**УО «Белорусский государственный технологический университет»**

Факультет **информационных технологий**

Кафедра **информационных систем и технологий**

Специальность **1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)»**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОЙ РАБОТЫ**

**по дисциплине «**Защита информации и надежность информационных систем»

**тема «**Реализация и исследование метода кодирования изображений на основе принципа кубика Рубика»

|  |  |  |
| --- | --- | --- |
| **Исполнитель** |  | |
| Студент 3 курса группы1 | подпись, дата | В. В. Пригодич  инициалы и фамилия |
|  |  |  |
| **Руководитель** |  |  |
| Ассистент | подпись, дата | М. Г. Савельева  инициалы и фамилия |

|  |  |
| --- | --- |
| **Курсовая работа защищена с оценкой** | **\_\_\_\_\_\_\_\_\_\_** |
| **Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  (подпись) | М. Г. Савельева  инициалы и фамилия |

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Смелов

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

**ЗАДАНИЕ**

**к курсовому проектированию по дисциплине**«Защита информации и надежность информационных систем»

Специальность: 1-40 05 01 03 «Информационные системы и технологии (издательство-полиграфический комплекс)»

Курс: 3, группа: 1

Студент: Пригодич Вера Валерьевна

**Тема:** Реализация и исследование метода кодирования изображений на основе принципа кубика Рубика

**1. Срок сдачи студентом законченной работы**: «05» мая 2023г.

**2. Исходные данные к проекту:**

**2.1**. Функционально должны быть выполнены следующие задачи:

* Разработка программных функций, реализующих алгоритмы шифрования и дешифрования в соответствии с темой
* Создание приложения с графическими интерфейсом, реализующего разработанные функции
* Сравнение реализованного алгоритма с другими алгоритмами шифрования изображений

**2.2. Требования:**

* Необходимо провести аналитический обзор литературы по теме проекта
* Необходимо описать сферу применимости метода
* Программное средство может быть разработано на любом языке
* Архитектура приложения выбирается разработчиком
* Листинги проекта должны содержать комментарии

**3. Содержание расчетно-пояснительной записки:**

* Введение
* Постановка задачи
* Описание метода
* Описание программного средства
* Тестирование программного средства
* Руководство пользователя
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления выполненного курсового проекта:**

* + Теоретическая часть курсового проекта должна быть представлена в формате MS Word.
  + Оформление записки должно быть согласно правилам.
  + Необходимые схемы, диаграммы и рисунки допускается делать в MS Office Visio или копии экрана (интерфейс).
  + Полные листинги проекта представляются в приложении.
  + К записке необходимо приложить CD (DVD), который должен содержать: пояснительную записку, листинги и файлы базы данных.

#### Календарный план

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Задание курсового проекта. Титульный лист | 10.02.2023 |  |
| 2 | Введение | 24.02.2023 |  |
| 3 | Аналитический обзор литературы по теме проекта | 10.03.2023 |  |
| 4 | Разработка метода | 22.03.2023 |  |
| 5 | Разработка прототипа программного средства | 30.03.2023 |  |
| 6 | Разработка программного средства | 08.04.2023 |  |
| 7 | Тестирование программного средства | 15.04.2023 |  |
| 8 | Написание руководства пользователя | 22.04.2023 |  |
| 9 | Оформление пояснительной записки | 30.04.2023 |  |
| 10 | Сдача проекта | 05.05.2023 |  |

**5. Дата выдачи задания** «10» февраля 2023г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *М. Г. Савельева*

(подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись студента)

**Оглавление**

[Введение 5](#_Toc133522955)

[1 Аналитический обзор литературы 6](#_Toc133522956)

[1.1 Аналоги 6](#_Toc133522957)

[1.1.1 RC4 6](#_Toc133522958)

[1.1.2 Grain 6](#_Toc133522959)

[1.1.3 Методы на основе динамического хаоса 8](#_Toc133522960)

[2 Описание метода 11](#_Toc133522961)

[2.1 Алгоритм кодирования 11](#_Toc133522962)

[2.2 Алгоритм декодирования 12](#_Toc133522963)

[2.3 Исследование метода 13](#_Toc133522964)

[3 Описание программного средства 15](#_Toc133522965)

[3.1 Описание технологии 15](#_Toc133522966)

[3.2 Разработка 15](#_Toc133522967)

[3.3 Тестирование 17](#_Toc133522968)

[3.4 Руководство пользователя 20](#_Toc133522969)

[Заключение 24](#_Toc133522970)

[Список используемых источников 25](#_Toc133522971)

Введение

В свете широкого распространения информационных технологий и Интернета возникает множество проблем, связанны с обеспечением безопасности хранения и передачи данных в виде изображений. Существует множество методов защиты информации и обеспечения ее целостности, в том числе криптографические методы. Однако не все из них могут быть эффективными при защите изображений, которые могут содержать конфиденциальную информацию.

Кодирование изображений является важным аспектом защиты графических данных, так как изображения могут содержать конфиденциальную информацию, которую необходимо сохранить в безопасности. В данной работе рассматривается метод кодирования изображений на основе принципа кубика Рубика.

Принцип кодирования изображений на основе кубика Рубика использует свойства кубика Рубика, такие как симметрия и повторяемость, для защиты конфиденциальных данных в виде изображений.

Кубик Рубика является трехмерным пазлом, который состоит из нескольких слоев, каждый из которых можно вращать вокруг одной из трех осей. При этом повороты могут быть выполнены только на углы 90 градусов.

Использование свойств кубика Рубика позволяет перемешать значения пикселей изображения для защиты конфиденциальных данных в виде изображений. При кодировании каждый блок изображения становится нечитаемым, что обеспечивает его защиту от несанкционированного доступа.

Цель данной работы заключается в реализации метода кодирования изображений на основе принципа кубика Рубика и исследовании его эффективности и надежности. Для достижения поставленной цели были выделены следующие задачи:

* реализовать программное средство кодирования изображений на основе принципа кубика Рубика;
* оценить криптостойкость метод и сравнить с другими методами кодирования;
* исследовать производительность разработанного метода кодирования изображений и сравнить с другими методами кодирования;
* составить руководство пользователя.

1. Аналитический обзор литературы
   1. Аналоги

Поскольку цифровое изображение состоит из битов, как и все другие цифровые данные, криптографический методы, используемые для защиты других форм информации, могут быть использованы для изображений.

Это означает, что для шифрования изображения может быть использованы следующие методы: RC4, Grain. Рассмотрим их подробнее.

1.1.1 RC4

RC4 работает на основе ключа переменной длины и оперирует над байтами. Шифрование RC4 осуществляется путем генерации псевдослучайной последовательности, которая затем комбинируется с открытым текстом для создания зашифрованного сообщения.

Ядро алгоритма состоит из функции генерации ключевого потока [1]. Другая часть алгоритма – функция инициализации, которая использует ключ переменной длины *Ki* для создания начального состояния генератора ключевого потока.

В основе алгоритма – размер блока или слова, определяемый параметром *n*. Обычно *n* = 8, но можно использовать и другие значения. Внутренне состояние шифра определяется массивом слов (*S*-блоком) размером 2*n*.

Другими элементами внутреннего состояния являются 2 счетчика (каждый размером в одно слово; обозначим их *i* и *j*) с нулевыми начальными значениями. В основе вычислений лежит операция по mod 2*n*. Генератор ключевого потока RC4 переставляет значения, хранящиеся в *S*, и каждый раз выбирает различное значение из *S* в качестве результата. В одном цикле RC4 определяется одно *n*-битное слово *K* из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста.

Дешифрование производится аналогичным образом путем генерации той же самой псевдослучайной последовательности и ее применения для получения исходного открытого текста.

