveHex(const QString& filename, const BufferedTransformation& bt);

private slots:

void accept();

private:

Ui::GenerateKey ui;

};

#endif // GENERATEKEY\_H

* Файл generatekey.cpp:

#include "generatekey.h"

GenerateKey::GenerateKey(QWidget \*parent)

: QDialog(parent)

{

ui.setupUi(this);

connect(ui.buttonBox, SIGNAL(accepted()), this, SLOT(accept()));

connect(ui.buttonBox, SIGNAL(rejected()), this, SLOT(close()));

}

GenerateKey::~GenerateKey()

{

}

void GenerateKey::SavePublicKey(const QString& filename, const PublicKey& key)

{

ByteQueue queue;

key.Save(queue);

SaveHex(filename, queue);

}

void GenerateKey::SavePrivateKey(const QString& filename, const PrivateKey& key)

{

ByteQueue queue;

key.Save(queue);

SaveHex(filename, queue);

}

void GenerateKey::Save(const QString& filename, const BufferedTransformation& bt)

{

FileSink file(filename.toStdString().c\_str());

bt.CopyTo(file);

file.MessageEnd();

}

void GenerateKey::accept()

{

AutoSeededRandomPool rnd;

QString path = ui.keyPathLineEdit->text();

RSAES\_OAEP\_SHA\_Decryptor rsaPrivate;

rsaPrivate.AccessKey().GenerateRandomWithKeySize(rnd, 1024);

RSAES\_OAEP\_SHA\_Encryptor rsaPublic(rsaPrivate);

SavePrivateKey(path + QString("\\") + QString(PRIVATE\_KEY), rsaPrivate.AccessKey());

SavePublicKey(path + QString("\\") + QString(PUBLIC\_KEY), rsaPublic.AccessKey());

close();

}

void GenerateKey::SaveHex(const QString& filename, const BufferedTransformation& bt)

{

HexEncoder encoder;

bt.CopyTo(encoder);

encoder.MessageEnd();

Save(filename, encoder);

}