Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

ПРИЛОЖЕНИЕ ДЛЯ РАБОТЫ С ШИФРОВАННЫМИ ФАЙЛАМИ

БГУИР КП 1-40 01 01 015 ПЗ

Студент Маталыга Е.А.

Руководитель Алексеев И.Г.

Минск 2024

# **ВВЕДЕНИЕ**

В современном цифровом мире безопасность информации становится одной из ключевых задач, стоящих перед пользователями и организациями. С ростом объема передаваемых и хранимых данных возрастает необходимость в надежных методах защиты конфиденциальной информации. Шифрование файлов представляет собой один из самых эффективных способов обеспечить безопасность данных, позволяя защитить их от несанкционированного доступа и утечек.

Цель данного курсового проекта заключается в разработке приложения для работы с шифрованными файлами, которое будет обеспечивать пользователям простоту и удобство в использовании, а также высокую степень безопасности. Приложение будет включать функциональность для шифрования и дешифрования файлов, а так же создания и управления хранилищами зашифрованных файлов.

Задачи курсовой работы включают:

1. Реализация алгоритма шифрования и дешифрования
2. Обеспечение безопасного хранения ключей
3. Создание пользовательского интерфейса для удобной работы с приложением.
4. Проведение тестирования и анализа безопасности разработанного приложения

Ожидается, что разработанное приложение станет полезным инструментом для пользователей, стремящихся защитить свои данные от потенциальных угроз, и повысит осведомленность о важности применения современных методов шифрования.

# **1 АНАЛИЗ ПРОТОТИПОВ**

## **1.1 Cryptomator**

Разработанное в ходе выполнения курсовой работы приложение основано на уже существующем приложении для работы с шифрованными файлами «Cryptomator».

Cryptmator является полностью бесплатным и свободным ПО, с открытым исходным кодом, для шифрования данных на стороне клиента.

К плюсам Cryptomator можно отнести:

1. Использование AES с 256 битным ключом
2. Пароль от хранилища защищен от брут-форс атак с помощью scrypt
3. Создание и управление несколькими хранилищами одновременно
4. Может работать с облачными хранилищами (при условии что на компьютере уже смонтирована папка облачного хранилища)
5. Прост в установке, прост в использовании

С помощью Cryptomator можно надежно защитить свои данные, поместив их в зашифрованные хранилища. Получить доступ к данным может только тот, кто знает пароль к хранилищу. При открытии хранилища с помощью Cryptomator, оно будет отображаться как виртуальный накопитель. С ним можно работать как с обычной USB-флешкой.

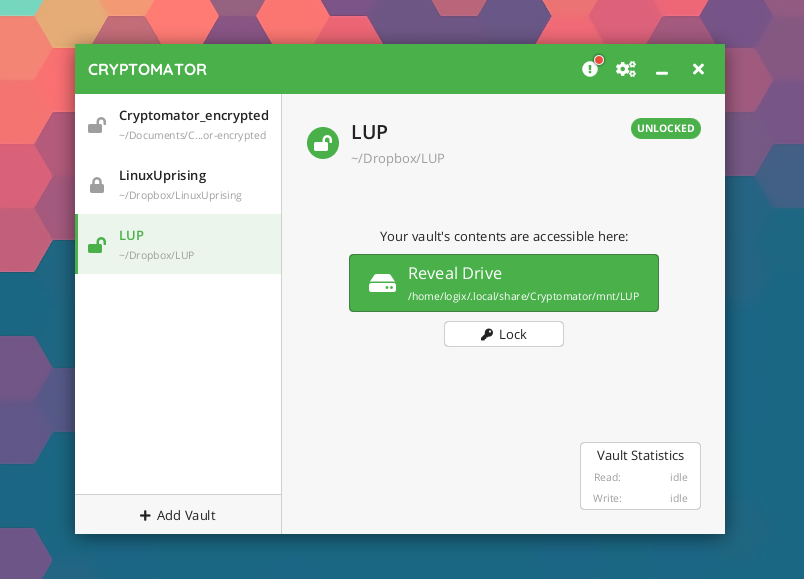


Рис. 1.1 – Интерфейс приложения Cryptomator

Зашифрованные данные хранятся в обычной папке. Если открыть хранилище без Cryptomator, то будет отображён только набор каталогов и файлов с непонятными именами. Распознать что это хранилище Cryptomator можно по файлу masterkey.cryptomator, который находится внутри папки.

## **1.2 Encrypto**

Encrypto – приложение для шифрования и дешифрования файлов с минимальным функционалом. После запуска приложения на экране компьютера появляется небольшое окно. Туда нужно переместить файл, который небходимо зашифровать. Переместив его, необходимо ввести пароль и кодовый вопрос (по желанию). Приложение начнёт процесс шифровки, по окончании которого представит вам файл с расширением .crypto. Для расшифровки файла достаточно ввода ключа в приложении.

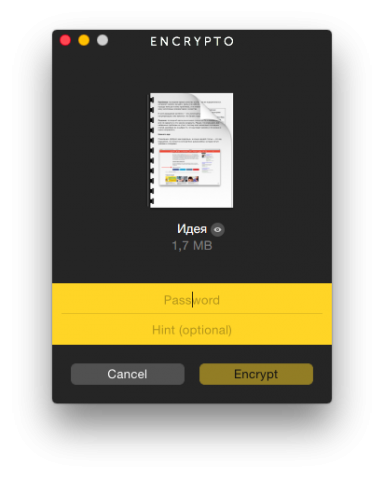


Рис. 1.2 – Интерфейс приложения Encrypto

К достоинствам Encrypto можно отнести:

1 Использование AES с 256 битным ключом;

2 Простота в установке и использовании;

3 Высокая скорость работы.

# **2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ К РАЗРАБАТЫВАЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

# **2.1 Описание и спецификация функциональных требований**

Основные требования к функционалу программного средства:

1 Корректное шифрование и дешифрование файлов;

2 Работа с файлами любых форматов;

3. Создание хранилищ;

4 Работа с файлами хранилища:

− Добавление;

− Удаление;

− Экспорт;

5 Обработка удаления хранилища;

6 Безопасное хранение ключей к хранилищу;

7 Безопасное хранение файлов хранилища.

## **2.2 Используемые ресурсы**

В данном курсовом проекте для разработки приложения, создающего зашифрованные хранилища, будут использованы язык программирования C++ и WinAPI. C++ обеспечивает высокую производительность и гибкость, позволяя разрабатывать эффективный код, в то время как WinAPI предоставляет необходимые функции для работы с интерфейсом Windows и системными ресурсами. Стандартная библиотека C++ упростит работу с данными и вводом/выводом. Для компиляции и отладки приложения будет использована интегрированная среда разработки Microsoft Visual Studio, что обеспечит удобство работы и доступ к инструментам для тестирования.

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **3.1 Структура программы**

Вся логика работы приложения содержится в файле «CryptoPanda.cpp». Помимо основного файла используется так же класс Encoder, в котором реализована логика генерации ключевой последовательности, шифрование и дешифрование файла.

## **3.2 Проектирование пользовательского интерфейса**

При разработке программного средства за основу будет взят дизайн приложения Crypromator.

Среди основных действий пользователя можно выделить следующие:

1 Создание хранилища;

2 Просмотр содержимого хранилища;

3 Добавление файла в хранилище;

4 Удаление файла из хранилища;

5 Экспорт файла из хранилища;

6 Блокировка и разблокировка хранилища.



Рис. 3.1 − Основные действия пользователя

**3.2.1** Главное окно

Главное окно приложение должно состоять из двух основных частей: списка уже созданных пользователем хранилищ с кнопкой создания нового хранилища и рабочей область, которая при открытии приложения является пустой, но, по мере работы в приложении, её содержимое должно меняться. Макет окна представлен на рисунке 3.2.

