МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

Допустить к защите
Заведующий кафедрой
канд. физ.-мат. наук, доц.
_____В.В. Подколзин
(подпись)
_____2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

ПОДСИСТЕМА ЗАЩИТЫ ГРАФИЧЕСКИХ ДАННЫХ С ПОМОЩЬЮ ОБЪЕКТОВ СО СКРЫТЫМ ИДЕНТИФИКАЦИОННЫМ СЛОЕМ

Работу выполнил _____ С.О. Брезицкий
(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Программирование и информационные технологии

Научный руководитель
канд. техн. наук, доц. _____ А.А. Полупанов
(подпись)

Нормоконтролер
канд. пед. наук, доц. _____ А.В. Харченко
(подпись)

Краснодар
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

Допустить к защите
Заведующий кафедрой
канд. физ.-мат. наук, доц.
_____В.В. Подколзин
(подпись)
_____2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

ПОДСИСТЕМА ЗАЩИТЫ ГРАФИЧЕСКИХ ДАННЫХ С ПОМОЩЬЮ ОБЪЕКТОВ СО СКРЫТЫМ ИДЕНТИФИКАЦИОННЫМ СЛОЕМ

Работу выполнил_____С.О. Брезицкий
(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Программирование и информационные технологии

Научный руководитель
канд. техн. наук, доц._____А.А. Полупанов
(подпись)
Нормоконтролер
канд. пед. наук, доц._____А.В. Харченко
(подпись)

Краснодар
2024

РЕФЕРАТ

Выпускная квалификационная работа 58 с., 18 рис., 18 источников.

СТЕГАНОГРАФИЯ, ЗАЩИТА, ГРАФИЧЕСКИЕ ДАННЫЕ, МЕТОД НАИМЕНЕЕ ЗНАЧИМЫХ БИТ, ПОДСИСТЕМА

Цель работы: спроектировать и разработать подсистему для защиты изображений с помощью объектов со скрытым идентификационным слоем.

В процессе работы были изучены: основные определения в сфере защиты информации, основные методы защиты графических данных, средства и процесс разработки методов защиты графических данных.

В практической части была разработана подсистема, при помощи которой пользователи смогут включать в изображение скрытый текст, а также извлекать его. Данный текст можно использовать для защиты авторских прав на изображение. Представлена программа, которая реализует подсистему.

Инструментарий разработки: язык программирования Python, библиотеки Tkinter, NumPy, Python Imaging Library, threading, PyInstaller.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, где цифровые технологии проникают во все сферы нашей жизни, вопросы защиты информации приобретают особенно актуальное значение. Особое внимание уделяется защите графических данных, которые могут содержать в себе конфиденциальную информацию, быть объектом интеллектуальной собственности или нести в себе важные визуальные сообщения. В этой связи разработка эффективных методов защиты графических данных становится приоритетной задачей в области информационной безопасности.

Одним из перспективных направлений в этом аспекте является создание подсистемы защиты графических данных с использованием объектов со скрытым идентификационным слоем. Данный подход позволяет не только обеспечить высокий уровень защиты информации, но и сохранить исходное качество графических данных, что является важным условием для многих прикладных задач.

Цель данной выпускной квалификационной работы заключается в разработке и анализе подсистемы защиты графических данных, которая бы интегрировала в себя современные методы стеганографии и криптографии с целью создания надежного и эффективного механизма защиты. Основное внимание уделяется изучению возможностей скрытого внедрения идентификационных данных в графические объекты без ухудшения их визуальных характеристик и обеспечения возможности их последующей верификации.

В рамках работы будут решены следующие задачи:

– анализ существующих методов защиты графических данных и их классификация;

– исследование принципов работы и методов внедрения скрытого идентификационного слоя в графические объекты;

– разработка алгоритмов и программного обеспечения для реализации подсистемы защиты;

– тестирование и анализ эффективности предложенной подсистемы на различных типах графических данных;

– разработка рекомендаций по применению подсистемы защиты в различных областях, где актуален вопрос защиты графической информации.

Результаты данной работы могут найти применение в различных сферах, где важна защита графической информации: от авторского права и защиты интеллектуальной собственности до областей, связанных с обеспечением информационной безопасности в целом. Разработанная подсистема предлагает комплексный подход к защите графических данных, сочетая в себе преимущества различных методов и технологий, что делает ее важным вкладом в развитие современных методов обеспечения информационной безопасности.

