Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Факультета компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Прозрачное шифрование данных (уровень доступа к диску)

Выполнил: студент группы 853505

Гаврик А.В.

Руководитель: ассистент кафедры информатики

Протько М.И.

Минск 2021

Содержание

Введение………………………………………………………………………...…....3

1. Прозрачное шифрование..…...…………………………………….......................4

* 1. Шифрование разделов…………………………………..…….……….....5

1.2 Шифрование на уровне файлов……………………….………….…..….5

2. AES(Стандарт шифрования)………...…………………………………….…......5

2.1 Описание………………………………….……………..……...................5

2.2 Шифрование………………………………………………………............6

2.3 Расшифровка….………………………………….….……………...……..9

3. Графический интерфейс……………………...……………………………........11

4. Методика работы с полученной программой………………………...……......12

4.1 Обработка ошибок………………………………………………….……18

4.2 Предупреждение пользователя…………………………………………19

5. Заключение……………………………………………………………………….21

6. Список использованных источников…………………………………………...22

**Введение**

В настоящее время одним из решений задачи обеспечения конфиденциальности информации, хранящейся в файловой системе, является шифрование. Существуют различные методы и способы шифрования. Например, пользователь может зашифровать свои файлы и хранить их в защищенном виде. При необходимости использования этих файлов пользователь должен расшифровать их, провести с ними нужные ему операции и снова их зашифровать. Неудобство данного способа шифрования состоит в том, что при каждом обращении к файлам пользователь сам должен расшифровывать, а затем зашифровывать их. Гораздо удобнее для него, если бы система сама делала данные операции.

**1. Прозрачное шифрование.**

Прозрачное шифрование, известное также как шифрование в режиме реального времени, является методом шифрования, при котором данные зашифровываются и расшифровываются без участия пользователя с помощью драйвера, работающего в фоновом режиме и следящего за всеми обращениями к данным. Основной целью этого метода шифрования является защита от атак, направленных на получение данных в обход операционной системы, т. е. путем загрузки через другую ОС или использования средства прямого доступа к жесткому диску. Среди преимуществ прозрачного шифрования, помимо преимуществ, присущих всем остальным видам шифрования файлов, можно выделить следующие:

а) простота использования – пользователю достаточно один раз отметить для файла использование шифрования, и при всех дальнейших обращениях к файлу со стороны пользователя он будет расшифровываться автоматически;

б) данные расшифровываются в оперативной памяти, и не требуется сохранения расшифрованных данных на диск.

Как правило, в средствах прозрачного шифрования используется технология цифрового конверта: данные шифруются симметричным алгоритмом, а соответствующий им симметричный ключ – асимметричным, с помощью открытого ключа пользователя. Принцип работы средства прозрачного шифрования выглядит следующим образом. Пусть пользователь работает с некоторым приложением и через это приложение пытается открыть какой-нибудь файл. Тогда приложение отправляет запрос файловой системе на открытие файла. Файловая система по атрибутам файла определяет, зашифрован он или нет. Если зашифрован, то файловая система через службу операционной системы отправляет запрос на расшифровку симметричного ключа агенту, который может взаимодействовать с пользователем, например, он может запросить токен с закрытым ключом пользователя или ввод пароля. Этот агент расшифровывает симметричный ключ с помощью соответствующего закрытого ключа пользователя и через службу операционной системы.

Существует два вида прозрачного шифрования: шифрование разделов (самый простой пример – шифрование всего диска) и шифрование на уровне файлов, которое, как правило, обеспечивается за счет надстройки над файловой системой, поддерживаемой операционной системой.

**1.1 Шифрование разделов**

1. Простота реализации
2. Шифрование всех файлов внутри раздела, независимо от их уровня конфиденциальности
3. Не предназначено для шифрования по сети
4. Полностью шифрует раздел с именами файлов, папок, метаданными
5. Возможность скрытия факта, что раздел находится в зашифрованном виде
6. Работает на любой файловой системе

**1.2 Шифрование на уровне файлов**

1. Высокий уровень сложности: необходимо изучение поддерживаемой файловой системы

2. Избирательный метод шифрования – только выбранные файлы

3. Может использоваться для шифрования файлов в сети на стороне сервера

4. Шифрует исключительно данные внутри файла

5. Наличие зашифрованных данных очевидно для наблюдателя

6. Работает на определенном наборе файловых систем

**2. AES(Стандарт шифрования)**

**AES** — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES.

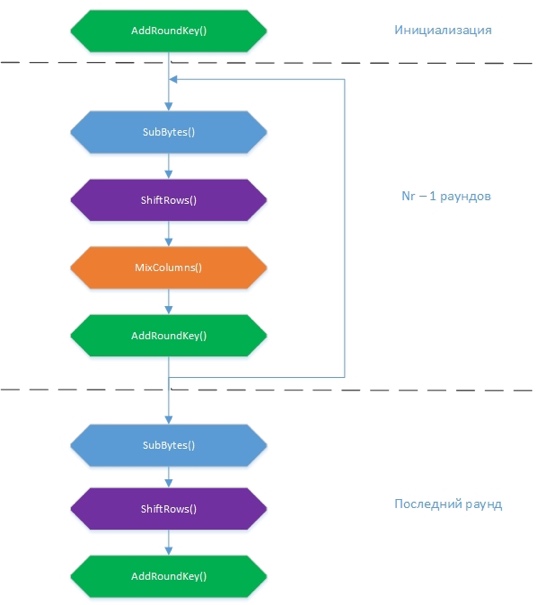
**2.1 Описание**

Advanced Encryption Standard является общеизвестным названием алгоритма Rijndael, который был разработан двумя бельгийскими криптографами Йоаном Дайменом и Винсентом Рэйменом. Алгоритм является блочным и симметричным. Принят в качестве стандарта шифрования данных для гос учреждений в США. Нашумевшее в последнее время Агенство Национальной Безопасности использует его для хранения документов: вплоть до уровня “секретно” применяется шифрование с ключом длиной в 128 бит, информация “совершенно секретно” требует ключа в 192 или 256 бит. В дополнение к высокой криптостойкости алгоритм базируется на не самой сложной математике.