RC4 считается быстрым и простым в реализации шифром, но у него есть некоторые уязвимости. Например, если использовать один и тот же ключ для шифрования большого количества сообщений, возможна атака на ключ, при которой злоумышленник может восстановить ключ, изучив зашифрованные сообщения. Кроме того, таблица ключей может быть подвергнута атаке, если она не генерируется достаточно случайным образом.

1.1.2 Grain

Grain ­– это потоковый шифр, который был разработан Мартином Шварцем и Юргеном Вихтенбахом в 2008 году.

Алгоритм Grain состоит из двух основных компонентов: генератора ключей и генератора псевдослучайного потока (PRNG) [2]. Генератор ключей используется для получения ключа и начального вектора (*IV*), которые затем используются генератором PRNG для генерации псевдослучайного потока.

Генератор ключей включает в себя несколько элементов: 16-битный регистр, 128-битный ключ и 96-битный *IV*. Регистр состоит из 16 ячеек, каждая из которых хранит один бит. Ключ и *IV* объединяются в одну 224-битную последовательность, которая затем используется для заполнения регистра с помощью операции XOR.

Генератор PRNG состоит из трех частей: линейного обратного связного регистра (LFSR), нелинейного фильтра и выходного шифра (рисунок 1.1).

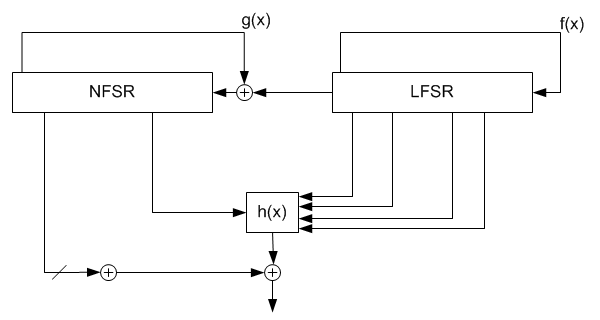


Рисунок 1.1 – Структура PRNG

LFSR состоит из 128 ячеек, каждая из которых хранит один бит. Нелинейный фильтр осуществляет преобразования над содержимым LFSR и используется для улучшения статистических свойств выходного потока. Выходной шифр объединяет данные из LFSR и нелинейного фильтра для генерации псевдослучайного потока.

В процессе генерации псевдослучайного потока LFSR и нелинейный фильтр работают параллельно. LFSR перемещает биты влево на одну позицию и выполняет операцию XOR с определенными ячейками, чтобы определить новый бит. Нелинейный фильтр включает в себя несколько операций, таких как умножение, сложение и битовые сдвиги, которые используются для преобразования содержимого LFSR и улучшения статистических свойств выходного потока.

Выходной шифр объединяет данные из LFSR и нелинейного фильтра и преобразует их с помощью операции XOR. Это преобразование выполняется для каждого байта псевдослучайного потока. Результатом является 128-битный псевдослучайный поток, который используется для шифрования данных.

В целом, Grain является надежным и безопасным потоковым шифром, который может использоваться для шифрования различных типов данных, включая изображения. Однако, как и любой шифр, Grain имеет свои ограничения и недостатки. Например, он не может использоваться для шифрования больших объемов данных, так как его память ограничена 128 битами.

Стоит отметить, что данные алгоритмы разрабатывались в первую очередь для текстовых данных и не учитывают особенностей кодирования изображения. С учетом того, что размер изображений достаточно большой, а некоторым приложениям необходимо работать в режиме реального времени, процесс шифрования должен осуществляться достаточно быстро. Также стоит учесть, что цифровые изображения сравнительно менее чувствительны, чем текстовые данные, так как однократное изменения пикселей не меняет всего изображения. Другими словами, небольшая модификация цифрового изображения приемлема по сравнению с текстовыми данными. Так как данные алгоритмы шифрования разрабатывались без учета этих требований, они не являются наиболее подходящими для данных целей.

1.1.3 Методы на основе динамического хаоса

В последние годы получают развитие новые методов шифрования изображений. Распространёнными являются методы изображений на основе динамического хаоса. Данные методы используют хаотические отображения для генерации ключей шифрования.

Основная идея заключается в том, что непрерывные хаотические отображения проявляют сходны свойства с дискретными криптографическими системами. Популярными являются такие хаотические отображения, как отображение кот Арнольда, отображение Эно, отображение «тент» и так далее. Например, отображение Лоренца, основанное на трех связанных дифференциальных уравнениях, которые описывают движение жидкости внутри замкнутой системы. Значения переменных, используемых в уравнениях, рассматриваются как координаты точки в трехмерном пространстве. Используя начальные значения для этих переменных, можно смоделировать движение точки во времени. Затем, полученные координаты используются для перестановки и замены пикселей изображения.

Отображение кот Арнольда – примечательное отображение из двумерного тора в себя [3]. Владимир Игоревич Арнольд обнаружил эффект диффузии, который он иллюстрировал с помощью отображения, получившего название «кот Арнольд». Арнольд использовал в качестве примера этого отображения картинку в виде головы кота в фазовом пространстве, представляющем собой поверхность тора, на которой одна переменная задает координату по параллели, а другая — по меридиану тора, причем обе определены на интервале от 0 до 1.

Двумерное отображение «кот Арнольда» задаётся системой уравнений (1.1):

 (1.1)

Штрих отмечает значения динамических переменных, относящиеся к следующему шагу дискретного времени. Операция mod 1 обозначает, что берется только дробная часть числа.

Отображение кота Арнольда является детерминированным, то есть его поведение полностью определяется начальными условиями. Начальные условия могут быть представлены позицией и цветом каждого пикселя на изображении. Каждый пиксель перемещается на новую позицию в соответствии с определенным алгоритмом, который растягивает и складывает изображение в различных направлениях. Это приводит к перемешиванию пикселей и изменению их положения на изображении.

Это отображение интересно тем, что при применении его простых итераций к любому изображению, это изображение разваливается на беспорядочное нагромождение точек, но при дальнейших итерациях вновь появляется изображение, очень близкое (хотя и не точное) к исходному (рисунок 1.2).

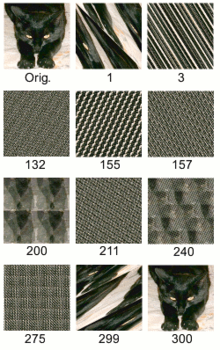


Рисунок 1.2 – Динамика отображения кота Арнольда

В связи с этим, отображение кот Арнольда часто используется для манипулирования пикселями изображений. Оно применяет преобразование к изображению, которое в основном перемешивает пиксели, растягивая и складывая изображение. Когда на изображение применяется оптимальное количество итераций преобразования, получаемое изображение становится непонятным и, следовательно, зашифрованным.

Отображение Эно — один из наиболее изученных примеров дискретных динамических систем, проявляющих хаотическое поведение. Отображение Эно было предложено Мишелем Эно для упрощения отображения Лоренца.

Отображение Эно сопоставляет точке (*xn*, *yn*) на плоскости новую точку по формуле (1.2):

 (1.2)

Система зависит от двух параметров *a* и *b*, в классическом отображении равными 1,4 и 0,3, соответственно. Данные параметры представляют ключевую информацию.

Отображение Эно обладает рядом интересных свойств, таких как чувствительность к начальным условиям, случайность и периодичность. Так же как и отображение кот Арнольда оно может использоваться для изменения позиций пикселей изображения и вычисления новых значений.

Хотя отображения обладают свойствами хаоса, их защищенность не гарантирована. Некоторые известные методы криптоанализа могут легко обойти шифры, основанные на хаотических отображениях, если не используются дополнительные методы защиты. Можно комбинировать отображения с целью дополнительного перемешивания. Например, чтобы повысить криптостойкость алгоритма кодирования изображения на основе отображения кот Арнольд, мы можем дополнительно применить к нему кодирование на основе отображения Эно.