Методологической основой исследования является комплексный подход, включающий в себя анализ литературных источников по теме, проектирование и разработку программного обеспечения, а также его последующее тестирование и анализ полученных результатов. Работа состоит из двух основных разделов, включая теоретический обзор, описание процесса разработки приложения, тестирование и выводы.

1 Анализ существующих методов защиты графических данных и их классификация

1.1 Определения в сфере защиты информации

В сфере защиты информации существует множество терминов и определений, которые помогают структурировать понимание различных аспектов безопасности данных.

ГОСТ Р 50922-2006 «Защита информации. Основные термины и определения» [1] – это российский стандарт, который устанавливает и определяет основные термины и определения в области защиты информации. Этот стандарт является частью системы стандартов по защите информации и предназначен для использования организациями и индивидуальными специалистами, занимающимися вопросами обеспечения безопасности информации в различных сферах.

ГОСТ Р 50922-2006 включает в себя определения таких понятий, как «информационная безопасность», «конфиденциальность информации», «целостность информации», «доступность информации», а также множество других терминов, касающихся различных аспектов защиты информации. Этот стандарт помогает унифицировать терминологию в области информационной безопасности, облегчая тем самым взаимопонимание между специалистами и организациями.

Важность ГОСТ Р 50922-2006 заключается в создании общего языка для обсуждения вопросов информационной безопасности, что критически важно для эффективного планирования, реализации и аудита мер по защите информации в организациях. Также стандарт способствует облегчению обмена информацией о рисках безопасности, мерах защиты и лучших практиках между различными сторонами, занимающимися защитой информации.

ГОСТ Р 50922-2006 может использоваться в качестве основы при разработке корпоративных стандартов и политик в области информационной безопасности, а также при подготовке нормативной и методической документации. Он также служит важным ресурсом для обучения и сертификации специалистов по информационной безопасности.

Конфиденциальность – это свойство, обеспечивающее, что информация не становится доступной или не раскрывается неавторизованным лицам, процессам или системам. Это одна из основных целей защиты информации, направленная на предотвращение несанкционированного доступа к данным. Это ключевой элемент в триаде основных принципов безопасности информации, известной как «CIA» (Конфиденциальность, Целостность, Доступность). Защита конфиденциальности данных важна во многих сферах, от личной приватности до корпоративной секретности и национальной безопасности. Защита личных данных от несанкционированного доступа важна для индивидуальной свободы и конфиденциальности. Для компаний важно сохранять в тайне коммерческую информацию, такую как финансовые данные, стратегии развития и интеллектуальная собственность, чтобы предотвратить потерю конкурентного преимущества. Государственные органы должны обеспечивать защиту конфиденциальных данных, касающихся национальной безопасности, чтобы предотвратить шпионаж и другие угрозы [2].

Целостность информации – это сохранение точности и полноты информации и методов её обработки. Это означает, что данные не были изменены неавторизованно, будь то случайно или преднамеренно, в процессе хранения, передачи или обработки. Целостность информации критически важна для поддержания доверия к данным и информационным системам. В бизнесе, правительстве, здравоохранении и многих других секторах надежные и точные данные необходимы для принятия обоснованных решений, обеспечения соответствия законодательству и нормативам, а также для поддержания операционной эффективности и репутации. Целостность данных

может быть подвергнута угрозам как из-за внутренних ошибок, так и в результате внешних атак. К внутренним ошибкам относятся технические неисправности, ошибки в программном обеспечении или ошибки операторов. Внешние угрозы включают в себя вирусы и вредоносное ПО, атаки типа «человек посередине», фишинг и другие виды кибератак, направленные на несанкционированное изменение данных.

Обеспечение целостности информации требует комплексного подхода, включая технические, организационные и процедурные меры, а также постоянное совершенствование методов защиты в соответствии с развитием технологий и изменением ландшафта угроз.

Доступность означает, что информация доступна и используема по запросу авторизованным пользователям. Это включает в себя обеспечение работоспособности и доступности систем и сервисов, необходимых для доступа к данным.