**2.2 Шифрование**

Для работы нам необходим набор байтов в качестве объекта шифрования и секретный ключ, который потребуется при расшифровке. Будем использовать ключ в 128 бит.

Алгоритм имеет четыре трансформации, каждая из которых своим образом влияет на промежуточный результат шифрования и в конечном итоге приводит к результату: SubBytes(), ShiftRows(), MixColumns() и AddRoundKey().



*Рисунок 1. Общая схема шифрования.*

В начале заполняется массив входными значениями За раз шифруется блок размером 16 байт.



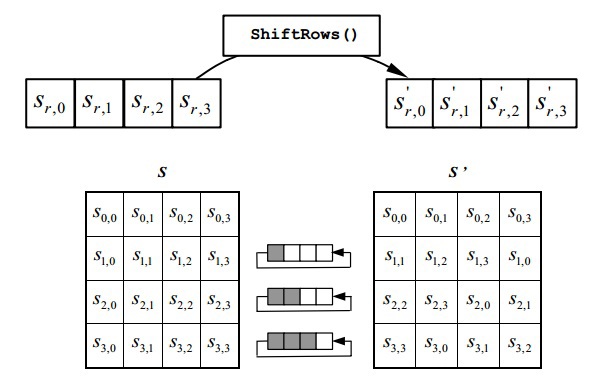
*Рисунок 2. Заполнение массива.*

##### **SybButes()**

Преобразование представляет собой замену каждого байта из массива на соответствующий ему из константной таблицы Sbox.

##### **ShiftRows()**

Простая трансформация. Она выполняет циклический сдвиг влево на 1 элемент для первой строки, на 2 для второй и на 3 для третьей. Нулевая строка не сдвигается.



*Рисунок 3. Трансформаиця ShiftRows().*

##### **MixColumns()**

В рамках этой трансформации каждая колонка в массиве представляется в виде многочлена и перемножается в поле GF(28) по модулю x4 + 1 с фиксированным многочленом 3x3 + x2 + x + 2.



*Рисунок 4. Работа MixColumns().*

##### **AddRoundKey()**

Трансформация производит побитовый XOR каждого элемента из массива с соответствующим элементом из RoundKey.

##### **KeyExpansion()**

Эта вспомогательная трансформация формирует набор раундовых ключей — KeySchedule. KeySchedule представляет собой длинную таблицу, состоящую из Nb\*(Nr + 1) столбцов или (Nr + 1) блоков, каждый из которых равен по размеру исходному массиву. Первый раундовый ключ заполняется на основе секретного ключа, который вы придумаете, по формуле  
KeySchedule[r][c] = SecretKey[r + 4c], r = 0,1...4; c = 0,1..Nk.  
  
 В KeySchedule мы должны заносить байты, чтобы были возможны дальнейшие операции. Используя этот алгоритм для шифрования, придется хранить в голове последовательность чисел, так что в нашей реализации KeyExpansion() будет на вход брать plaintext строку и применяя ord() для каждого из символов, записывать результат в ячейки KeySchedule. Отсюда вытекают ограничения: не более 16 символов длиной и, так как мы работаем с байтами, ord() символа не должен возвращать значения большие чем 255 или 11111111 в двоичной, иначе получим на выходе неверное шифрование. Получается, что с помощью ключа на русском языке зашифровать не получится.



*Рисунок 5. макет KeySchedule для AES-128.*

**2.3 Расшифровка**

Если с тем же ключевым словом выполнить последовательность трансформаций, инверсных трансформациям шифрования, то получится исходное сообщение. Такими инверсными трансформациями являются *InvSubBytes(), InvShiftRows(), InvMixColumns()* и *AddRoundKey()*.



*Рисунок 6. Общая схема алгоритма расшифровки.*

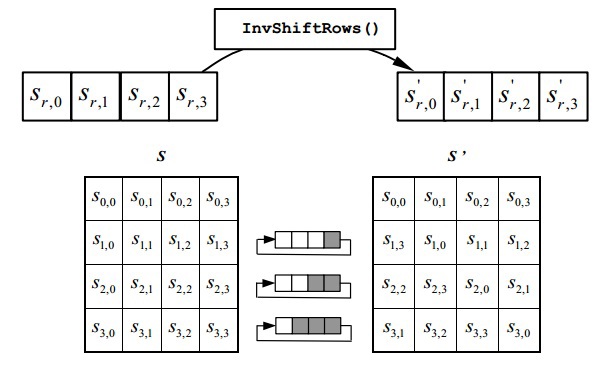
Стоит отметить, что последовательность добавления раундовых ключей в AddRoundKey() должна быть обратной: от Nr + 1 до 0. Изначально, как и при шифровании, из массива входных байтов формируется таблица . Затем над ней в каждом раунде производятся преобразования, в конце которых должно получиться расшифрованный файл.

##### **InvSubBytes()**

Работает точно так же, как и SubBytes(), за исключением того, что замены делаются из константной таблицы InvSbox.  
  
Оставшиеся обратные трансформации тоже будут очень похожи на свои прямые аналоги, поэтому в коде не выделяем под них отдельных функций. Каждая функция, описывающая трансформацию, будет иметь входную переменную inv. Если она равна False, то функция будет работать в обычном или прямом режиме(шифрование), если True — в инверсном(дешифровка).