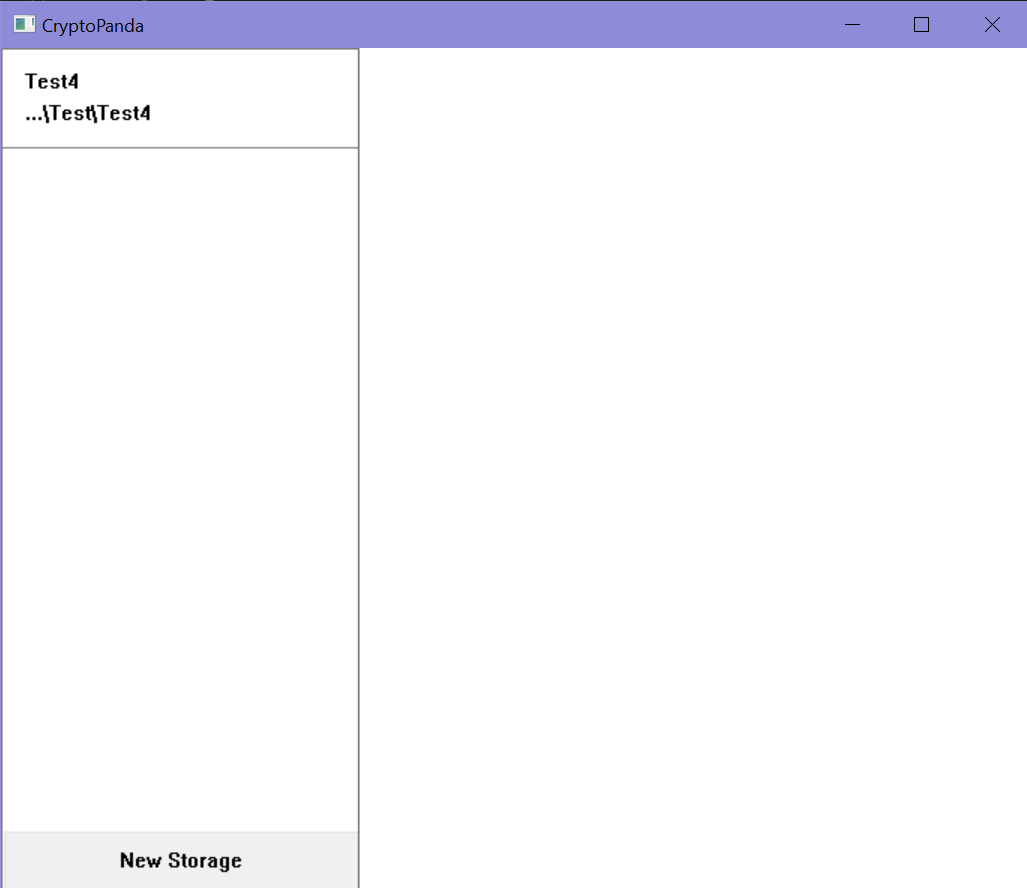


Рис 3.2 − Главное окно

**3.2.2** Окно создания хранилища

Окно создания хранилища должно открываться на месте рабочей области главного окна при нажатии на кнопку «New Storage». Для создания нового хранилища необходимо будет ввести путь к директории, в которой должно быть создано нового хранилище, имя хранилища и пароль, который будет необходим для разблокировки хранилища и которым будут шифроваться все файлы в хранилище. Макет окна представлен на рисунке 3.3.

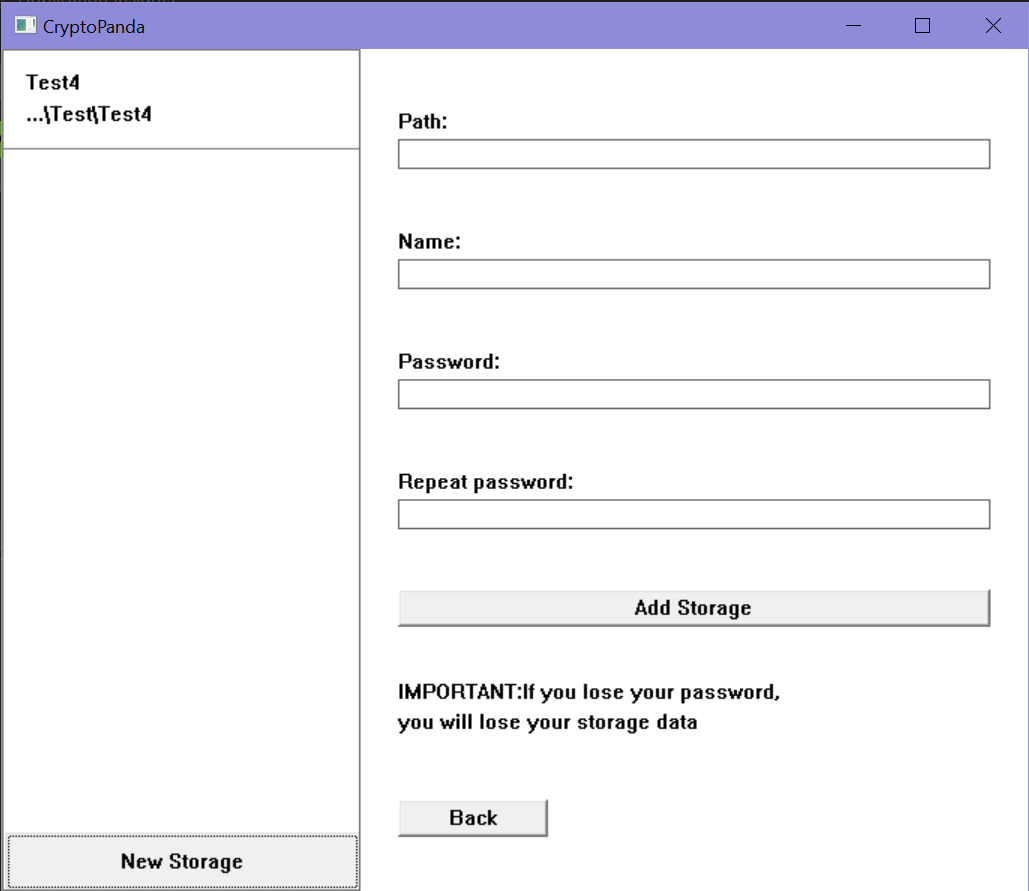


Рис. 3.3 − Окно создания нового хранилища

**3.2.3** Окно просмотра содержимого хранилища

Окно просмотра так же должно открываться на месте рабочей области основного окна. У пользователя должна быть возможность просмотреть список всех файлов, которые лежат в хранилище, с расшифрованными именами, добавить новый файл, заблокировать хранилище, чтобы для повторного входа требовался пароль, а так же удалить или экспортировать файл из хранилища в расшифрованном виде. Макет страницы представлен на рисунке 3.4.

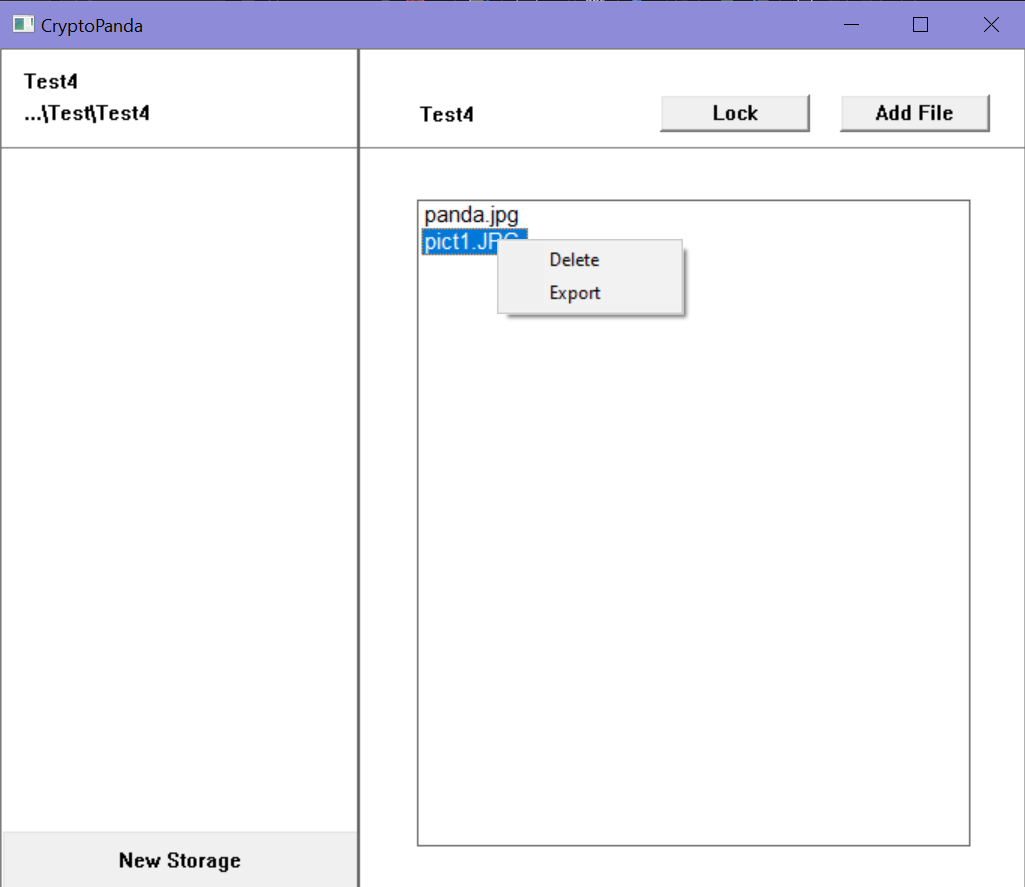


Рис. 3.4 − Окно просмотра содержимого хранилища

**3.2.4** Дополнительные окна

Помимо основных окон, в которых реализуется основная логика работы приложения, должны быть так же и дополнительные окна для ввода пароля при разблокировке приложения и добавления файла в хранилище.

Окно для ввода пароля должно указывать, доступ к какому хранилищу пользователь хочет получить доступ, и содержать всего одно поле и кнопки подтверждения и отмены операции. Макет окна представлен на рисунке 3.5.