Методы кодирования изображений на основе динамического хаоса имеют ряд преимуществ, включая высокую стойкость к криптоанализу, возможность быстрого шифрования больших объемов данных, а также устойчивость к изменениям в исходном изображении. Тем не менее, методы кодирования изображений на основе хаоса требуют больших вычислительных затрат. Более того, система хаоса определяется на действительных числах, тогда как криптосистемы определяются на конечных множествах целых чисел. Одномерные хаотические криптосистемы ограничены небольшим пространством ключей и слабой безопасностью.

В данной главе был представлен аналитический обзор литературы, связанной с методами кодирования изображений. Были рассмотрены аналоги реализуемого метода, такие как RC4, Grain и методы на основе динамического хаоса, включая отображения кота Арнальда и Эно. Каждый метод использует свой подход к шифрованию изображений и может быть использован в зависимости от требований конкретного приложения. На основе выделенных преимуществ и недостатков, для реализации был выбран метод, сочетающий в себе преимущества других аналогов. Проектирование, разработка и тестирование программного средства будет описана в следующих главах.

1. Описание метода
   1. Алгоритм кодирования

Пусть дано изображение *I*0 с глубиной цвета *a-bit* и размером *M* × *N*. Шаги алгоритма кодирования следующие:

1. Сгенерировать 2 случайных вектора *KR* и*KC* длины *M* и *N*, соответственно. Элементы *KR*(*i*)и *KC*(*i*)принимают случайное значение из набора *A* = {0, 1, 2, …, 2*a* – 1}*.* Отметим, что *KR* и*KC* не должны содержать константных значений.
2. Определить количество итераций, *ITERmax*, и инициализировать *ITER* как 0.
3. Инкрементировать счетчик на 1: *ITER* = *ITER* + 1.
4. Для каждой строки *i* изображения *I*0,
   1. подсчитать сумму всех элементов в строке *i*, эта сумма обозначается *a*(*i*)

** (2.1)

* 1. разделить *a*(*i*) по модулю 2, обозначаем *Ma*(*i*),
  2. строка *i* циклически сдивгается вправо или влево на *KR*(*i*) позиций, по следующему правилу: если *Ma*(*i*) = 0, сдвиг вправо, иначе сдвиг влево.

1. Для каждого столбца *j* изображения *I*0,
   1. подсчитать сумму всех элементов в столбце *j*, эта сумма обозначается 𝛽(*j*)

** (2.2)

* 1. разделить 𝛽(*j*) по модулю 2, обозначаем *M*𝛽(*j*),
  2. столбец *j* циклически сдвигается вверх или вниз на *KC*(*j*) позиций, по следующему правилу: если *M*𝛽(*j*) = 0, сдвиг вверх, иначе сдвиг вниз.

Шаги 4 и 5 выше создадут закодированное изображения, обозначаемое *I*SCR.

1. Используя вектор *KC*, побитовая операция XOR применяется к каждой строке закодированного изображения *I*SCR используя следующее выражение:

 (2.3)

где и представляют побитовую операцию XOR и отображение вектора *KC* слева направо, соответственно.

1. Используя вектор *KR*, побитовая операция XOR применяется к каждому столбцу изображения *I*1 используя следующее выражение:

 (2.4)

где представляют отображение вектора *KR* слева направо.

1. Если *ITER* = *ITERmax*, тогда зашифрованное сообщение *IENC* создано и процесс кодирования завершен; иначе алгоритм переходит к шагу 3.

Векторы *KR*,*KC* и максимальное количество итераций *ITERmax* считаются секретными ключами в данном алгоритме шифрования. Однако для быстрой работы алгоритма предпочтительно установиться *ITERmax* = 1 (единственная итерация). И наоборот, если *ITERmax* > 1, тогда алгоритм более надежный, поскольку пространство ключей больше, чем для *ITERmax* = 1

* 1. Алгоритм декодирования

Расшированное изображение, *I*0, восстанавливается из зашифрованного изображения, *IENC* , и секретных ключей, *KR*,*KC* и *ITERmax* как показано далее:

1. Инициализировать *ITER* = 0.
2. Инкрементировать счетчик на 1: *ITER* = *ITER* + 1.
3. Побитовая операция XOR применяется к вектору *KR* и каждому столбцу изображения *IENC* используя следующее выражение:

 (2.5)

1. Далее, используя вектор *KC*, побитовая операция XOR применяется к каждой строке изображения *I*1 используя следующее выражение:

 (2.6)

1. Для каждого столбца *j* изображения *ISCR*,
   1. подсчитать сумму всех элементов в столбце *j*, эта сумма обозначается 𝛽*SCR*(*j*)

** (2.7)

* 1. разделить 𝛽*SCR*(*j*) по модулю 2, обозначаем *M*𝛽(*j*),
  2. столбец *j* циклически сдвигается вверх или вниз на *KC*(*j*) позиций, по следующему правилу: если *M*𝛽(*j*) = 0, сдвиг вниз, иначе сдвиг вверг.

1. Для каждой строки *i* изображения *I*0,
   1. подсчитать сумму всех элементов в строке *i*, эта сумма обозначается *aSCR*(*i*)

** (2.8)

* 1. разделить *aSCR*(*i*) по модулю 2, обозначаем *Ma*(*i*),
  2. строка *i* циклически сдивгается вправо или влево на *KR*(*i*) позиций, по следующему правилу: если *Ma*(*i*) = 0, сдвиг влево, иначе сдвиг вправо.

Если *ITER* = *ITERmax*, тогда зашифрованное сообщение *IENC* расшифровано и процесс декодирования завершен; иначе алгоритм переходит к шагу 2.

* 1. Исследование метода

Для оценки стойкости алгоритма к различным видам криптоанализа разработана программа на языке Python. В тестах использовались изображения «Lena», «Mandrill» и «Peppers», которые являются стандартными тестовыми изображениями для проверки работы алгоритмов обработки изображений. Показатели данного алгоритма были сравнены с показателями алгоритмов, рассмотренных в аналогах.

Для оценки стойкости алгоритма к статистическому криптоанализу были вычислены коэффициенты корреляции между соседними пикселями по горизонтали, вертикали и диагонали, а также информационная энтропия.

Корреляция является мерой, которая показывает зависимость между двумя соседними пикселами в изображении. Коэффициент корреляции может быть вычислен следующим образом:

 (2.9)

 (2.10)

 (2.11)

где *xi* – яркость *i*-го пиксела; *yi* – яркость соседнего по горизонтали, вертикали или диагонали (в зависимости от типа корреляции) к *i*-му пиксела; ,– средние значения яркости

Коэффициенты корреляции для незашифрованного изображения, как правило, имеют значения, близкие к единице. Это означает, что соседние пикселы связаны между собой некоторой зависимостью. Для зашифрованного же изображения коэффициент корреляции должен стремиться к нулю. Чем ближе коэффициент к нулю, тем меньше связаны соседние пикселы.

Таблица 2.1 – Коэффициенты корреляции

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Изображение | | Корреляция | | | | |
| Рубик | RC4 | Grain | Arnold’s cat map | Henon map |
| «Lena 512×512» | Откр. |  |  |  |  |  |
| Зашифр. |  |  |  |  |  |
| «Mandril 512×512» | Откр. |  |  |  |  |  |
| Зашифр. |  |  |  |  |  |
| «Peppers 512×512» | Откр. |  |  |  |  |  |
| Зашифр. |  |  |  |  |  |

Важной характеристикой изображений служит информационная энтропия, которая является мерой неопределенности и показывает, насколько случайны значения яркости пикселей. Она дает количественную оценку информации, содержащейся в данных, как правило, в битах или битах на символ.

Информационная энтропия дает количественную оценку информации, содержащейся в данных, как правило, в битах или битах на символ. Энтропия вычисляется по формуле

 (2.12)

где P(mi) – вероятность символа *mi*.