##### **InvShiftRows()**

Трансформация производит циклический сдвиг вправо на 1 элемент для первой строки массива, на 2 для второй и на 3 для третьей. Нулевая строка не поворачивается.



*Рисунок 7. Работа invShiftRows().*

##### **InvMixColumns()**

Операции те же, но каждая колонка массива перемножается с другим многочленом {0b}x3 + {0d}x2 + {09}x + {0e}. В матричной форме это выглядит так:



*Рисунок 8. Работа invMixColumns().*

##### **AddRoundKey()**

Никаких изменений не вносим. Эта трансформация обратна сама себе в силу свойства операции XOR.

# **3. Графический интерфейс**

Tkinter — это графическая библиотека, позволяющая создавать программы с оконным интерфейсом. Библиотека является интерфейсом к популярному языку программирования и инструменту создания графических приложений tcl/tk. Tkinter, как и tcl/tk, является кроссплатформенной библиотекой и может быть использована в большинстве распространённых операционных систем (Windows, Linux, Mac OS X и др.).

Tk является базовым классом любого Tkinter приложения. При создании объекта этого класса запускается интерпретатор tcl/tk и создаётся базовое окно приложения.

Tkinter является событийно-ориентированной библиотекой. В приложениях такого типа имеется главный цикл обработки событий. В Tkinter такой цикл запускается методом mainloop. Для явного выхода из интерпретатора и завершения цикла обработки событий используется метод quit.

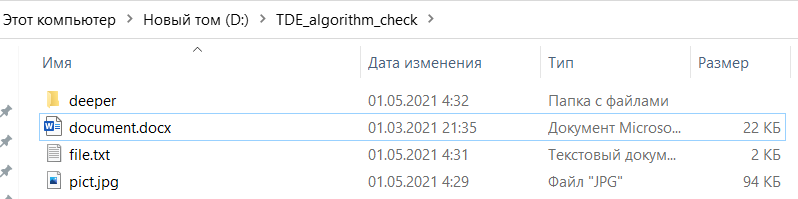
**4. Методика работы с полученной программой**

Для демонстрации работы программы создадим папку, в которой будут хранится файлы, которые мы хотим защитить.



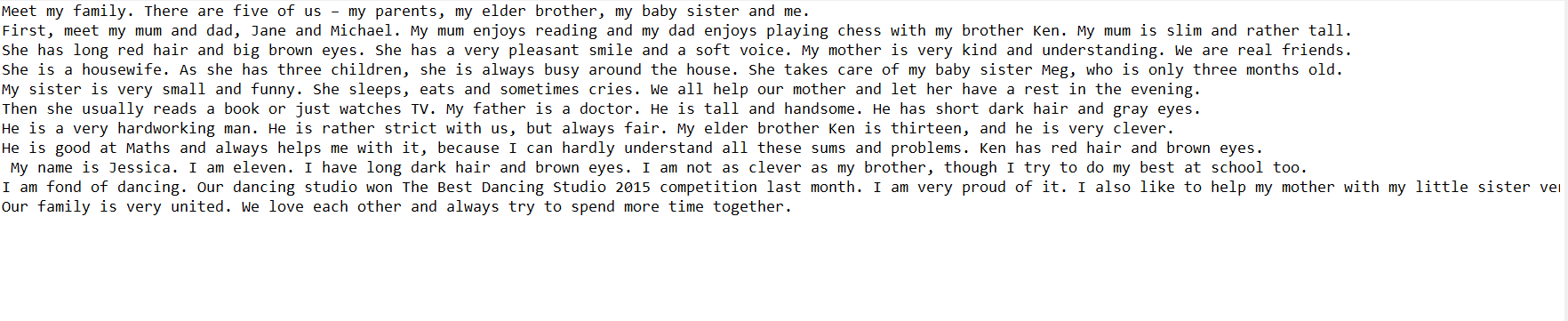
*Рисунок 9. Создание директории.*

Внутри этой папки будем хранить картинку, документ, текстовый файл и папку, в которой будем хранить те же файлы (демонстрация работы приложения с вложенными файлами).