Аутентификация – это процесс подтверждения подлинности. В контексте защиты информации это обычно относится к процессу проверки идентичности пользователя или системы, пытающихся получить доступ к защищенным ресурсам. Цели аутентификации:

− предотвращение несанкционированного доступа к системам и данным;

− установление и поддержание контроля над тем, кто может получить доступ к определенным ресурсам и информации;

− поддержание целостности (необходима гарантия того, что только авторизованные пользователи могут вносить изменения в данные или системы);

− протоколирование и аудит (отслеживание действий пользователей для анализа безопасности и расследования инцидентов) [3].

Авторизация – это процесс предоставления или отказа в правах доступа к ресурсам после успешной аутентификации. Это означает определение того, что разрешено делать аутентифицированному пользователю или системе в

пределах среды или системы. В отличие от аутентификации, которая удостоверяет личность пользователя, авторизация проверяет и предоставляет доступ к функциям и ресурсам в соответствии с присвоенными правами и политиками безопасности.

Шифрование – это процесс преобразования информации в форму, недоступную для понимания без соответствующего ключа, который используется для шифрования и расшифровки данных. Шифрование является ключевым инструментом в стратегиях защиты информации и обеспечения конфиденциальности в цифровом мире, предоставляя средства для защиты данных от взлома, кражи и других форм несанкционированного доступа.

Стеганография – это метод скрытия информации путём внедрения сообщений в неподозрительные носители таким образом, чтобы скрытое присутствие информации было не обнаружимо без специальных методов её извлечения. Цель стеганографии – обеспечить конфиденциальность сообщения, скрыв не только содержание, но и сам факт передачи информации. Это делает её ценным инструментом в областях, где необходимо максимально скрыть факт передачи данных [4].

Цифровой водяной знак – это вид стеганографии, представляющий собой скрытую вставку информации в цифровой контент (например, изображения, видео, аудио) с целью идентификации владельца, подтверждения подлинности или защиты авторских прав [5].

Цифровая подпись – это электронная форма подписи, которая использует криптографические алгоритмы для подтверждения подлинности документа или сообщения и обеспечения целостности его содержания. это криптографический механизм, который позволяет подтвердить подлинность и целостность цифрового документа или сообщения. Это аналог физической подписи в электронном виде, но с дополнительными преимуществами безопасности, предоставляемыми криптографией. Цифровые подписи используются для обеспечения того, чтобы получатель мог удостовериться в том, что сообщение или документ не были изменены после подписания и что

подпись была сделана конкретным отправителем (аутентификация). Генерация подписи происходит следующим образом: при отправке сообщения или документа отправитель использует свой закрытый ключ для создания цифровой подписи. Это делается путем генерации хэша (компактного представления) содержимого сообщения, а затем шифрования этого хэша с использованием закрытого ключа отправителя. Верификация подписи происходит следующим образом: при получении сообщения получатель может проверить подпись, используя открытый ключ отправителя. Получатель генерирует хэш содержимого сообщения так же, как это сделал отправитель, и затем расшифровывает цифровую подпись с использованием открытого ключа отправителя, чтобы получить оригинальный хэш. Если два хэша совпадают, это подтверждает, что сообщение не было изменено после подписания, и подтверждает аутентичность отправителя.

Эти определения составляют основу для понимания принципов и методов защиты информации, а также для разработки и реализации эффективных стратегий и систем безопасности в различных сферах деятельности.

1.2 Классификация методов защиты графических данных

Анализ существующих методов защиты графических данных и их классификация является ключевым этапом в разработке эффективных механизмов обеспечения безопасности визуальной информации. В современной практике применяются разнообразные подходы, каждый из которых имеет свои преимущества и ограничения. Для систематизации этих методов можно выделить несколько основных категорий, основываясь на их принципах работы и целях использования.

Первая категория – криптографические методы. Эти методы включают в себя шифрование данных с целью превращения исходной информации в форму, недоступную для понимания без использования специального ключа.

Криптографические методы обеспечивают конфиденциальность данных и защищают их от несанкционированного доступа.

Симметричное шифрование использует один и тот же ключ для шифрования и расшифровки данных. Например: AES и DES.