Для источника, который выдает 28 символа с равной вероятностью, энтропия будет равна восьми. Следовательно, чем ближе значение энтропии изображения к восьми, тем ближе данное изображение к случайному. Для изображений с высокой степенью случайности (высокой энтропией), сложнее проводить криптоанализ, поскольку статистические методы могут оказаться менее эффективными в выявлении зависимостей между пикселями.

Таблица 2.2 – Информационная энтропия

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Изображение | Энтропия | | | | | |
| Исходное | Зашифрованное по алгоритму | | | | |
| На основе принципа кубика Рубика | RC4 | Grain | Кот Арнольда | Отображение Эно |
| «Lena 512×512» | 7.77237 | 7.99957 | 7.99979 | 7.99974 | 7.77237 | 7.98261 |
| «Mandril 512×512» | 4.71737 | 7.94844 | 7.99979 | 7.99978 | 4.71737 | 7.64330 |
| «Peppers 512×512» | 7.66982 | 7.99945 | 6.78795 | 7.99972 | 7.66982 | 7.98067 |

Дифференциальный криптоанализ – это один из наиболее популярных видов криптоанализа. Суть его заключается в следующем: криптоаналитик или взломщик создают небольшое изменение в исходном изображении, затем шифруют исходное и измененное изображения, после чего ищут различия в двух шифрах, чтобы найти закономерности между изменениями в шифрах и исходных изображениях [4].

Открытый текст изображения зашифровывается и получается изображение-шифр C1. Затем выбирается произвольный пиксел в открытом тексте, чтобы обеспечить небольшое изменение, которое добавляется (вычитается) к его десятичному значению, или переключается младший значащий бит. Измененное изображение шифруется с использованием того же ключа для получения нового изображения-шифра C2. Эти два изображения-шифра сравниваются с помощью следующих критериев:

1. Процента измененных пикселов (NPCR – Near Pixel Change Rate)

Чем ближе коэффициент NPCR к 100 %, тем большую стойкость имеет рассматриваемый алгоритм к дифференциальному криптоанализу.

1. Среднего изменения интенсивности (UACI – Unified Averaged Changed Intensity) – меры различия средней интенсивности между двумя шифрами. Чем ближе данный показатель к 33 %, тем больше стойкость к дифференциальному криптоанализу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Изображение | UACI, % | | | | |
| На основе принципа кубика Рубика | RC4 | Grain | Кот Арнольда | Отображение Эно |
| «Lena 512×512» |  |  | 30.657924078661125 |  |  |
| «Mandril 512×512» |  |  | 29.685606613672412 |  |  |
| «Peppers 512×512» |  |  | 32.248282837723416 |  |  |

1. Индекс структурного сходства (SSIM –­ Structure Similarity) – метод измерения схожести между двумя изображениями.

Также проведено тестирование производительности алгоритма. Для этого данным алгоритмом с использованием сжатия без потерь было зашифровано тестовое изображение «Мандрил» в различных разрешениях (табл. 2.1).

Также было проведено тестирование времени работы метода в зависимости от значений параметра ITERmax.

Тестирование проводилось на компьютере с центральным процессором Intel Core i7-10675G.

Таблица 2.3 – Оценки времени шифрования изображения «Mandrill»

|  |  |  |  |
| --- | --- | --- | --- |
| Разрешение | Количество пикселов | Размер, кбайт | Время шифрования, с |
| 32×32 |  | 2,61 |  |
| 64×64 |  | 9,31 |  |
| 128×128 |  | 36,5 |  |
| 256×256 |  | 143 |  |
| 512×512 |  | 769 |  |

В данной главе был описан реализуемый метод кодирования изображений на основе принципа кубика Рубика. Были представлены алгоритмы кодирования и декодирования изображений. Было проведено исследования разработанного проведенного метода, анализ качества кодирования, а также анализ стойкости метода к атакам. Было также проведено сравнение с аналогами.

1. Описание программного средства
   1. Описание технологии

В данной главе описаны технологии, которые были использованы для разработки программного средства, для кодирования изображений на основе принципа кубика Рубика. Для начала рассмотрим системы и технологии, которые мы будем использовать в данном курсовом проекте.

Python 3 – высокоуровневый язык программирования общего назначения. Стандартная библиотека включает большой набор полезных переносимых функций. Дополнительные возможности могут реализовываться посредством обширного количества сторонних библиотек. В данном проекте были использованы следующие библиотеки:

* Pillow – оригинальная библиотека Python для работы с изображениями;
* NumPy – библиотека, применяемая для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй;
* Tkinter – это кроссплатформенная библиотека для разработки графического интерфейса на языке Python.
* Json ­– встроенный модуль Python, используемая в проекте для хранения ключа в формате JSON.

Текстовый редактор Visual Studio Code ­– это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения.

Само приложение должно:

* принимать файл изображения пользователя;
* принимать от пользователя информацию, необходимую для генерации ключа и генерировать его;
* кодировать и декодировать изображения по указанному алгоритму.
  1. Разработка

Для реализации этих функций было создано 2 файла, содержащие 2 основные класса. Файл *rubik.py* содержит класс *Rubik*, отвечающий за непосредственную работу алгоритма и содержащий необходимые для этого функции. Функции алгоритмов кодирования, декодирования и генерации ключа представлены на рисунке 3.1.



Рисунок 3.1 – Функции алгоритма

Программная реализация всех функций класса *Rubik* представлена в приложении А.

Функция \_\_*init*\_\_ необходима для инициализации объекта класса *Rubik*. Входным параметром этой функции является строка пути к файлу изображения. В этой же функции происходит инициализации основных массивов, необходимых далее для работы алгоритма. Реализовано это с помощью функции *initialize\_rgb*.

Функция генерации ключа *create\_key* по описанному выше алгоритму генерирует ключевую информацию и сохраняет ее в файл формата JSON. Путь к файлу задается параметром функции. Также задается количество итераций работы алгортма *iter\_max*.

Для кодирования изображений используется функция *encrypt.* В качестве входных параметров эта функция принимает путь, куда будет сохранено закодированное изображение, а также параметры, необходимые для работы функции *create\_key.*

Для декодирования изображений используется функция *decrypt.* В качестве входных параметров эта функция принимает путь, куда будет сохранено декодированное изображение, и путь к ключу*.*

Файл *gui.py* содержит класс *GUI*, необходимый для визуализации работы алгоритма. Основные функции класса представлены на рисунке 3.2.

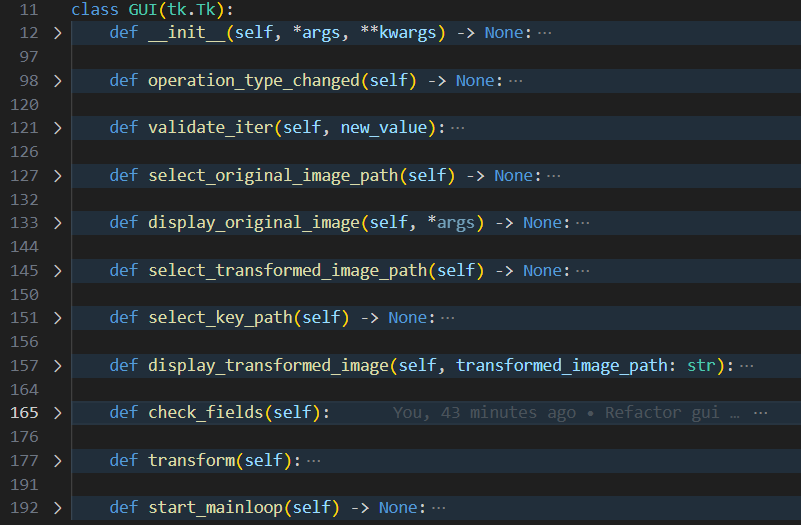


Рисунок 3.2 – Функции класса *GUI*

Программная реализация всех функций класса представлена в приложении Б.