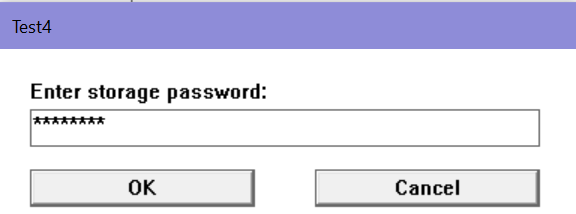


Рис. 3.5 – Макет окна для ввода пароля от хранилища

Окно добавления файла в хранилище должно иметь поля для ввода пути к файлу, который нужно зашифровать, и поле для ввода имени, под которым этот файл будет отображаться в хранилище при его открытии в приложении. В окне должны так же быть кнопки для подтверждения и отмены операции. Макет окна представлен на рисунке 3.6.

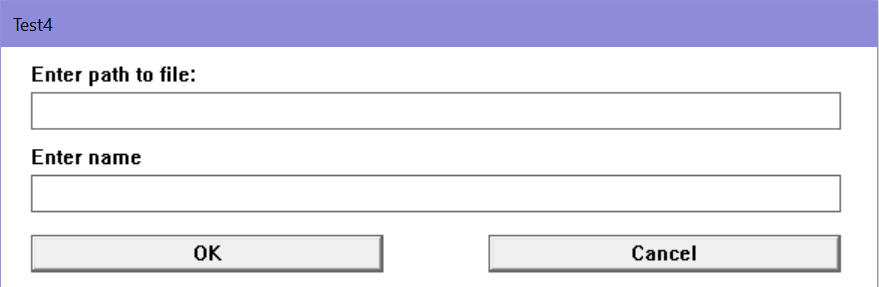


Рис. 3.6 − Окно добавления файла в хранилище

## **3.3 Проектирование функционала программного средства**

**3.3.1** Алгоритм шифрования

В данном курсовом проекте для реализации защиты данных используется потоковое шифрование с операцией XOR. Этот метод шифрования обеспечивает высокий уровень безопасности при относительно низких затратах на вычисления. Основной принцип заключается в том, что каждый бит исходных данных шифруется с помощью соответствующего бита псевдослучайной последовательности.

В качестве ключа для шифрования используется псевдослучайная последовательность, которая генерируется на основе ключа пользователя. Эта последовательность создается с помощью линейного генератора с обратной связью (LFSR), заданного определённым примитивным многочленом.

Для генерации ключевой последовательности был выбран LFSR, т.к. он достаточно просто реализуется в программном и аппаратном виде, обладает высокой скоростью генерации и большим периодом ключа. Регистр LFSR состоит из двух частей: сдвигового регистра, выполняющего сдвиг своих разрядов влево на один разряд, и функции обратной связи, вычисляющей вдвигаемое в первый разряд значение.

Преимущества такого подхода:

1 Скорость. Потоковое шифрование позволяет быстро обрабатывать данные;

2 Гибкость. Поддержка шифрования данных произвольных размеров;

3 Простота реализации. Алгоритм легко реализовать и адаптировать под различные требования;

4 Одинаковая процедура шифрования и дешифрования.

На рисунке 3.7 представлена схема шифрования и дешифрования файла и сохранения результата.

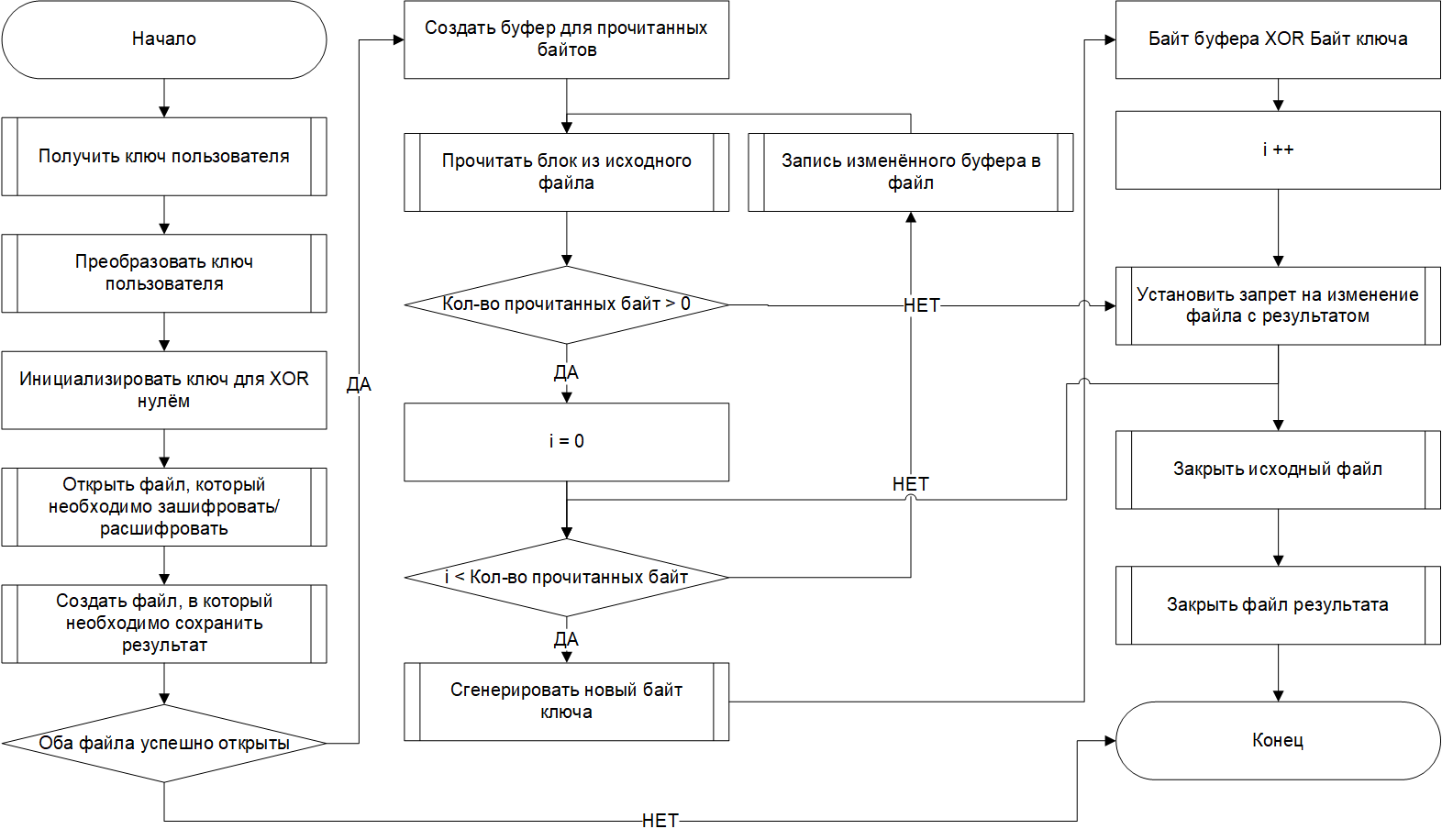


Рис. 3.7 − Алгоритм шифрования файла

**3.3.2** Хранение ключей

Все ключи, которые пользователь задаёт при создании хранилища, не хранятся в открытом виде, а шифруются при помощи DPAPI и сохраняются в зашифрованном виде в бинарном файле.

Data Protection API (DPAPI) — это встроенный механизм шифрования в операционной системе Windows, предназначенный для защиты конфиденциальной информации, такой как ключи шифрования, пароли и другие чувствительные данные. DPAPI обеспечивает простой и безопасный способ шифрования данных, используя системные ключи, которые могут быть связаны с конкретной учётной записью или машиной.

DPAPI предоставляет разработчикам API для шифрования и расшифровки данных с использованием двух основных функций:

1 CryptProtectData: Эта функция используется для шифрования данных. Она принимает открытые данные, которые нужно защитить, и возвращает зашифрованные данные.

2 CryptUnprotectData: Эта функция используется для расшифровки данных. Она принимает зашифрованные данные и возвращает исходные открытые данные.

Функции DPAPI используются на двух этапах: при создании нового хранилища и при разблокировке хранилища (рисунок 3.8).