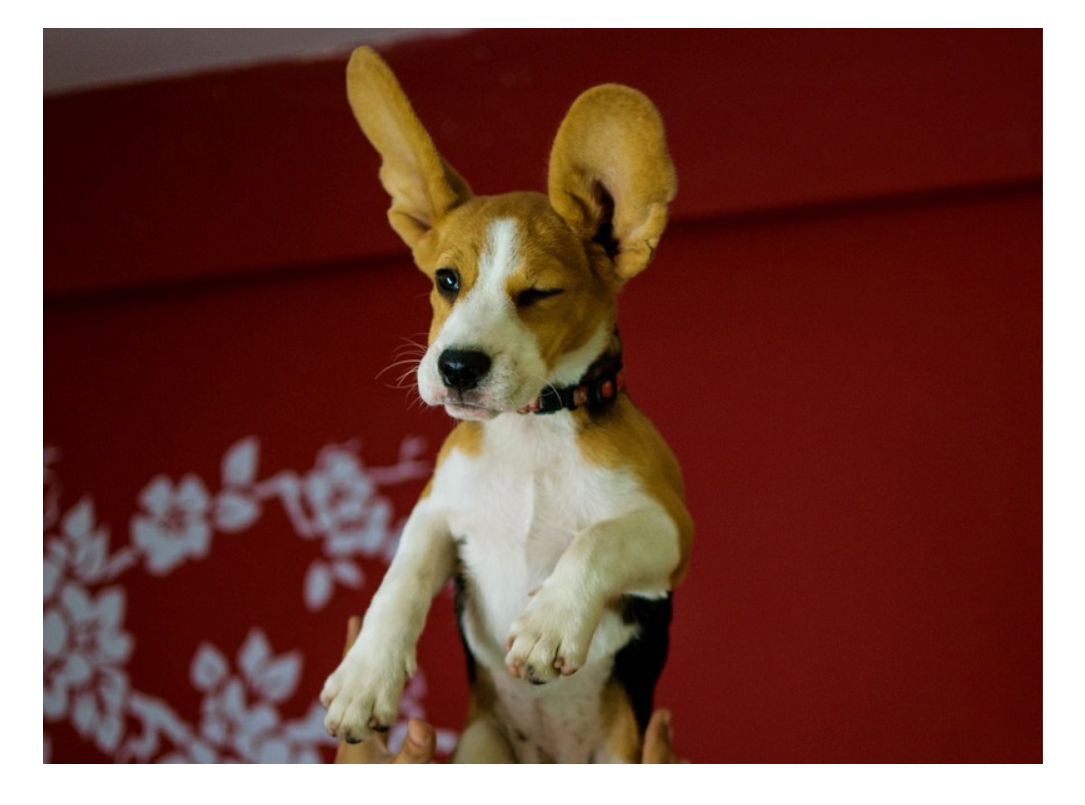


*Рисунок 10. Созданная директория с файлами.*

Далее продемонстрируем информацию, которою хранит каждый файл для отслеживания работы приложения.

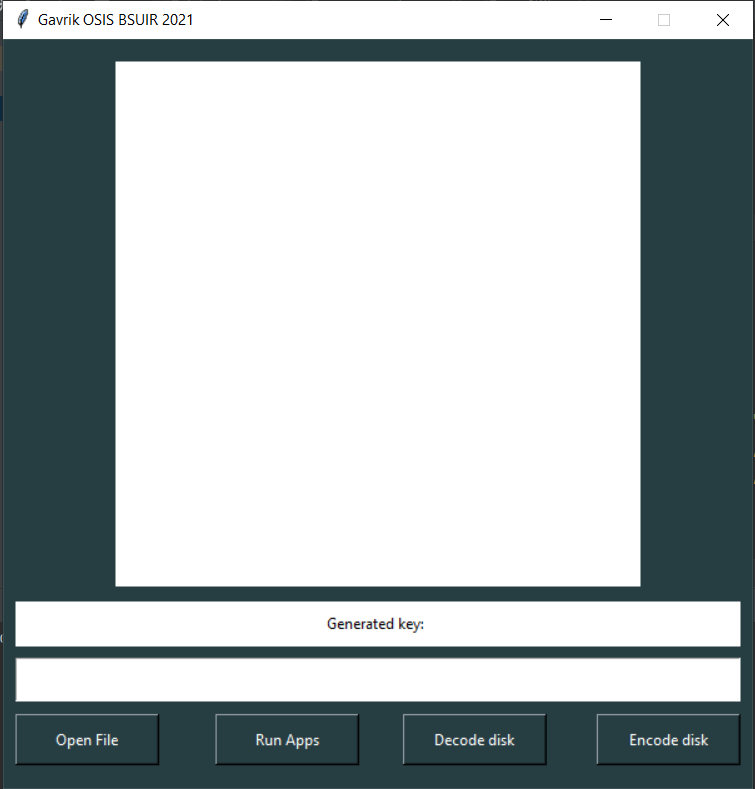


*Рисунок 11. Информация в текстовом файле.*



*Рисунок 12. Информация в картинке.*

Запускаем приложение



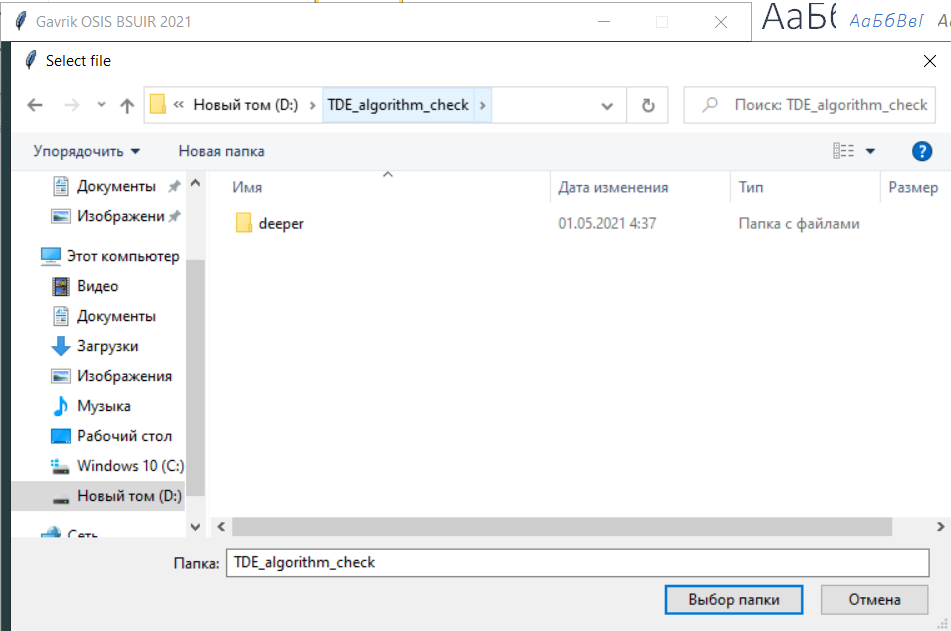
*Рисунок 13. интерфейс приложения.*

Для того, чтобы зашифровать или дешифровать диск нужно выбрать путь и ввести ключ.



*Рисунок 14. Выбор файла.*

Выбираем диск.