С помощью представленных функций происходит создание элементов интерфейса приложения и обработка взаимодействия с ними. Класс *GUI* создает объект класса *Rubik* для реализации алгоритма.

* 1. Тестирование

На данном этапе курсового проекта проведем тестирование разработанного приложения, чтобы удостовериться в правильности выполнения функция, а также проверить программу на отказоустойчивость.

Ниже приведены основные исключительные ситуации.

Для кодирования необходимо выбрать исходное изображение. Приложение поддерживает кодирование и декодирование изображений в форматах *png* и *bmp*. Для выбора изображения можно воспользоваться полем для ввода или кнопкой, открывающее окно файлового менеджера, позволяющего открывать файлы только в указанном формате (рисунок 3.3).

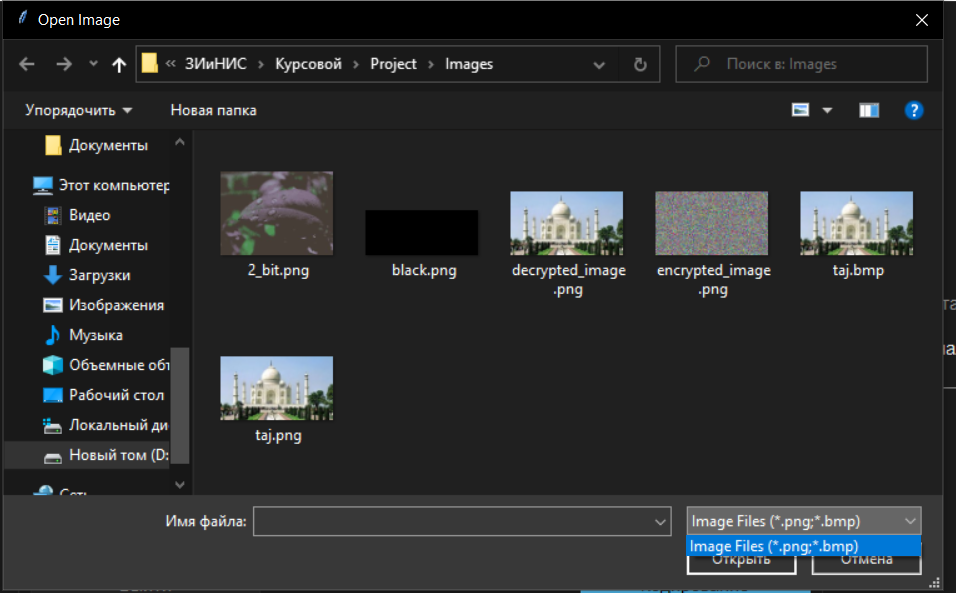


Рисунок 3.3 – Окно файлового менеджера

Если, воспользовавшись полем для ввода, пользователь ввел некорректное имя файла или указанный файл не существует, ему будет показана ошибка (рисунок 4.2).

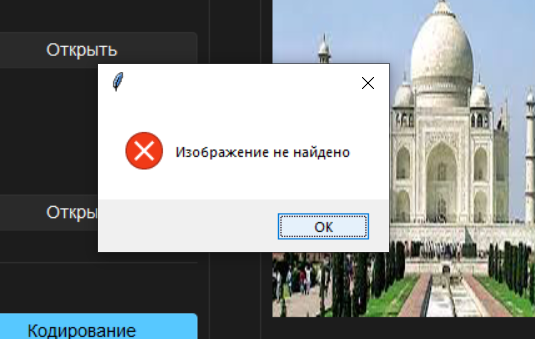


Рисунок 3.4 – Изображение не найдено

Для ввода количества итераций используется поле для ввода, в которое можно вводить только цифры, любые другие символы вставить невозможно. Как видно из рисунка 4.3, цифры, в данном случае «333», были вставлены, но буква «к», введенная с экранной клавиатуры – нет.

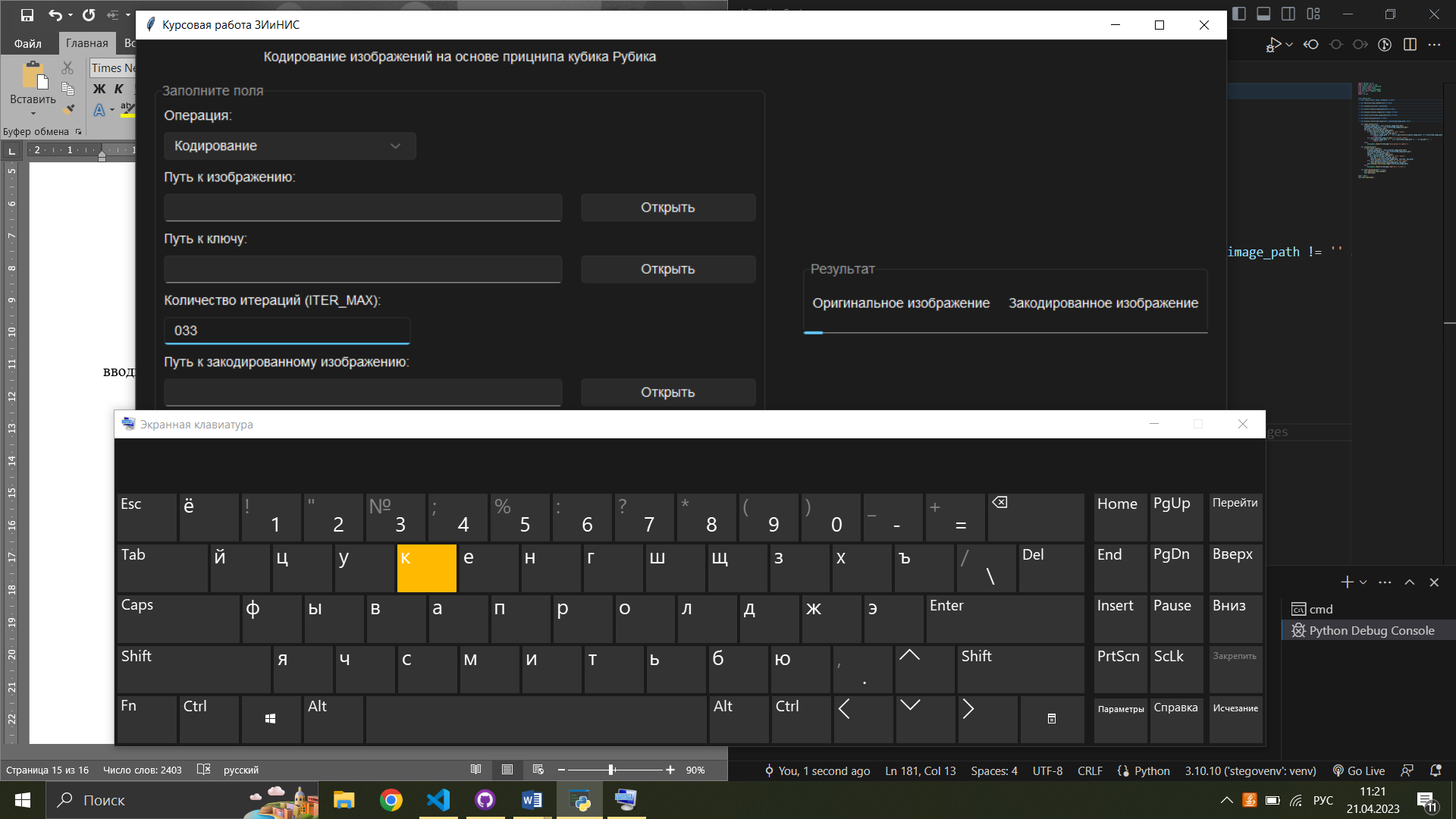


Рисунок 3.5 – Заполнение поля количества итераций

Полученный файл сохраняется в виде изображения. Для выбора изображения можно воспользоваться полем для ввода или кнопкой. Если было выбрано уже существующее изображение, оно будет перезаписано. Если пользователь ввел путь с указанием неправильного формата, ему будет показана ошибка (рисунок 4.4).