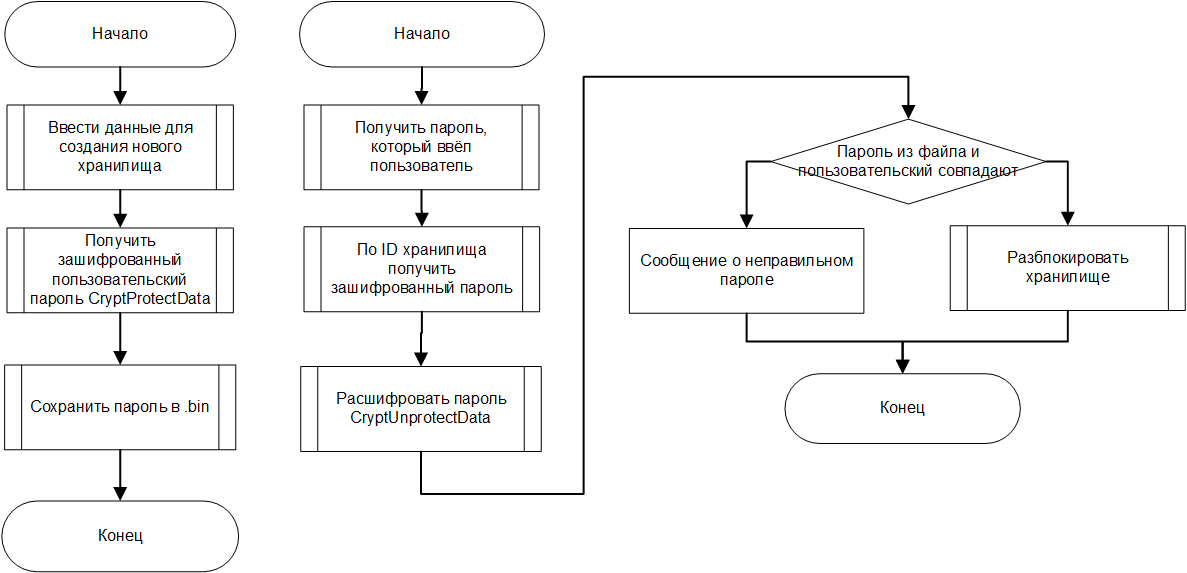


Рис. 3.8 − Использование функций DPAPI

**3.3.3** Управление хранилищем

В разблокированном хранилище пользователь может выполнять следующие действия:

1 Заблокировать хранилище, чтобы при следующем открытии требовался пароль

2 Добавить файл в хранилище

3 Удалить файл из хранилища

4 Экспортировать файл из хранилища

Схема управления хранилищем представлена на рисунке 3.9.

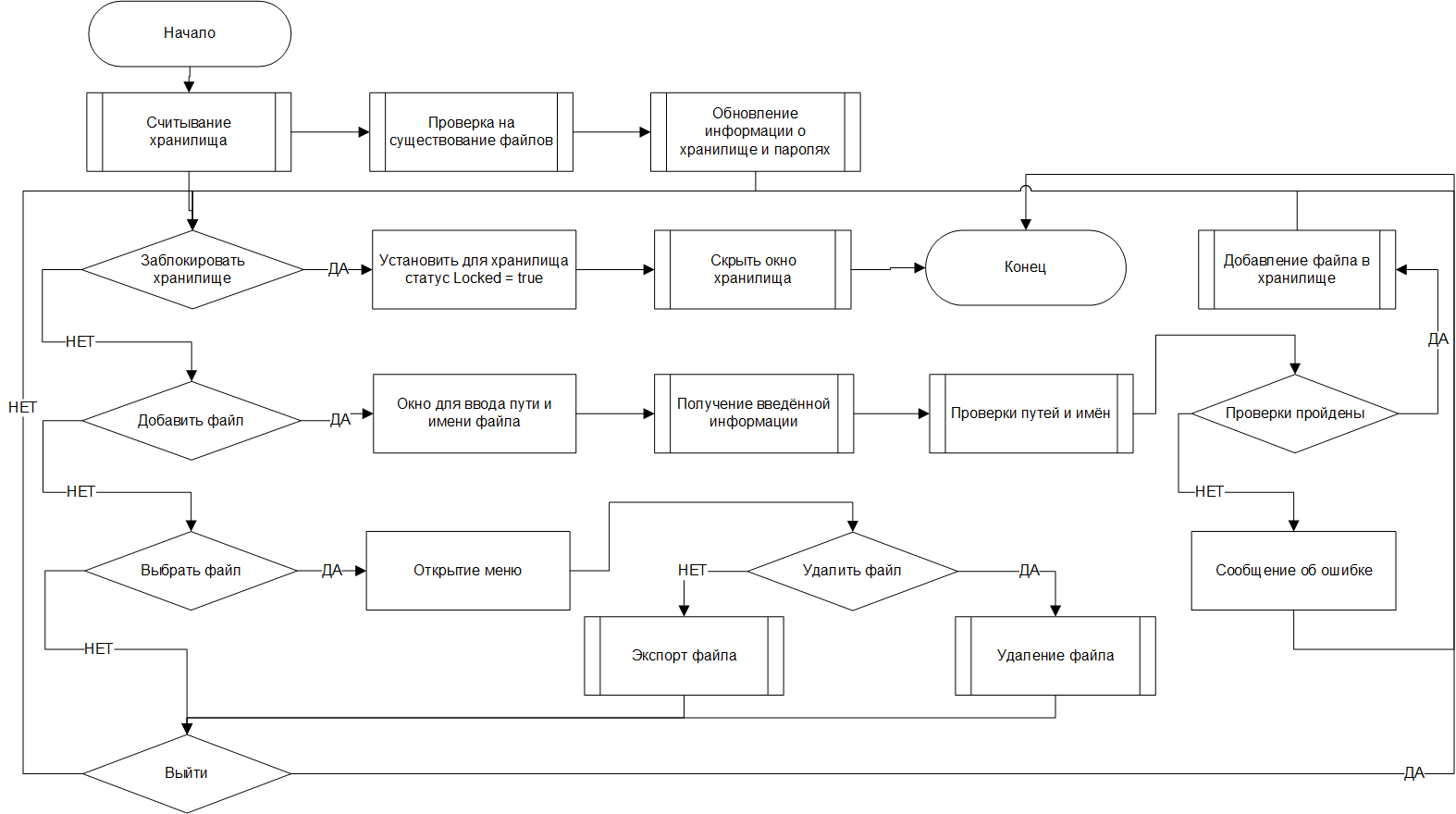


Рисунок 3.9 − Управление хранилищем

Все файлы хранилищ имеют особые настройки доступа, которые не позволяют изменять файлы, чтобы не нарушить их целостность и, как следствие, не потерять данные при расшифровке.

Однако у пользователя должна быть возможность удалить хранилище или файл. В таком случае, необходимо обеспечить корректную обработку этого удаления. Для этого каждый раз при загрузке приложения происходит анализ файла со списком хранилищ, проверка существования этих хранилищ и изменение информации о хранилищах и их паролях.

# **4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## **4.1 Шифрование**

Основная часть работы с шифрованными файлами, включающая открытие, создание файлов, шифрование и установку атрибутов безопасности, реализована в классе Encoder. Для этого используются библиотеки для работы с файловым вводом и выводом <fstream>, <filesystem>, <vector> для хранения информации, <bitset> для работы с битам, <Windows.h> для настройки прав доступа и <locale> для настройки локали. Следующий код отражает процедуру шифрования и расшифровки:

bool Encoder::EncryptDecryptFile(const std::wstring& input, const std::wstring& output) {

std::ifstream file\_in(input, std::ios::binary);

std::ofstream file\_out(output, std::ios::binary);

if (!file\_in.is\_open()) {

return false;

}

if (!file\_out.is\_open()) {

return false;

}

std::vector<uint8\_t> buffer(BlockSize);

while (file\_in.read(reinterpret\_cast<char\*>(buffer.data()), buffer.size()) || file\_in.gcount() > 0) {

int bytesRead = file\_in.gcount();

int j = 0;

for (int i = 0; i < bytesRead; i++) {

shift();

buffer[i] ^= Key;

}

file\_out.write(reinterpret\_cast<char\*>(buffer.data()), bytesRead);

}

DWORD attributes = GetFileAttributes(output.c\_str());

if (attributes != INVALID\_FILE\_ATTRIBUTES) {

SetFileAttributes(output.c\_str(), attributes | FILE\_ATTRIBUTE\_READONLY);

}

file\_in.close();

file\_out.close();

return true;

}

## **4.2 Добавление нового хранилища**

Для добавления нового хранилища необходимо сначала получить всю информацию, введённую пользователем, проверить её на корректность, проверить существование пути и обновить информацию о хранилищах в файлах и в самом приложении. Для этого также используются библиотеки для работы с файловым вводом и выводом <fstream>, <filesystem>, <vector> для хранения информации, <dpapi.h> для работы с DPAPI. Следующий код реализует механизм добавления нового хранилища:

void AddNewStorage() {

//RECT clearRect = { stListWndWidth, 0, clientWidth - stListWndWidth, stListWndHeight };

int x = GetWindowTextLength(newDirInput);

if (x == 0) {

MessageBox(mainWnd, L"Enter path", L"failed", MB\_OK);

return;

}

std::wstring directory(x, L'\0');

GetWindowText(newDirInput, &directory[0], x + 1);

x = GetWindowTextLength(newNameInput);

if (x == 0) {

MessageBox(mainWnd, L"Enter name", L"failed", MB\_OK);

return;

}

if (x > 256 - directory.size()) {

MessageBox(mainWnd, L"Name is too long", L"failed", MB\_OK);

return;

}

std::wstring name(x, L'\0');

GetWindowText(newNameInput, &name[0], x + 1);

if (СontainsInvalidChars(name)) {

MessageBox(mainWnd, L"Name contains invalid chars", L"failed", MB\_OK);

return;

}

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

{

if (storages[i].name == name)

{

MessageBox(mainWnd, L"Storage with this name already exists", L"failed", MB\_OK);

return;