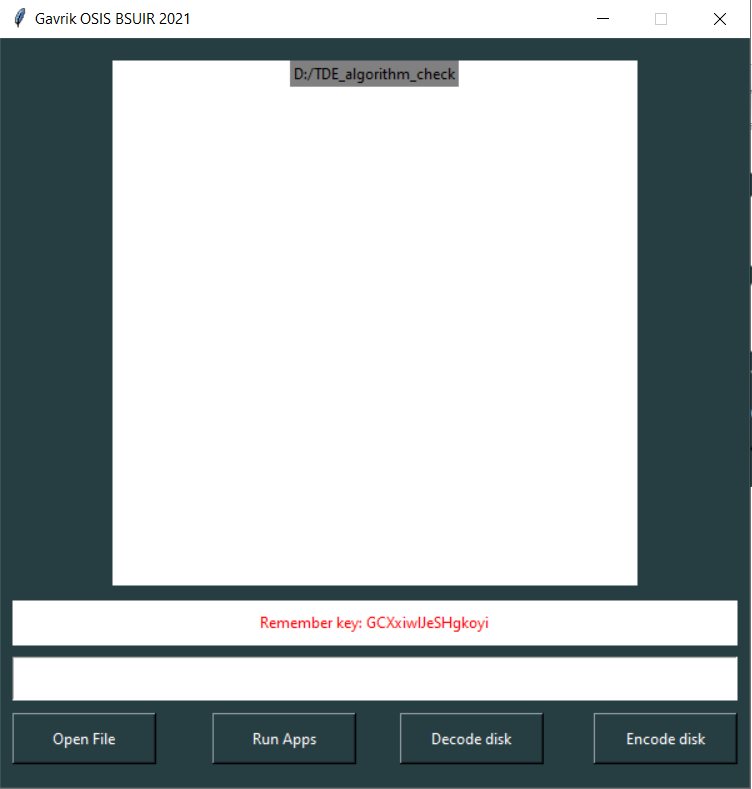
*Рисунок 15. Выбираем диск.*

После выбора файла в интерфейсе приложения мы видим путь к файлу.



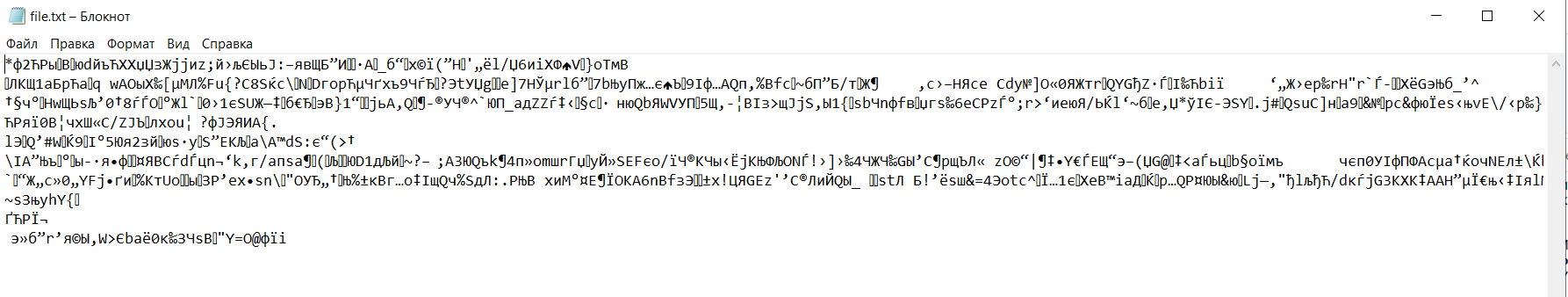
*Рисунок 16. Интерфейс приложения.*

Нажимаем кнопку для шифрования диска и видим ключ, который нужно запомнить для дальнейшего дешифрования.

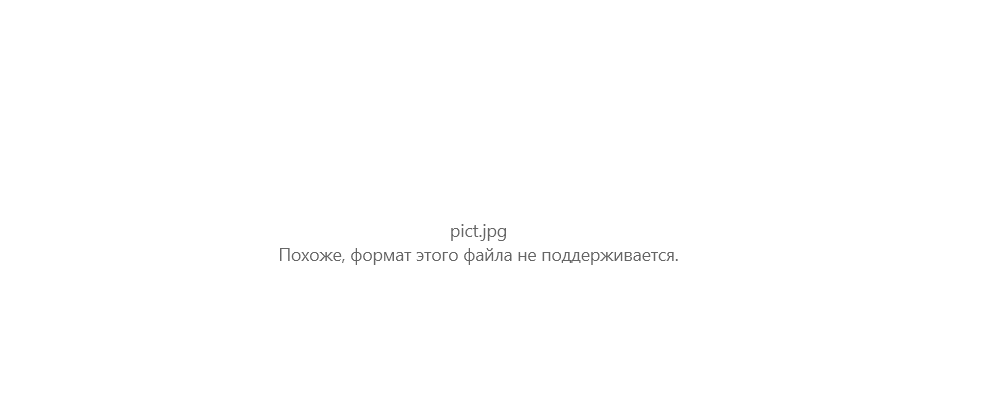


*Рисунок 17. Интерфейс приложения.*

После шифрования диска проверяем файлы.