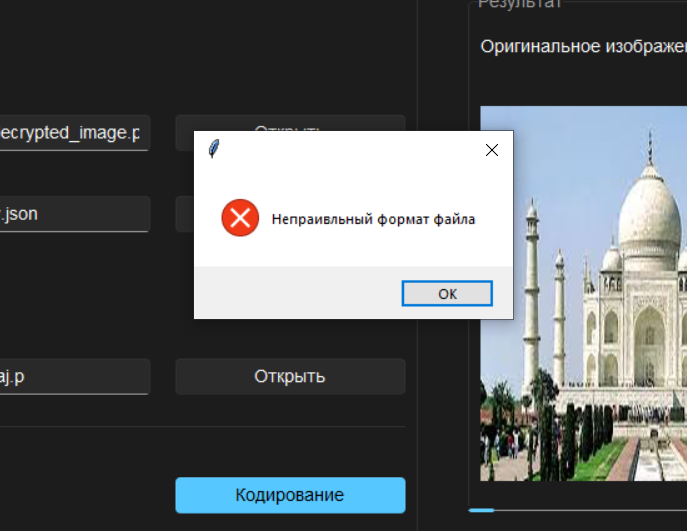


Рисунок 3.5 – Неправильный формат файла

Следует отметить, что приложение разработано таким образом, чтобы минимизировать возможность появления пользовательских ошибок.

* 1. Руководство пользователя

Чтобы запустить приложение нужно запустить exe-файл – Cubic.exe. При запуске открывается окно приложения (рисунок 5.1). Окно содержит окна для ввода и вывода данных и кнопки для совершения операций кодирования и декодирования.

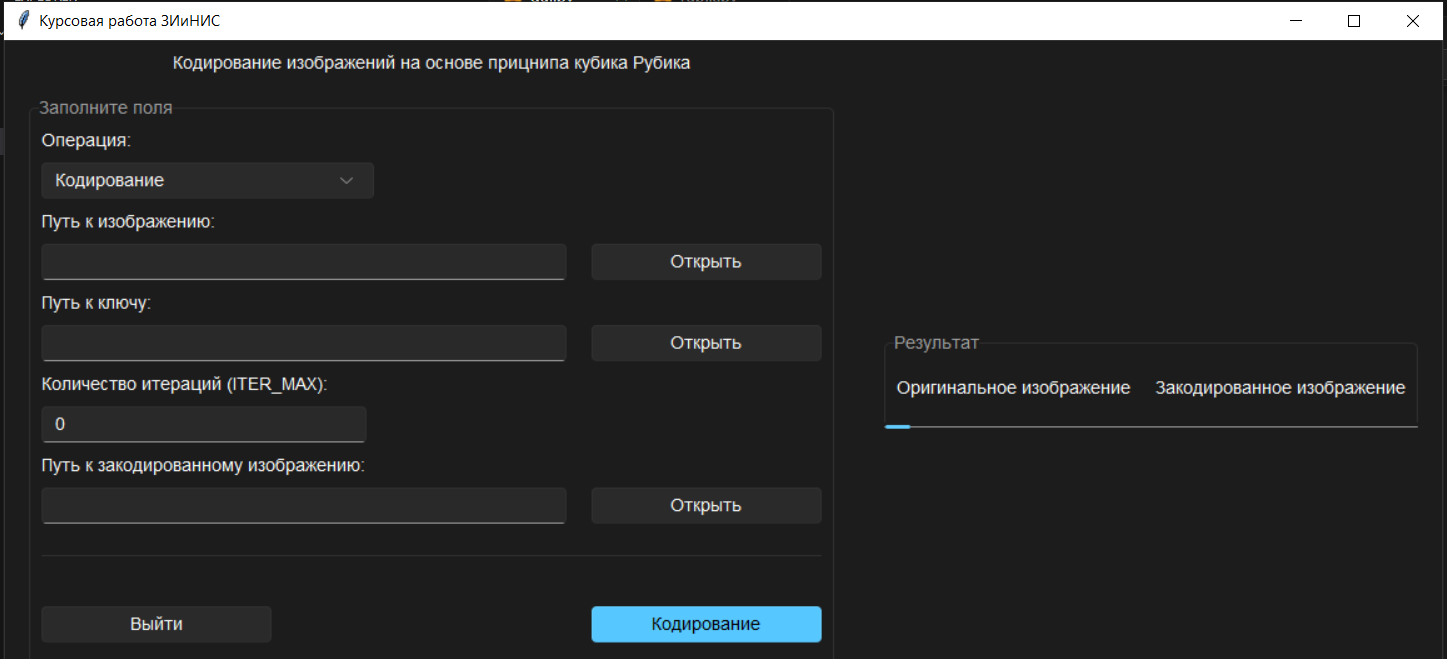


Рисунок 3.6 – Окно приложения

Проведем кодирование изображения. После запуска приложения необходимо выбрать файл исходного изображения в формате png или bmp. Воспользуемся соответствующей кнопкой «Открыть». Появится окно файлового менеджера (рисунок 5.2). С помощью него выберем изображение.

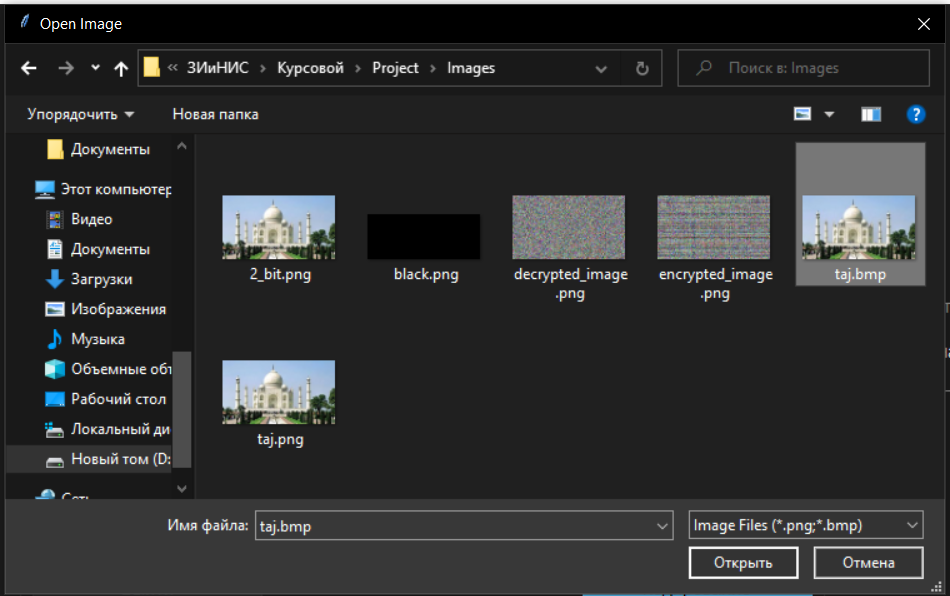


Рисунок 3.7 – Окно файлового менеджера

Выбранное изображение отобразится на панели справа (рисунок 5.3).