}

}

x = GetWindowTextLength(newPassInput);

if (x > 32) {

MessageBox(mainWnd, L"Password is too long", L"failed", MB\_OK);

return;

}

if (x < 8) {

MessageBox(mainWnd, L"Password is too short", L"failed", MB\_OK);

return;

}

if (x == 0) {

MessageBox(mainWnd, L"Enter Password", L"failed", MB\_OK);

return;

}

std::wstring password(x, L'\0');

GetWindowText(newPassInput, &password[0], x + 1);

x = GetWindowTextLength(newPassRepInput);

if (x <= 0 || x > 22) {

MessageBox(mainWnd, L"Mismatch", L"failed", MB\_OK);

return;

}

std::wstring passwordRep(x, L'\0');

GetWindowText(newPassRepInput, &passwordRep[0], x + 1);

if (password.compare(passwordRep) != 0) {

MessageBox(mainWnd, L"Mismatch", L"failed", MB\_OK);

return;

}

std::wstring fullPath = directory + L"\\" + name;

if (!CreateDirectory(fullPath.c\_str(), NULL) && GetLastError() != ERROR\_ALREADY\_EXISTS) {

MessageBox(mainWnd, L"Failed to create directory", L"failed", MB\_OK);

return;

}

SaveToFile(L"CPKeys.bin", ProtectData(reinterpret\_cast<const BYTE\*>(password.c\_str()), password.size() \* sizeof(wchar\_t)));

std::wofstream outFile("CPStorages.txt", std::ios::app);

if (!outFile.is\_open()) {

MessageBox(mainWnd, L"Failed to open CPStorages.txt for writing", L"failed", MB\_OK);

return;

}

outFile.imbue(std::locale(".utf-8"));

outFile << fullPath << std::endl;

outFile.close();

storages.clear();

ReadEntriesFromFile(L"D:\\БГУИР\\Курсовые\\CryptoPanda\\CryptoPanda\\CPStorages.txt");

}

## **4.3 Добавление файла в хранилище**

Добавление файла в хранилище так же подразумевает проверки пользовательского ввода, действительность путей, непосредственно добавление файла в хранилище и обновление информации о списке файлов в хранилище. Следующая функция реализует добавление нового файла в хранилище:

void AddNewFileInStorage()

{

int x = GetWindowTextLength(newFilePathEdit);

if (x == 0)

{

MessageBox(mainWnd, L"Enter path", L"failed", MB\_OK);

return;

}

std::wstring direcory(x, L'\0');

GetWindowText(newFilePathEdit, &direcory[0], x + 1);

x = GetWindowTextLength(newFileNameEdit);

if (x == 0)

{

MessageBox(mainWnd, L"Enter name", L"failed", MB\_OK);

return;

}

std::wstring name(x, L'\0');

GetWindowText(newFileNameEdit, &name[0], x + 1);

for (int i = 0;i < filesInStorage.size();i++) {

if (filesInStorage[i].name == name) {

MessageBox(viewStorageWnd, L"This file name is already in storage.Choose another one", L"Failed", MB\_OK);

return;

}

}

std::wstring key = GetPassword();

encoder->SetKey(key);

std::wstring encodedName = L" ";

encodedName = encoder->EncryptLine(name);

encoder->SetKey(key);

std::wstring newFilePath = storages[storageID].path + L"\\" + encodedName + L".txt";

if (std::filesystem::exists(direcory) && encoder->EncryptDecryptFile(direcory, newFilePath))

{

AddItemInFileTable(name);

filesInStorage.push\_back({name, encodedName});

}

else MessageBox(viewStorageWnd, L"FilePath doesn't exist", L"Faild", MB\_OK);

}

## **4.4 Отображение файлов в хранилище**

При открытии хранилища не в приложении, пользователь увидит зашифрованные файлы с зашифрованными именами. Для обнаружения какого-либо зашифрованного файла, его настоящее имя нужно найти в хранилище, открытом в приложении. Для отображения списка файла в понятном для пользователя виде, необходимо получить ключ пользователя и расшифровать им имена всех файлов хранилища. Следующая функция реализует описанный алгоритм:

void GetFilesInStorage() {

std::wstring key = GetPassword();

std::wstring dirPath = storages[storageID].path;

try {

for (const std::filesystem::directory\_entry& entry : std::filesystem::recursive\_directory\_iterator(dirPath)) {

if (std::filesystem::is\_regular\_file(entry.status())) {

FileNames filename;

std::filesystem::path filePath = entry.path(); // Получаем путь к файлу

std::wstring filenameWithoutExtension = filePath.stem().wstring();

encoder->SetKey(key);

filename.encodedName = filenameWithoutExtension;

std::wstring name = encoder->EncryptLine(filenameWithoutExtension);

filename.name = name;

filesInStorage.push\_back(filename);

}

}

}

catch (const std::filesystem::filesystem\_error& e) {

MessageBox(viewStorageWnd, L"Can not reade storage", L"Failed", NULL);

}

}

## **4.5 Вспомогательные функции**

Таблица 4.1 − Вспомогательные функции

|  |  |  |
| --- | --- | --- |
| Модуль | Прототип функции | Назначение |
| Encoder.cpp | void Encoder::shift() | Формирует байт ключевой последовательности |
| Encoder.cpp | std::wstring Encoder::EncryptLine(const std::wstring& input) | Используется для шифрования и расшифровки имён файлов |
| CryproPanda.cpp | DeleteFileFromStorage(int fileId) | Удаляет выбранный файл из хранилища |
| CryproPanda.cpp | bool ExportFileFromStorage(int fileId) | Экспортирует выбранный файл из хранилища в расшифрованном виде в папку Export |
| CryproPanda.cpp | std::wstring GetPassword() | Получение ключа хранилища при помощи DPAPI |
| CryproPanda.cpp | bool ComparePassword(const std::wstring& inputPassword, const std::wstring& filename, size\_t index) | Сравнивает пароль, введённый пользователем, и пароль хранилища |
| CryproPanda.cpp | bool ComparePassword(const std::wstring& inputPassword, const std::wstring& filename, size\_t index) | Обновляет информацию о паролях, если хранилище было удалено |

# **5. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Для обеспечения надежности и безопасности разработанного приложения были проведены различные тесты, направленные на проверку корректности его работы и устойчивости к ошибкам. В следующей таблице представлены результаты тестирования ключевых функций приложения.

Таблица включает в себя следующие параметры: название теста, описание, ожидаемый результат и фактический результат.

Таблица 5.1 − Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Название функции | Действия пользователя | Ожидаемое поведение системы | Результат |
| Ввод верного пароля хранилища | Ввод пользователем верной последовательности в поле для ввода и нажатие на кнопку OK | Разблокировка хранилища. Отображение содержимого хранилища на экране | Пройден |
| Ввод неверного пароля хранилища | Ввод пользователем неверной последовательности в поле для ввода и нажатие на кнопку OK | Сообщение о неверном пароле | Пройден |
| Открытие окна добаввления нового хранилища | Нажатие пользователем кнопки «New Storage» | Обновление рабочей области. Отображение полей для ввода информации о новом хранилище | Пройден |
| Добавление нового хранилища. Корректные данные | Ввод пользователем корректной информации о создаваемом хранилище. Нажатие на кнопку «Add Storage» | Создание нового хранилища. Обновление списка хранилищ. Сохранение информации о новом хранилище | Пройден |

Таблица 5.1 − Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Добавление нового хранилища. Несуществующий путь | Ввод пользователем пути к несуществующей папке | Сообщение об ошибке создания нового хранилища | Пройден |
| Добавление нового хранилища. Повторное создание | Ввод пользователем такой же информации, как для уже сущетсвующего хранилища | Сообщение об ошибке создания нового хранилища | Пройден |
| Добавление нового хранилища. Не совпадают пароли | Ввод пользователем отличающихся паролей в поле создание пароля и его подтверждения | Сообщение о несовпадении паролей | Пройден |
| Добавление нового хранилища. Закрытие окна | Выход из окна создания хранилища без создания при помощи кнопки «Back» | Открытие рабочей области главного окна | Пройден |
| Блокировка хранилища | Нажатие пользователем кнопки Lock | Блокировка хранилища. Открытие рабочей области главного окна | Пройден |
| Добавление файла в хранилище. Корректные данные | Ввод пользователем существующего пути и неповторяющегося имени | Добавление нового файла. Обновление списка файлов | Пройден |
| Добавление файла в хранилище. Несуществующий путь | Ввод пользователем несуществующего пути к файлу | Сообщение об ошибке добавления нового файла | Пройден |
| Добавление файла в хранилище. Повтор имени | Ввод пользователем имени для файла, которое уже есть в хранилище | Сообщение об ошибке добавления нового файла | Пройден |

Таблица 5.1 − Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| Удаление файла из хранилища | Выбор файла. Выбор в контекстном меню опции «Delete» | Удаление файла из хранилища. Обновление списка файлов хранилища | Пройден |
| Экспорт файла из хранилища | Выбор файла. Выбор в контекстном меню опции «Export» | Экспорт файла из хранилища в папку Export | Пройден |
| Изменение файла | Попытка пользователем изменить содержимое файла из проводника | Создание копии файла, сохранение прежней версии | Пройден |
| Удаление хранилища | Удаление пользователем хранилища из проводника | Обновление списка хранилищ и паролей | Пройден |

# **6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

## **6.1 Интерфейс программного средства**

**6.1.1** Главное окно

Главное окно программы отображает только список уже созданных хранилищ, каждый элемент которого состоит из имени хранилища и пути к хранилищу, и кнопку для создания нового хранилища. При добавлении большого числа хранилищ у окна появится скроллбар. Вид главного окна представлен на рисунке 6.1.