DES (Data Encryption Standard) — это симметричный алгоритм блочного шифрования, который был принят в качестве федерального стандарта шифрования в США в 1977 году. DES разработан корпорацией IBM в начале 1970-х годов и стал одним из самых широко используемых алгоритмов шифрования для защиты электронных данных. На протяжении многих лет DES был стандартом де-факто для защиты коммерческой и правительственной информации, включая банковские переводы, конфиденциальную переписку и другие виды чувствительных данных. Данный алгоритм очень широко распространен и хорошо изучен. Он прост в реализации, т.к. был разработан с учетом возможности эффективной реализации как на аппаратном, так и на программном уровне.

AES (Advanced Encryption Standard) — это симметричный алгоритм блочного шифрования, принятый в качестве стандарта шифрования правительством США и широко используемый по всему миру [6].

LSB (Least Significant Bit) модификация: замена наименее значимых битов пикселей изображения информацией для передачи. Метод заключается в модификации наименее значимых битов (LSB) в данных носителя с целью внедрения скрытого сообщения. Из-за того, что изменения вносятся в наименее значимые биты, визуальное или аудиальное восприятие исходного файла остаётся практически неизменным, что делает этот метод особенно привлекательным для скрытой передачи информации. Скрытие информации: Внедрение конфиденциальных сообщений или данных в медиафайлы для передачи без привлечения внимания. Водяные знаки: Внедрение информации о правах на интеллектуальную собственность в цифровые медиа. Правильно реализованный метод LSB делает скрытое сообщение незаметным для невооружённого глаза в изображениях и неслышимым в аудиофайлах. Кроме

того, для внедрения и извлечения информации не требуются сложные алгоритмы.

Методы, основанные на преобразованиях: использование преобразований, таких как DCT (дискретное косинусное преобразование) и DWT (дискретное вейвлет-преобразование), для внедрения информации в коэффициенты преобразования.

Вторая категория – водяные знаки. Водяные знаки – это вид стеганографии, который особенно полезен для защиты авторских прав и подтверждения подлинности. Водяные знаки могут быть видимыми или невидимыми и могут содержать информацию о владельце авторских прав, оригинальности данных или о других аспектах происхождения и статуса графических данных.

Третья категория – методы обнаружения и предотвращения. Эти методы фокусируются на обнаружении и предотвращении несанкционированного использования или изменения графических данных. Они могут включать в себя системы обнаружения вторжений, анализ поведения и различные техники мониторинга.

Цифровые подписи: предоставление верифицируемого цифрового подтверждения происхождения и целостности данных.

Системы управления цифровыми правами (DRM): контроль за доступом к цифровым медиа и управлением правами на их использование.

1.3 Обоснование выбора языка программирования

Для создания разнообразных приложений используют чаще всего следующие языки программирования:

− C#;

− Python;

− PHP;

− C++;

− Java.

Язык C#, разработанный компанией Microsoft в рамках инициативы .NET в начале 2000-х годов, уже многие годы входит в список самых востребованных языков программирования среди разработчиков программного обеспечения. Синтаксис этого языка похож на синтаксис языков Java и C++, однако у C# своя уникальная концепция, в нём устранены недостатки и недоработки первых двух языков. На нём создают как консольные приложения, так и программы с графическим интерфейсом. C# полностью объектно-ориентированный и он является неотъемлемой частью платформы .NET Framework [7].

Python – это объектно-ориентированный, интерпретируемый, высокоуровневый язык программирования с динамической семантикой, разработанный Гвидо ван Россумом. Встроенные структуры данных высокого уровня в сочетании с динамической типизацией и связыванием делают язык привлекательным для быстрой разработки приложений (Rapid Application Development, RAD). Кроме того, его можно использовать в качестве сценарного языка для связи компонентов программного обеспечения [8].

PHP, изначально сокращение от Personal Home Page, но теперь официально называемый "PHP: Hypertext Preprocessor", является одним из наиболее широко используемых скриптовых языков программирования для разработки веб-приложений. PHP разработан с упором на простоту создания динамических веб-страниц. С течением времени PHP обрел множество функциональных возможностей, включая поддержку объектно-ориентированного программирования (ООП) с выпуском PHP 5, улучшенную поддержку исключений, анонимные функции, пространства имен и многое другое [9].

C++ – компилируемый, статически типизированный язык программирования, является одним из самых популярных языков. Этот язык можно использовать практически для любой задачи программирования, будь то низкоуровневые аппаратные драйверы, целые операционные системы,

программные пакеты для сетевого оборудования и т.д. Производительность программ на C++ трудно сопоставить с любым другим языком программирования, и, как таковой, это язык де-факто для написания быстрых и мощных программ корпоративного класса [10].