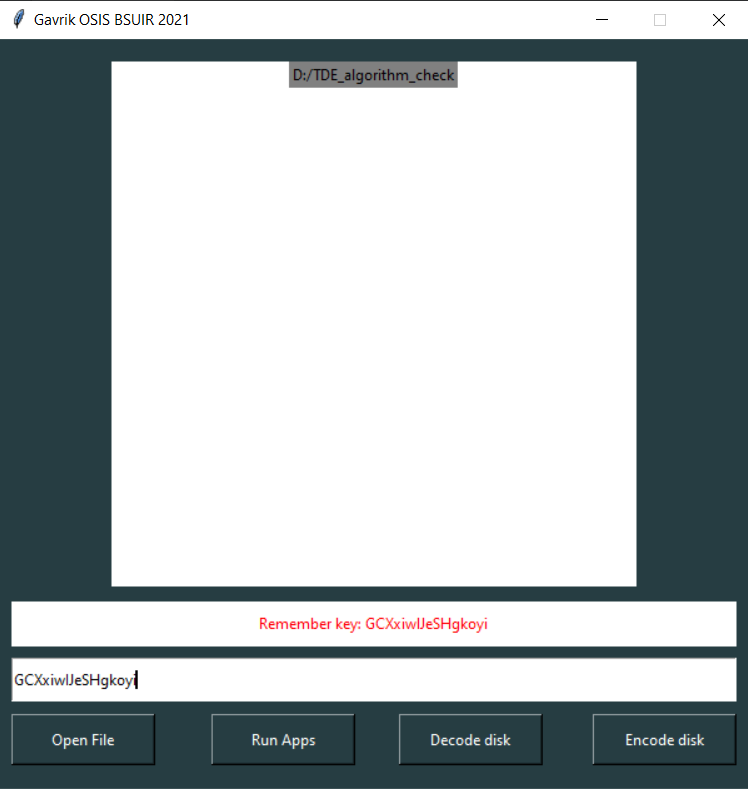


*Рисунок 18. Зашифрованный текстовый файл.*



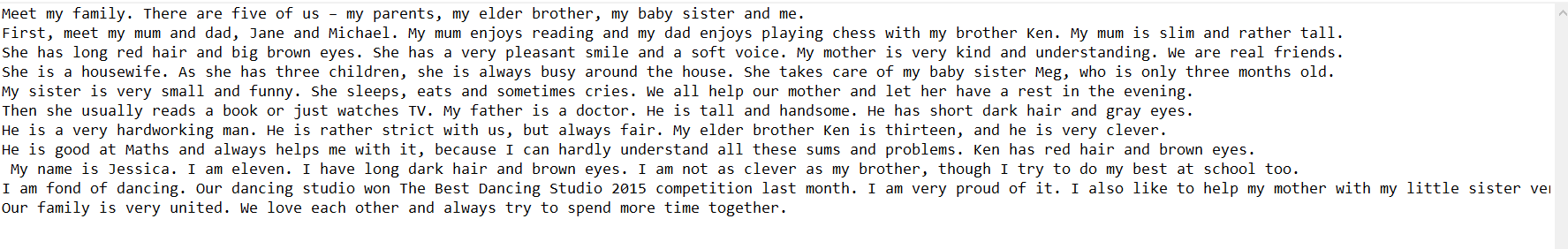
*Рисунок 19. Зашифрованная картинка.*

Теперь проведем операцию дешифрования диска. Вводим ключ по которому будет происходить процесс дешифрации, путь к диску указан.



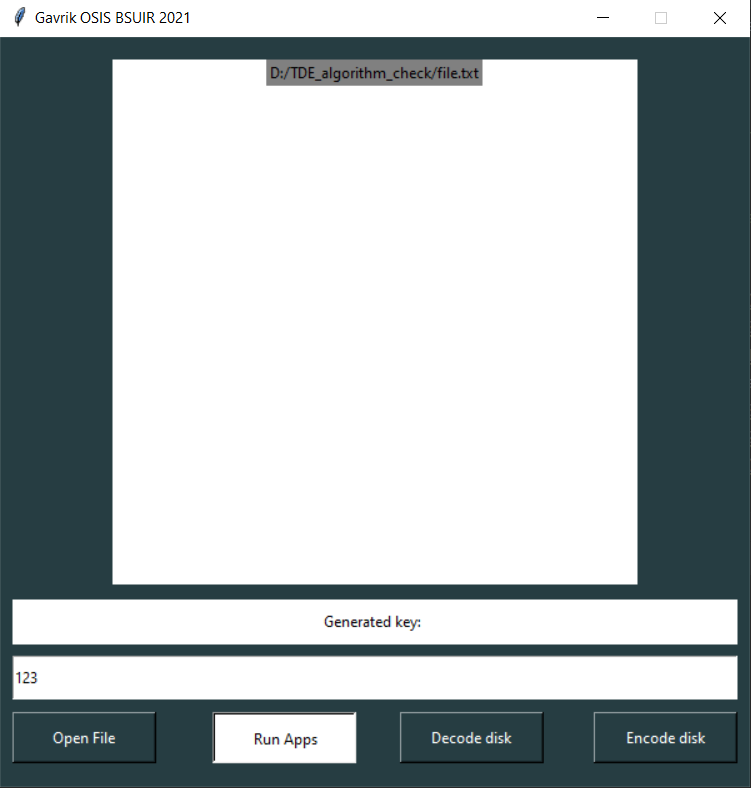
*Рисунок 20. Интерфейс приложения.*

После дешифрации проверяем файлы.



*Рисунок 21. Дешифрованный текстовый файл.*

Последняя функция приложения, запускает зашифрованный файл. Дешифрация файла происходит в оперативной памяти, и после завершения работы пользователем файл зашифровывается.