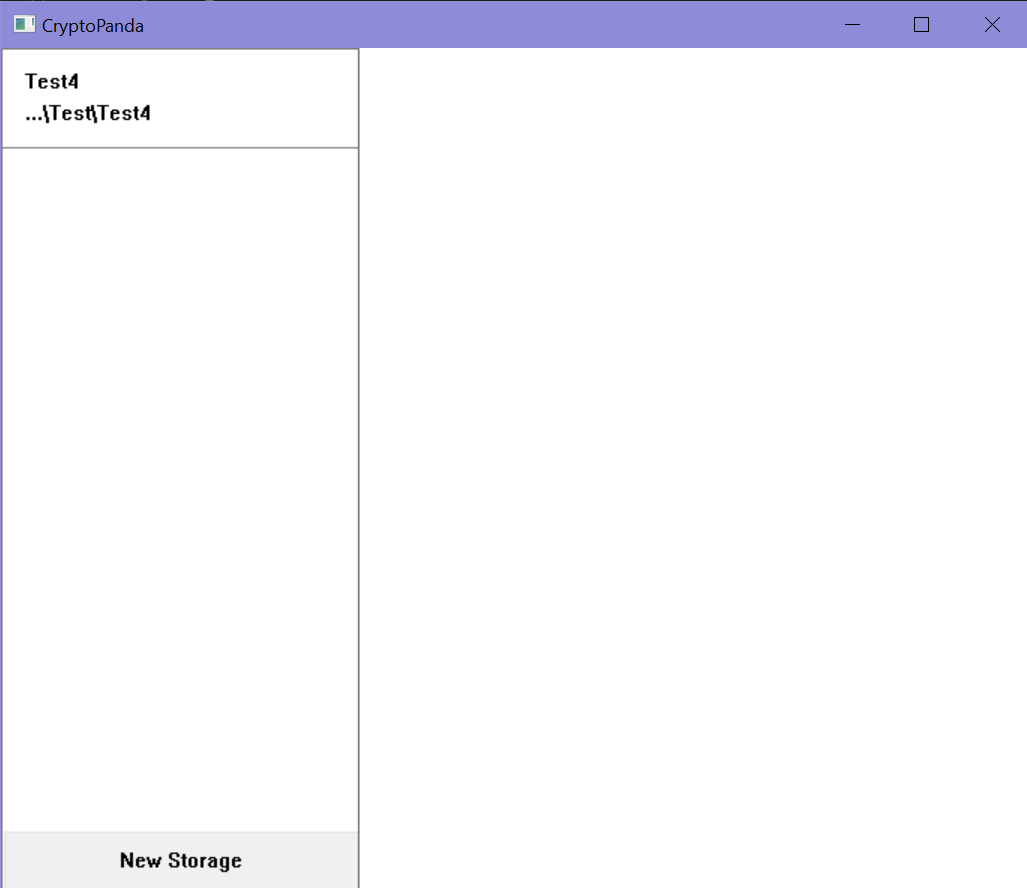


Рисунок 6.1 − Главное окно

Для добавления нового хранилища необходимо нажать на кнопку «New Storage», после чего откроется окно с полями для ввода.

**6.1.2** Окно создания нового хранилища

Для создания нового хранилища необходимо ввести следующие данные:

1 Путь к папке, в которой необходимо создать хранилище;

2 Имя хранилища;

3 Пароль, который будет использоваться для шифрования файлов. Пароль должен быть от 8 до 32 символов и не содержать запрещённых символов;

4 Повторить пароль, введённый в поле выше.

Пример заполнения данных представлен на рисунке 6.2.

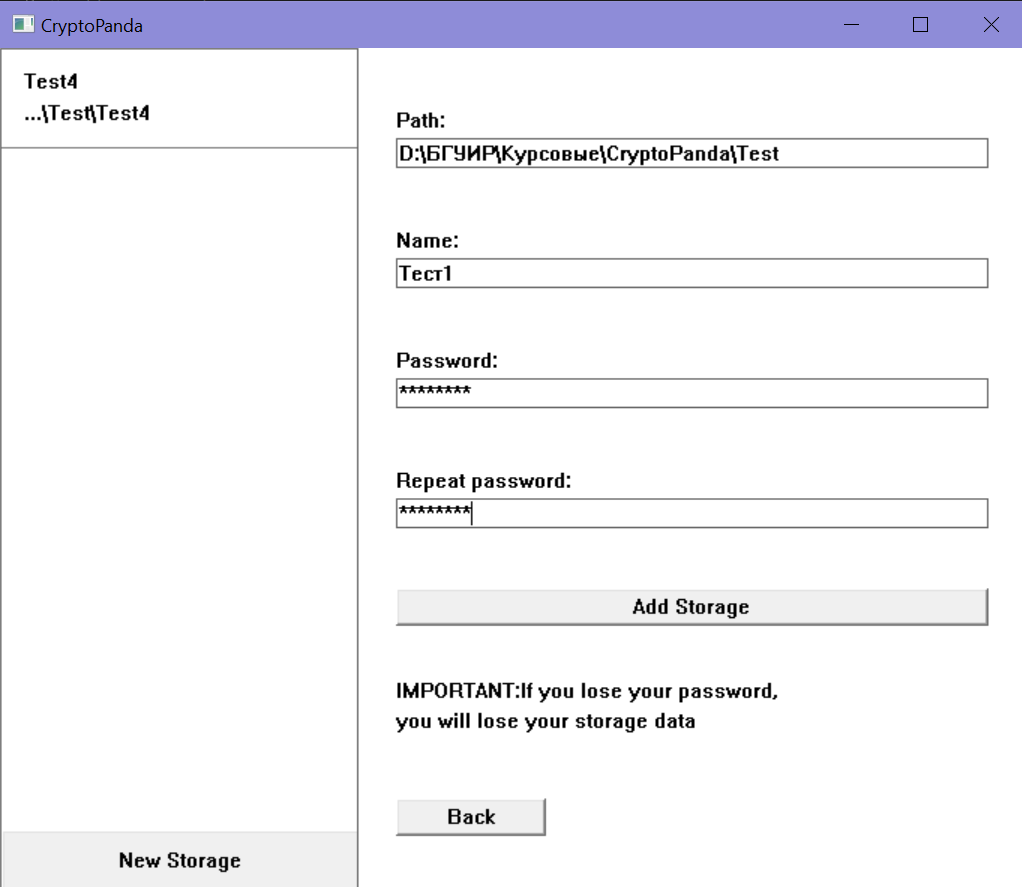


Рисунок 6.2 − Пример заполнения данных

Если данные введены верно, то после нажатия на кнопку «New Storage» в списке доступных хранилищ появится только что созданное (рис. 6.3). При нажатии на кнопку «Back» хранилище создано не будет.