Java – строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. На нём написано очень большое количество разнообразных приложений, начиная от банковского ПО и заканчивая встраиваемыми системами. Благодаря гибкости и лаконичному синтаксису Java позволяет моделировать различные объекты и явления, а также решать разнообразные математические задачи. К преимуществам данного языка так же относятся богатейшая экосистема и производительная, стабильная платформа [11].

Результаты сравнения перечисленных языков в контексте парадигм программирования размещены в табличном виде (таблица 1).

Анализ представленной таблицы позволяет сделать несколько выводов относительно поддержки различных парадигм программирования пятью популярными языками программирования: C#, Java, C++, PHP и Python.

Таблица 1 – Результаты сравнения языков программирования

Критерий	Языки				
	C#	Java	C++	PHP	Python
Императивная парадигма	+	+	+	+	+
Объектно-ориентированная парадигма	+	+	+	+/-	+
Функциональная парадигма	-/+	-/+	-/+	-/+	-/+
Рефлексивная парадигма	+	+	-	+	+
Обобщённое программирование	+	+	+	+	+
Декларативная парадигма	-	-	-	-	-

Продолжение таблицы 1

Критерий	Языки				
	C#	Java	C++	PHP	Python
Распределённая парадигма	-/+	+	-	-	-/+

Для разработки подсистемы выбран язык Python, так как он наиболее гибкий и универсальный по сравнению с остальными языками программирования, а также он обладает обширными стандартными библиотеками и библиотекой сторонних пакетов, которые помогают при разработке.

2 Математическое обеспечение

2.1 Общий подход к реализации

Математическое обеспечение для подсистемы защиты графических данных с использованием объектов со скрытым идентификационным слоем может включать в себя несколько ключевых элементов, связанных с обработкой изображений, криптографией и стеганографией. Ниже описан общий подход к реализации такой системы.

Первый шаг – встраивание идентификационного слоя в изображение.

Для встраивания идентификационной информации (например, цифровых водяных знаков или скрытых сообщений) в изображение можно использовать метод наименее значимых битов (LSB), стеганографические методы в преобразованных областях (например, DCT или DWT) или другие техники, обеспечивающие незаметность встраиваемых данных.

Метод наименее значимых битов (LSB): для каждого пикселя изображения изменяем наименее значимый бит (или несколько битов) на биты скрытой информации. Математически это можно описать как:

$$I`(x, y) = I(x, y) \pm M(x, y) \quad (1)$$

где

$I(x, y)$ – исходное изображение;

$I`(x, y)$ – модифицированное изображение;

$M(x, y)$ – встраиваемое сообщение, представленное в битах;

$\pm$ – операция добавления или удаления информации из пикселя [12].

Методы в преобразованных областях:

– применение преобразования, например, дискретного косинусного преобразования (DCT), к блокам изображения:

$$DCT(I) = C(u,v) \quad (2)$$

где

$C(u,v)$ — коэффициенты DCT для блока изображения;

– модификация выбранных коэффициентов DCT для встраивания информации, а затем обратное преобразование для получения модифицированного изображения [13].

Второй шаг – шифрование идентификационной информации.

Перед встраиванием идентификационной информации в изображение её можно зашифровать с использованием симметричных (например, AES) или асимметричных (например, RSA) алгоритмов шифрования для дополнительной защиты.

Шифрование сообщения M с помощью ключа K:

$$C = Encrypt\left(M, K\right) \quad (3)$$

где

C — зашифрованное сообщение.

Третий шаг – извлечение и верификация идентификационной информации.

Процесс извлечения включает обратное применение используемого стеганографического метода для получения зашифрованного сообщения, а затем его расшифровку.

Расшифровка сообщения C с помощью ключа K:

$$M = Decrypt\left(C, K\right) \quad (4)$$

Для верификации целостности данных может использоваться генерация и сравнение хеш-сумм исходного и извлечённого сообщений.

Эти математические операции и методы формируют основу для разработки подсистемы защиты графических данных, обеспечивая не только встраивание и скрытие идентификационной информации в изображениях, но и её защиту с помощью шифрования и последующую верификацию.