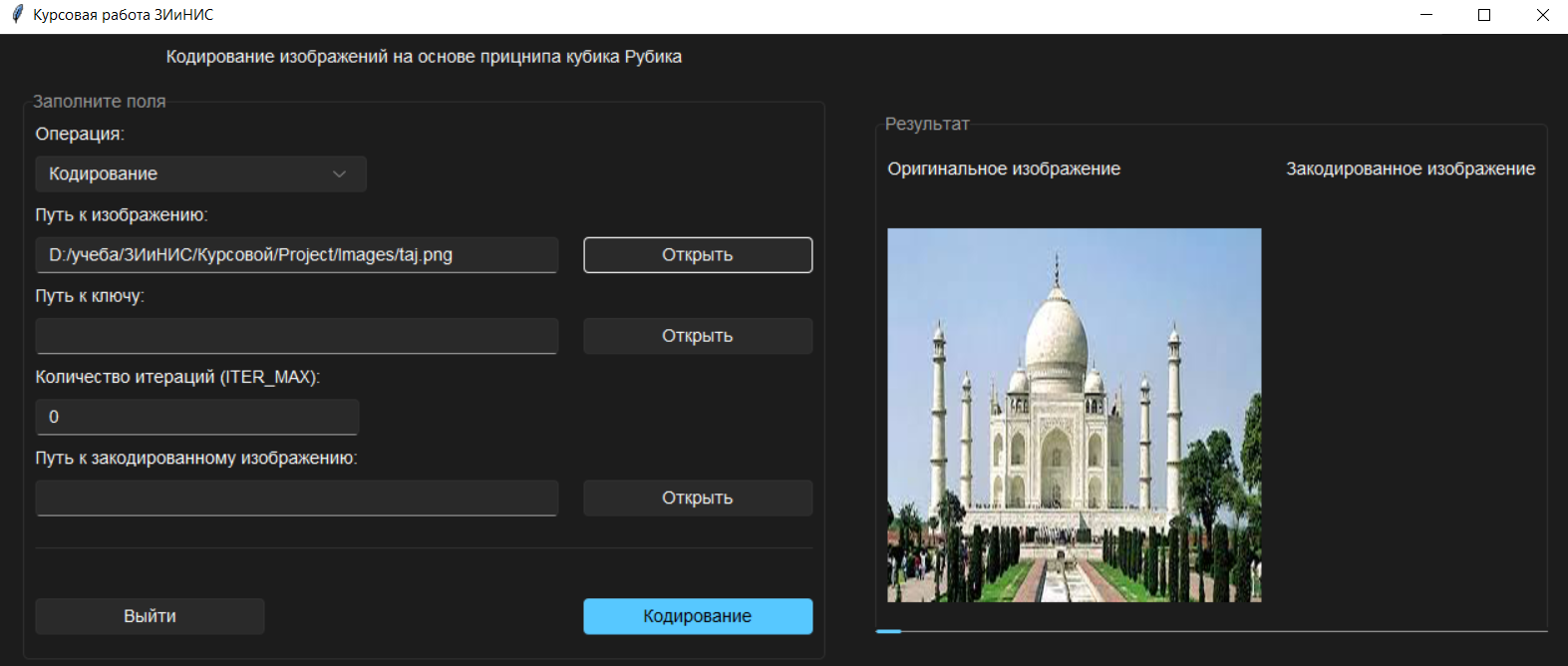


Рисунок 3.8 – Выбранное исходное изображение

Далее необходимо указать путь, куда будет сохранен ключ. Можно опять воспользоваться кнопкой «Открыть» или ввести путь с клавиатуры (рисунок 5.4).

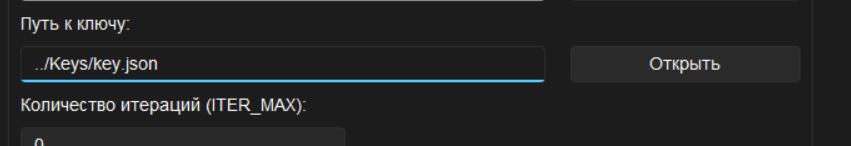


Рисунок 3.9 – Ввод ключа

Далее введем количество итераций для работы алгоритма кодирования.

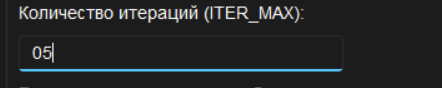


Рисунок 3.10 – Ввод количества итераций

Укажем путь, куда будет сохранено закодированное изображение. Сохраним его в ту же папку, что и исходное под именем *encrypted\_image.png* (рисунок 5.6).

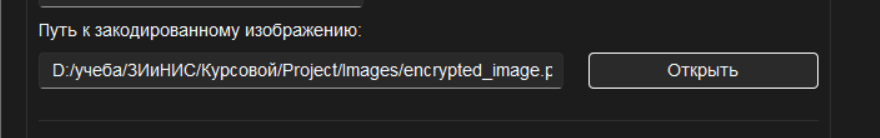


Рисунок 3.11 – Ввод количества итераций

Необходимые поля заполнены. Нажмем на кнопку «Кодирование». После того как алгоритм отработает, справа появится закодированное сообщение (рисунок 5.7). Мы так же можем посмотреть его по указанному ранее пути.

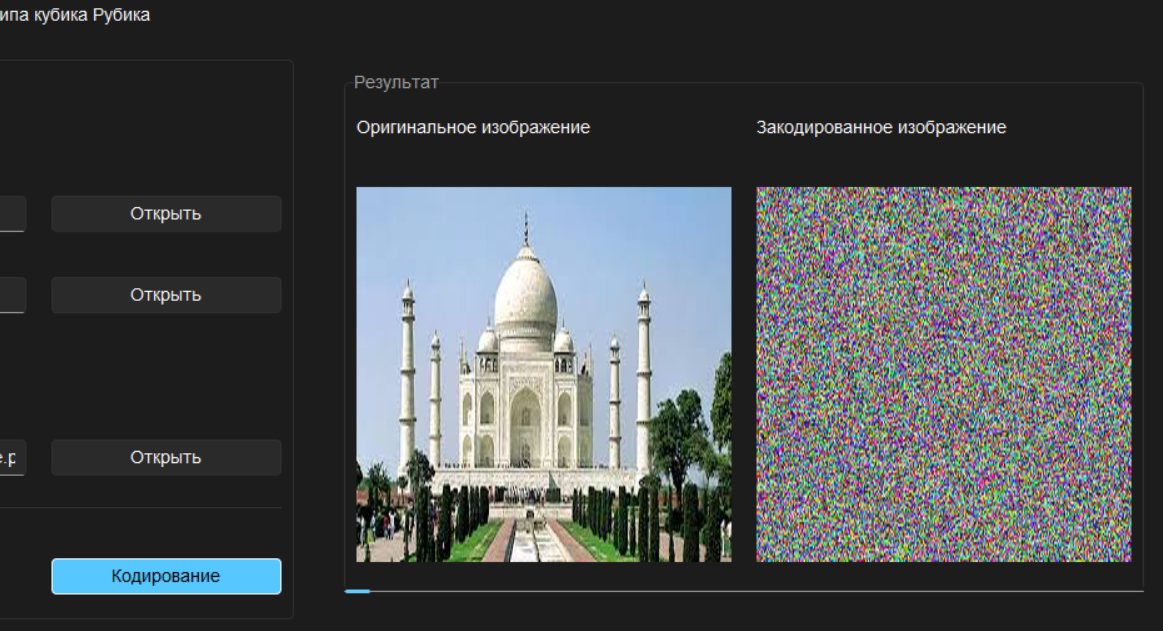


Рисунок 3.12 – Закодированное изображение

С помощью сгенерированных данных проведем декодирование изображения. Для этого необходимо выбрать тип операции «Декодирование» (рисунок 5.8).