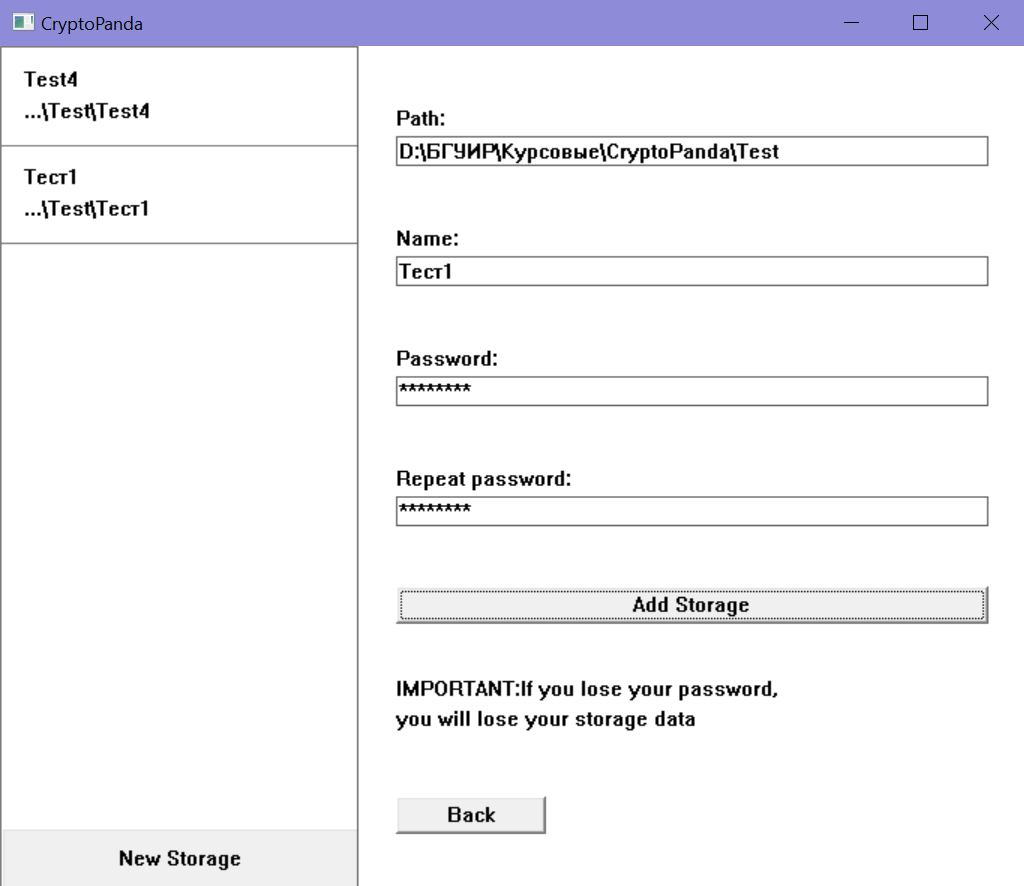


Рисунок 6.3 − Успешное создание хранилища

**6.1.3** Открытие хранилища

Для открытия хранилища необходимо выбрать одно из доступных и нажать на него. Появится окно для ввода пароля. Если пароль введён верно, то откроется окно управления хранилищем. Если пароль неверный, появится сообщение об ошибке. Пример ввода отображён на рисунке 6.4.

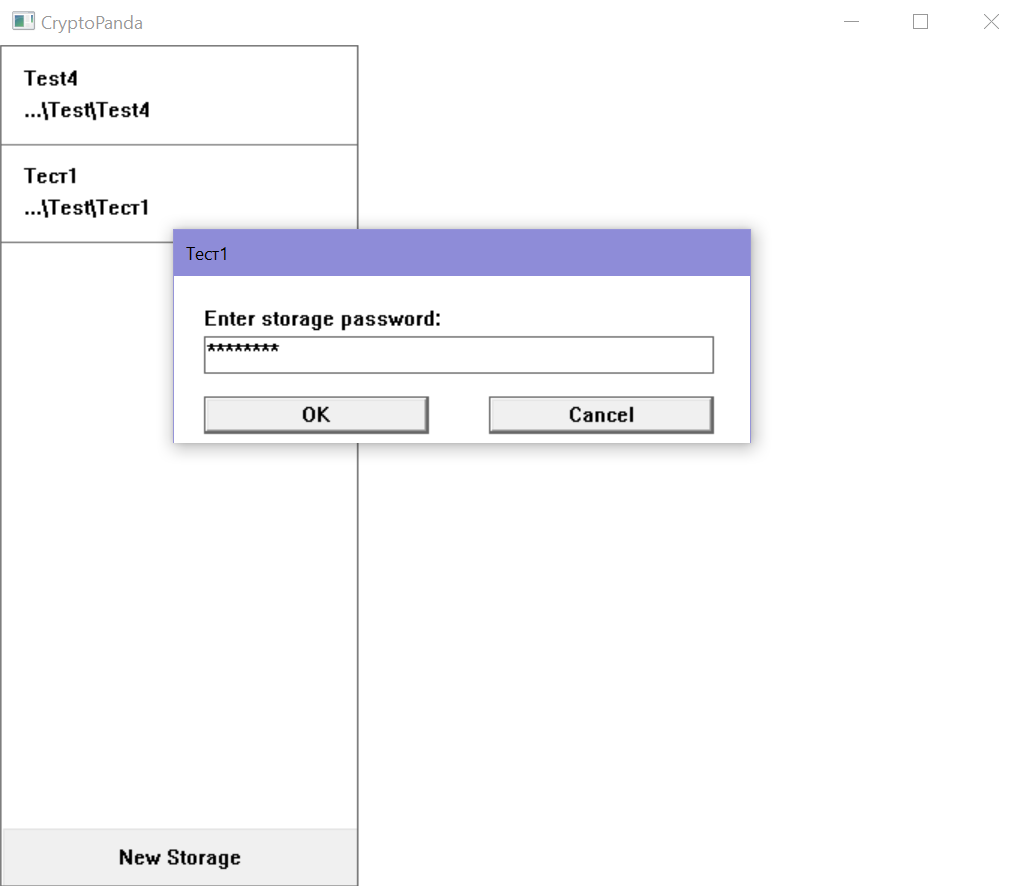


Рисунок 6.4 − Пример ввода пароля

## **6.2 Управление хранилищем**

При открытии хранилища появится окно, в котором пользователь сможет совершить следующие действия:

1 Заблокировать хранилище, нажав на кнопку «Lock»;

2 Добавить файл в хранилище, нажав на кнопку «Add File»;

3 Экспортировать файл в исходном виде;

4 Удалить файл.

**6.2.2** Добавление файла

Для добавления файла в хранилище необходимо ввести путь к исходному файлу и имя, под которым файл нужно сохранить, в окно, которое появится после нажатия на кнопку «Add File». Пример заполнения представлен на рисунке 6.5.

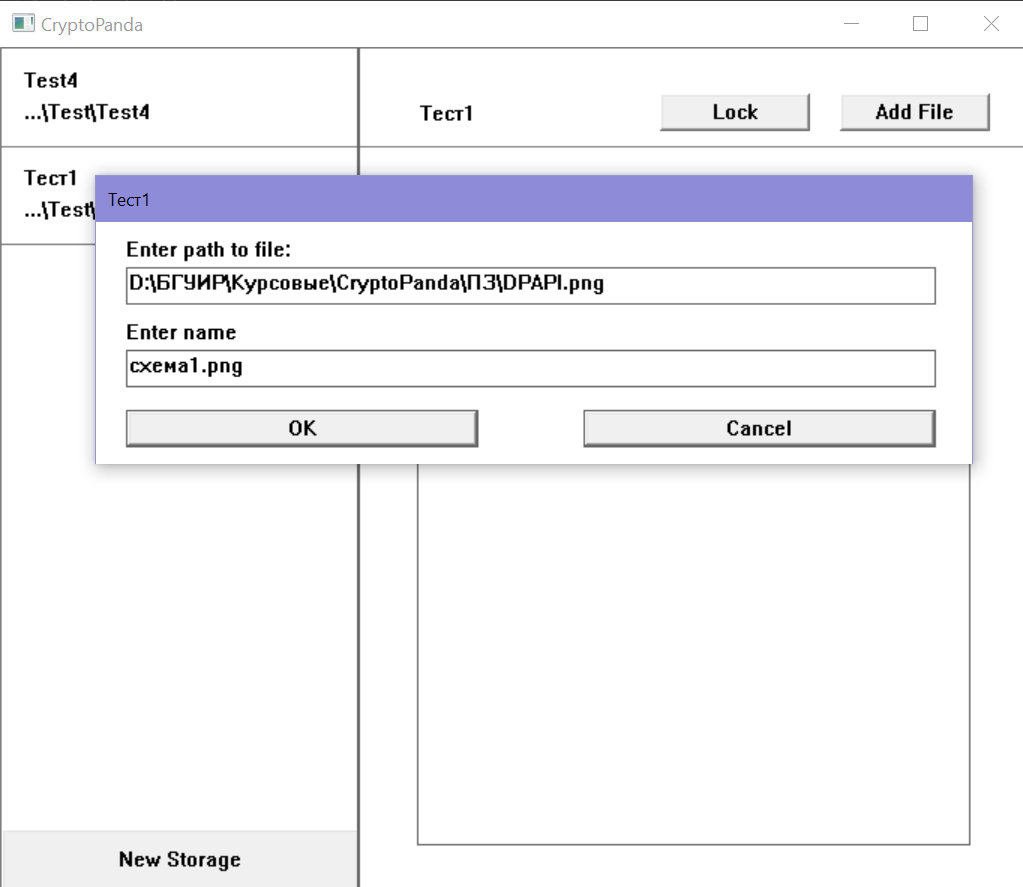


Рисунок 6.5 − Пример заполнения полей для добавления файла

При удачном добавлении имя файла появится в списке, а в хранилище появится файл с зашифрованной информацией. Результат добавления файла представлен на рисунке 6.6.