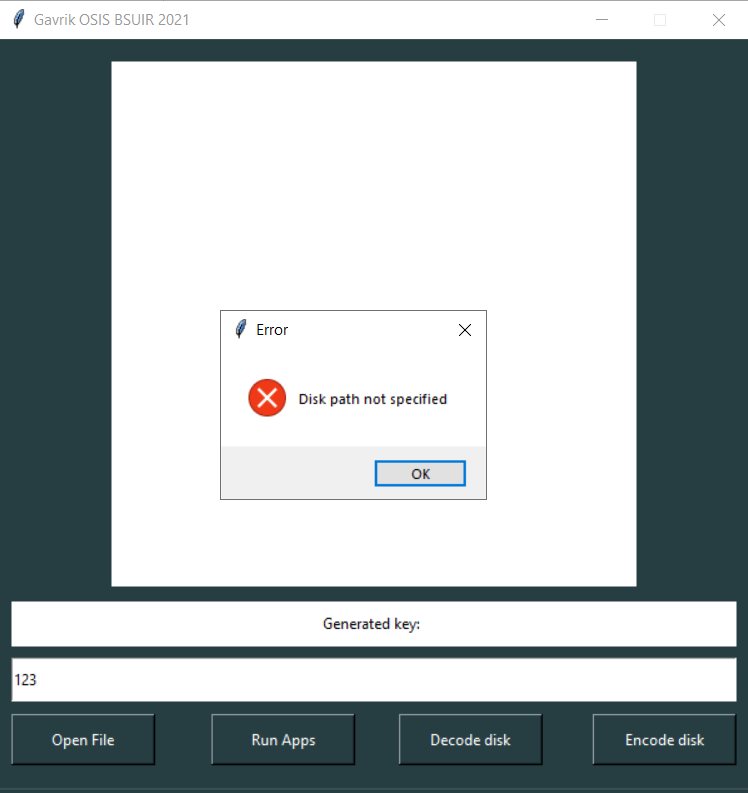
*Рисунок 22. Интерфейс приложения.*



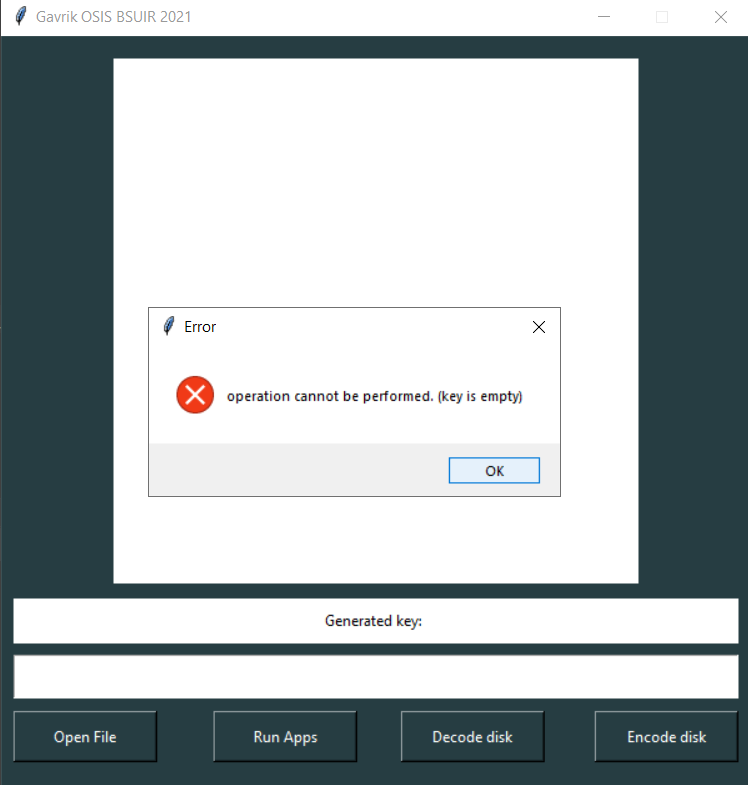
*Рисунок 23. Открытый файл.*

**4.1 Обработка ошибок**

Если пользователь неправильно работает с приложением, то отображаются окна, которые уведомляют о совершении ошибки.