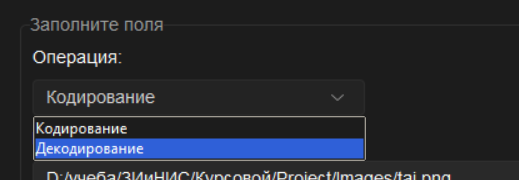


Рисунок 3.13 – Операция декодирования

В качестве исходного изображения укажаем *encrypted\_image.png*, в качестве декодированного ­– *decrypted\_image.png*, путь к ключ – тот же, что использовался для кодирования (рисунок 5.8).

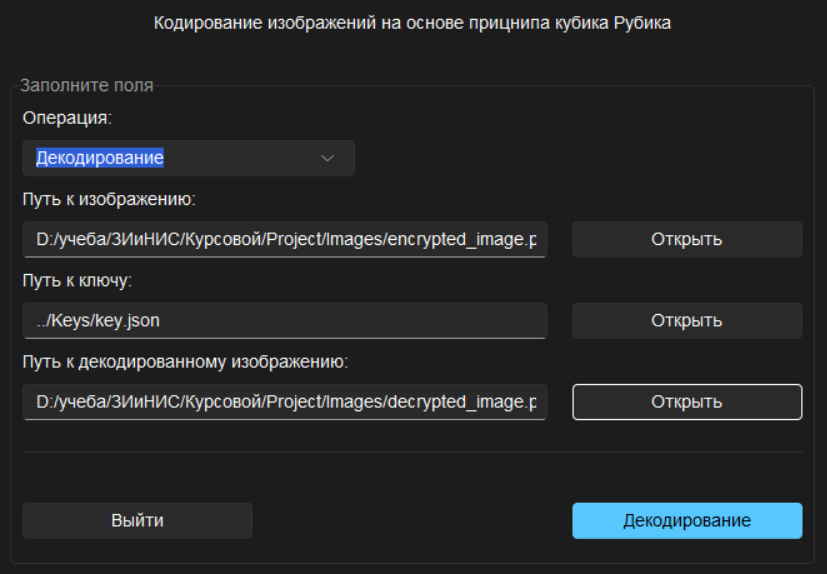


Рисунок 3.14 – Заполнение полей для декодирования

Необходимые поля заполнены. Нажмем на кнопку «Декодирование». После того как алгоритм отработает, справа появится закодированное сообщение (рисунок 5.9).

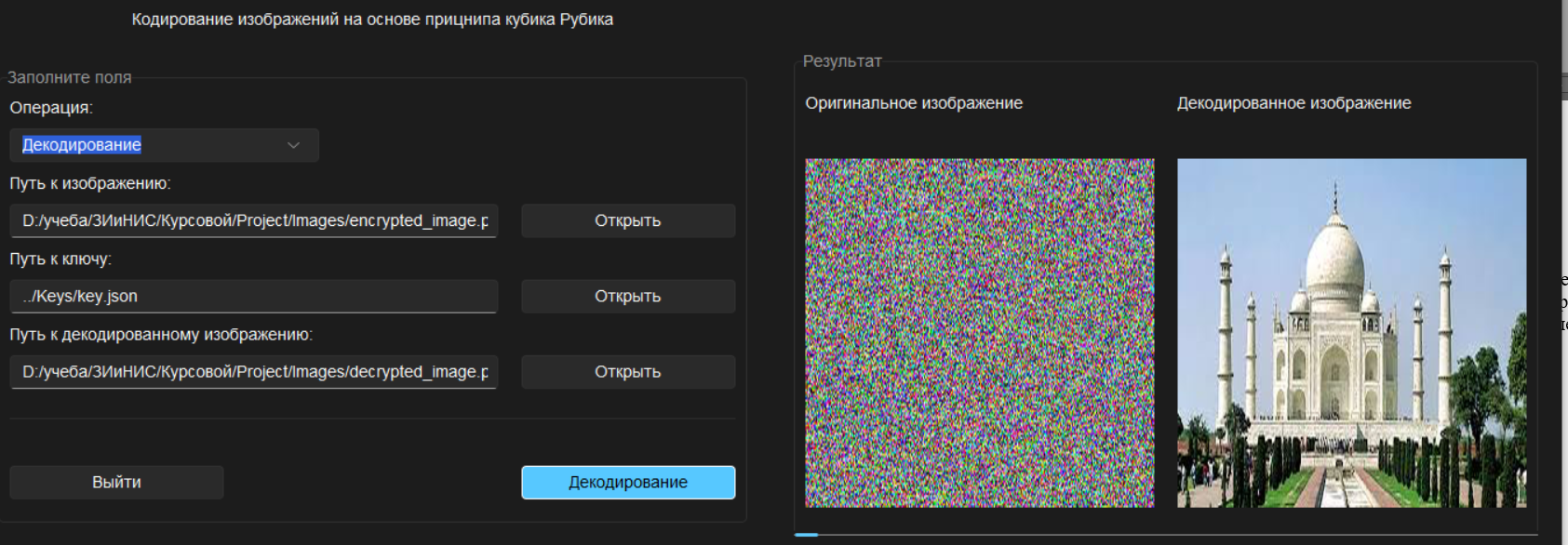


Рисунок 3.15 – Декодированное изображение

В данной главе было проведено описание программного средства, которое было разработано на основе метода кодирования изображений на основе принципа кубика Рубика. Были рассмотрены технологии, использованные при разработке программного средства, а также описаны этапы разработки и тестирования.

Было проведено тестирование программного средства, которое показало его эффективность и правильную работу. Также было составлено руководство пользователя, которое описывает все необходимые действия для работы с программным средством.

В целом, разработанное программное средство является эффективным и может быть использовано для защиты изображений в различных областях. Благодаря разработанному руководству пользователя, его использование становится более простым и доступным для пользователей.

Заключение

В ходе выполнения курсового проекта была достигнута поставленная цель по реализации и исследованию метода кодирования изображений на основе принципа кубика Рубика. Были созданы необходимые классы, функции и пользовательский интерфейс.

При разработке проекта выполнены следующие задачи:

* реализовать программное средство кодирования изображений на основе принципа кубика Рубика;
* оценить криптостойкость метод и сравнить с другими методами кодирования;
* исследовать производительность разработанного метода кодирования изображений и сравнить с другими методами кодирования;
* составить руководство пользователя.

В процессе выполнения проекта были рассмотрены различные методы кодирования изображений. Был проведен их анализ и сравнение.

На следующем этапе было спроектировано приложение, а также выбраны технологии для его реализации. Этими технологиями стали язык программирования Python и его библиотеки, редактор кода Visual Studio Code.

Результатом третьего этапа стало приложение, рассчитанное для использования в ОС Windows. Приложение предоставляет все регламентированные функции, а также графический интерфейс.

По результатам тестирования на следующих этапах были устранены и исправлены всевозможные ошибки, а также проверена работоспособность, и отказоустойчивость программного средства.

Было составлено руководство пользователя.

По результатам выполнения программы можно сделать вывод о том, что разработанная программа работает должным образом и требования технического задания полностью выполнены.

Список используемых источников

1. Урбанович, П. П. Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 2. Криптографические и стеганографические методы защиты информации / П. П. Урбанович, Н. П. Шутько. – Минск: БГТУ, 2020. – 93 с. (<https://elib.belstu.by/handle/123456789/37338>)
2. Martin Hell. Grain – A Stream Cipher for Constrained Environments / Martin Hell, Thomas Johansson, Willi Meier. – Dept. of Information Technology, Lund University, Lund, Sweden, 2005. – 2с.
3. Бекман И. Н. Нелинейная динамика сложных систем: теория и практика / И.Н. Бекман – Москва: МГУ, 2018. – 572с.
4. Wu, Y. NPCR and UACI randomness tests for image encryption / Y. Wu, J.P. Noonan, S. Agaian // Multidisciplinary journals in science and technology. Journal of selected areas in telecommunications (JSAT). – 2011. – Apr. ed. – P. 31–38.

ПРИЛОЖЕНИЕ А