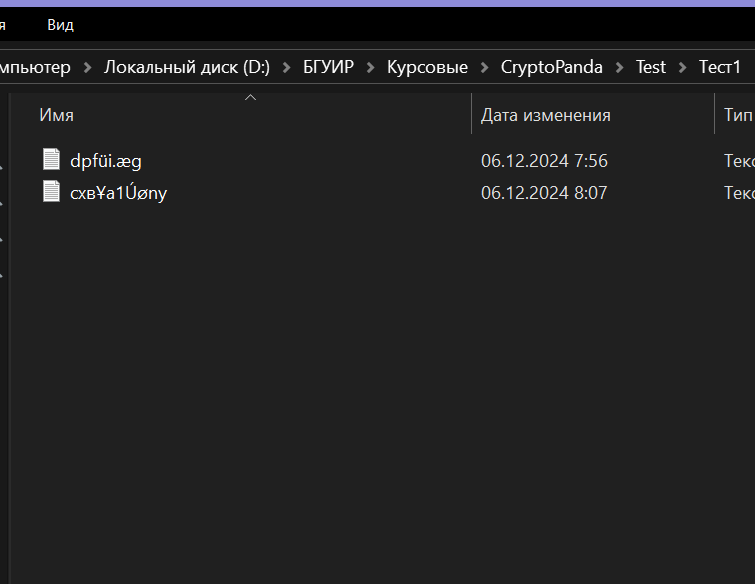
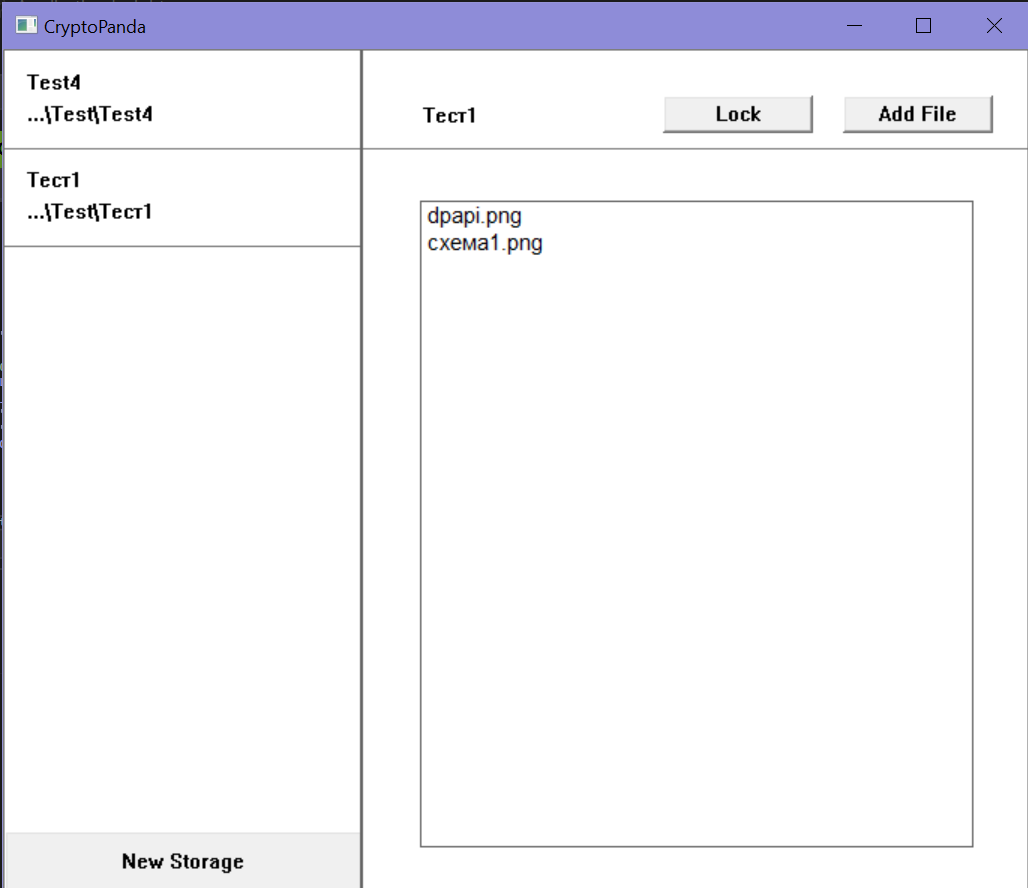


Рисунок 6.6 − Результат добавления файла в хранилище

**6.2.3** Управление файлами

Приложение предоставляет ещё 2 функции для работы с файлами: экспорт и удаление. Обе эти функции активируются при выборе их в контекстном меню выбранного файла. При выборе экспорта файл появится в расшифрованном виде в папке Export. Пример выполнения на рисунке 6.7. При удалении изменится содержимое хранилища, а так же списка файлов хранилища в приложении. Пример выполнения удаления на рисунке 6.8

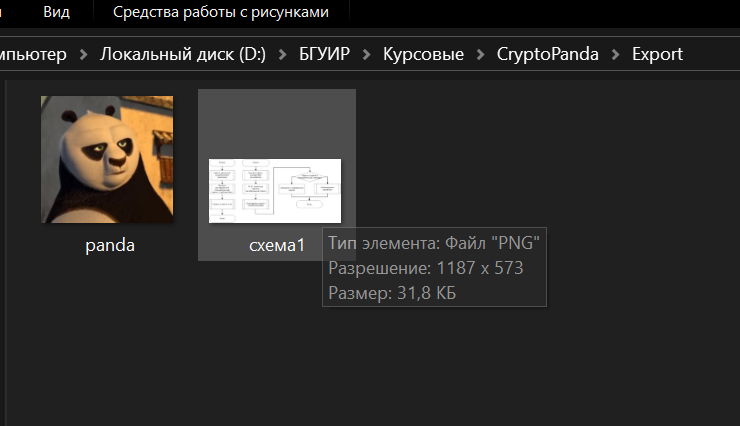
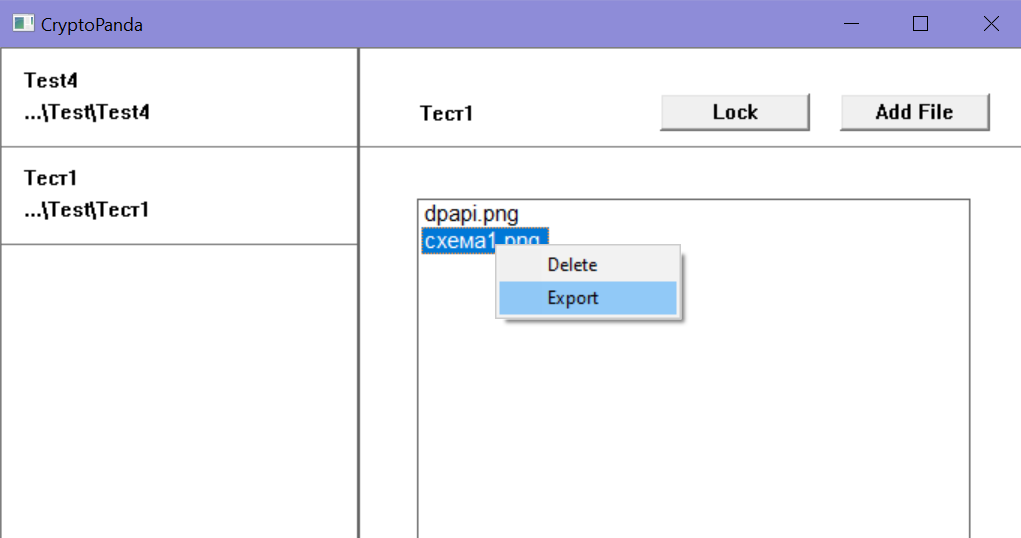


Рисунок 6.7 − Пример выполнения экспорта файла

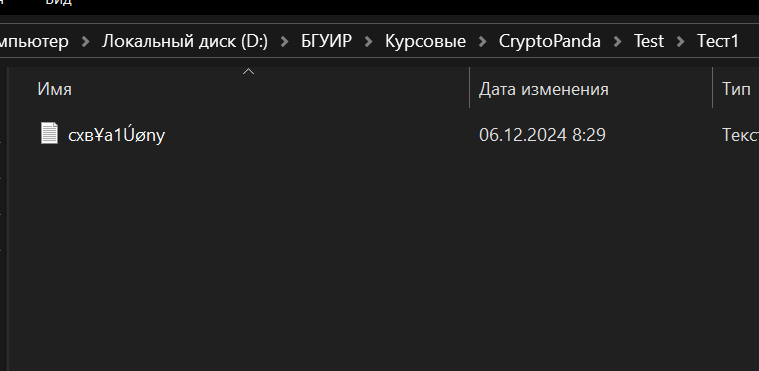
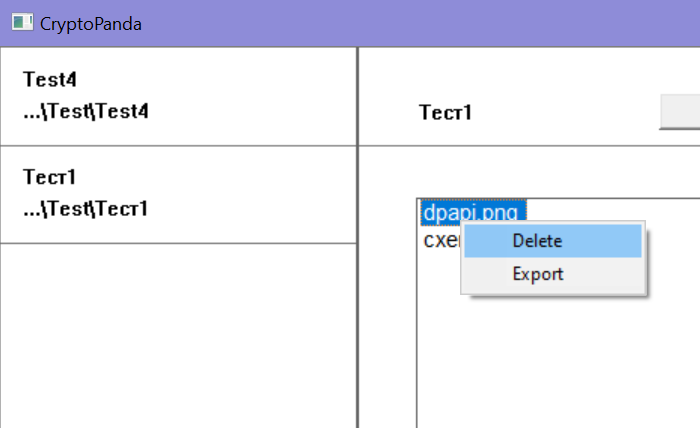


Рисунок 6.8 − Пример выполнения удаления файла