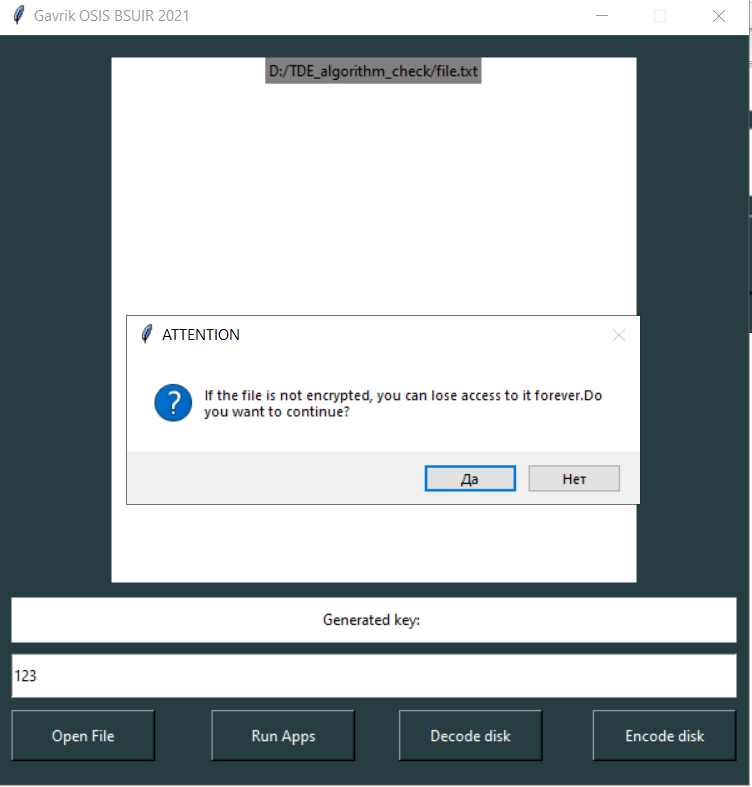


*Рисунок 24. Уведомление об ошибке “не указан путь к диску”.*

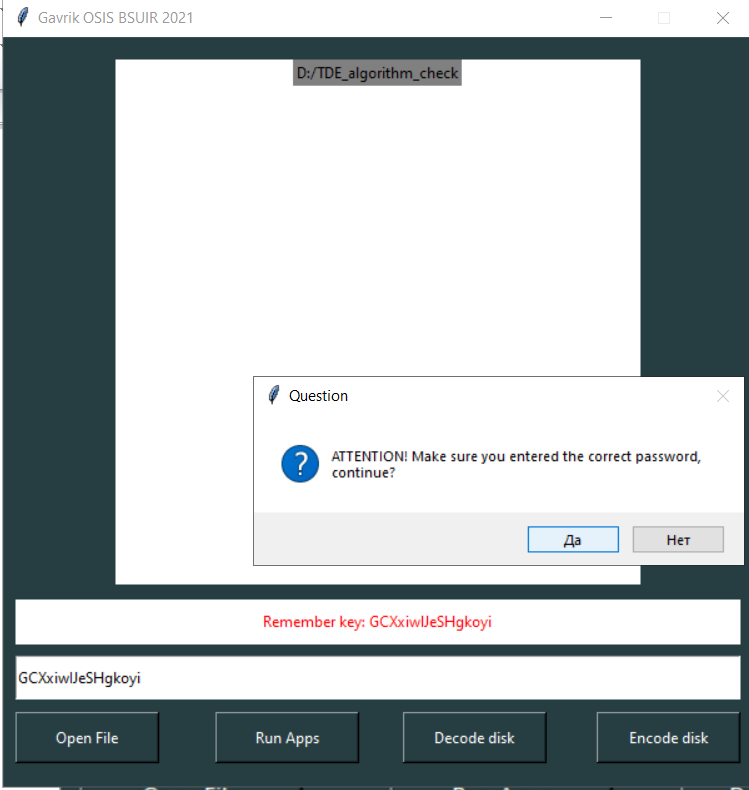


*Рисунок 25. Уведомление об ошибке “не указан ключ”.*

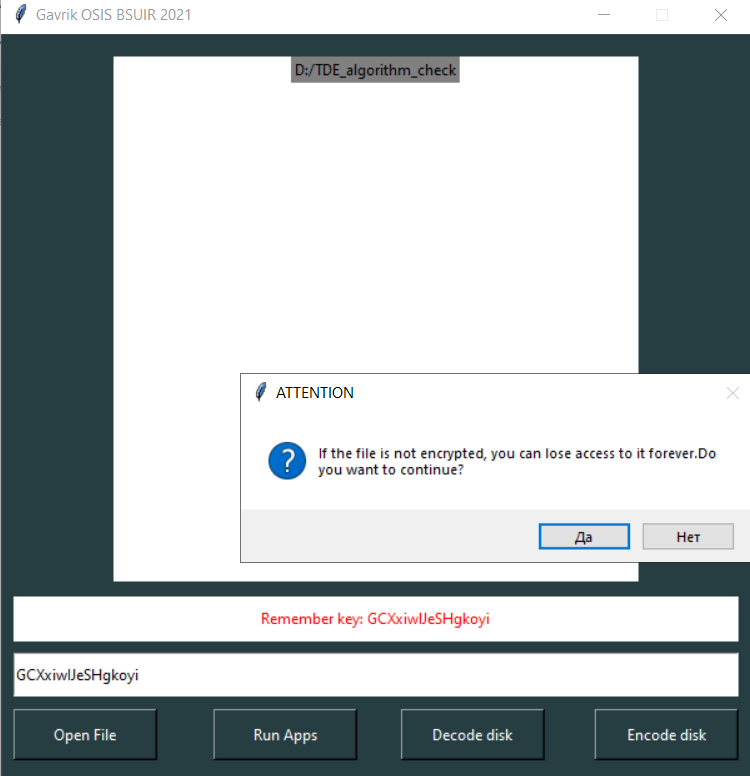
**4.2 Предупреждение пользователя**



*Рисунок 26. Предупреждение пользователя.*



*Рисунок 27. Предупреждение пользователя.*



*Рисунок 28. Предупреждение пользователя.*

**Заключение**

В результате курсовой работы мы разобрались в понятиях:

Прозрачное шифрование, рассмотрели шифрование разделов и шифрование на уровне диска. Разобрались со стандартом шифрования AES. Научились шифровать и расшифровывать файлы с помощью данного алгоритма. Научились использовать Фреймворк Tkinter для создания графического интерфейса приложения.

**Список использованных источников**

1. <https://ru.wikiversity.org/wiki/Курс_по_библиотеке_Tkinter_языка_Python#:~:text=Tkinter%20(от%20англ.%20tk%20interface),инструменту%20создания%20графических%20приложений%20tcl%2Ftk>
2. Шайдеманн Ф. Прозрачное шифрование файлов // Журнал сетевых решений/LAN. 2005. № 15. C. 6.
3. 4. ЖТЯИ. 00050-02 30 01 Средство криптографической защиты информации «КриптоПро CSP». Формуляр. C. 18. 5.
4. ЖТЯИ. 00051-01 30 01 Средство хранения конфиденциальной информации «КриптоПро EFS». Формуляр. C. 17.
5. <https://effbot.org/>
6. <https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf>
7. <https://ru.wikipedia.org/wiki/AES_(стандарт_шифрования)>
8. <https://en.wikipedia.org/wiki/Rijndael_MixColumns>