Для данной работы выбор метода наименее значимого бита (LSB) для стеганографии обусловлен рядом факторов, которые делают его особенно подходящим для реализации подсистемы защиты графических данных. Вот основные причины выбора метода LSB:

– низкая заметность изменений: LSB позволяет внедрять скрытую информацию в изображения таким образом, что визуальные изменения практически невозможно обнаружить невооружённым глазом. Это обеспечивает высокую степень дискретности и незаметности для внедрённых данных;

– высокая совместимость: LSB можно применять к различным типам изображений, что обеспечивает универсальность метода и его пригодность для работы с широким спектром графических данных;

– эффективность для небольших объёмов данных: данный метод наиболее эффективен для включения небольшого количества информации, что чаще всего является достаточным для внедрения водяных знаков или для целей идентификации и защиты авторских прав;

– гибкость в управлении скрытой информацией: метод позволяет легко управлять процессом внедрения и извлечения информации, предоставляя разработчикам гибкие инструменты для контроля над процессом стеганографии;

– интеграция с методами шифрования: LSB можно использовать в сочетании с криптографическими методами для обеспечения дополнительной защиты внедрённой информации, повышая уровень безопасности подсистемы.

Эти причины делают метод LSB особенно привлекательным для разработки системы защиты графических данных, где важны незаметность, простота реализации и сохранение качества исходных медиаданных.

2.2 Описание алгоритма

Метод наименее значимого бита (Least Significant Bit, LSB) — это популярный метод стеганографии, который используется для сокрытия информации внутри цифровых изображений, аудиофайлов или других типов медиафайлов. Основная идея этого метода – модифицирование наименее значимых битов (LSB) элементов данных (например, пикселей в изображении или выборок в аудиофайле) для внедрения в них секретного сообщения. Такие изменения обычно незаметны для человеческого глаза или уха, что делает метод эффективным для скрытой передачи информации.

Как работает метод LSB в изображениях:

– выбор носителя: выбирается цифровое изображение в качестве носителя для скрытой информации. Изображение должно быть достаточно большим, чтобы вместить в себя внедряемое сообщение;

– подготовка сообщения: секретное сообщение, которое необходимо скрыть, может быть текстом или другими данными. Сообщение преобразуется в двоичный формат;

– модификация LSB: для каждого байта секретного сообщения наименее значимый бит одного или нескольких пикселей изображения-носителя заменяется на бит из сообщения. Процесс повторяется, пока не будет внедрена вся информация.

Например, если пиксель имеет цветовое значение RGB (11010101, 10110010, 01101100), а бит сообщения — 1, то LSB третьего компонента цвета может быть изменён с 0 на 1, делая новое значение (01101101).

Извлечение сообщения: получатель изображения может извлечь скрытое сообщение, зная метод внедрения. Путём обратного процесса – считывания LSB из каждого пикселя в порядке внедрения – получатель может восстановить исходное двоичное сообщение.

Преимущества метода LSB:

– незаметность: правильно выполненное внедрение сообщения обычно незаметно для наблюдателя;

– простота реализации: метод не требует сложных вычислений или алгоритмов.

Недостатки метода LSB:

– уязвимость к изменениям: изменение размера или формата изображения, а также его сжатие могут легко уничтожить скрытое сообщение;

– ограниченный объём данных: объём скрываемой информации ограничен размерами и свойствами носителя.

Типичные применения метода LSB:

– сокрытие конфиденциальной информации: защита личных данных или секретной информации путём скрытия в обычных медиафайлах;

– внедрение идентификаторов авторства или подлинности в цифровые изображения и другие медиа.

Важно отметить, что для повышения устойчивости скрытой информации к различным видам обработки медиафайлов стеганографические методы часто комбинируются с методами криптографического шифрования.

2.3 Проектирование подсистемы

Проектирование подсистемы – это комплексный процесс, который включает в себя различные этапы и аспекты, начиная от определения требований и заканчивая реализацией и тестированием. Далее будут описаны шаги проектирования данной подсистемы.

Первый шаг – сбор и анализ требований.

Первый шаг в проектировании приложения – понимание того, что от него требуется. Это включает в себя определение функциональных и нефункциональных требований, целевой аудитории, и как приложение будет использоваться. Анализ требований помогает определить цели проекта и ограничения [14].

В результате была разработана модель требований в нотации UML, представленная на рисунке 1.

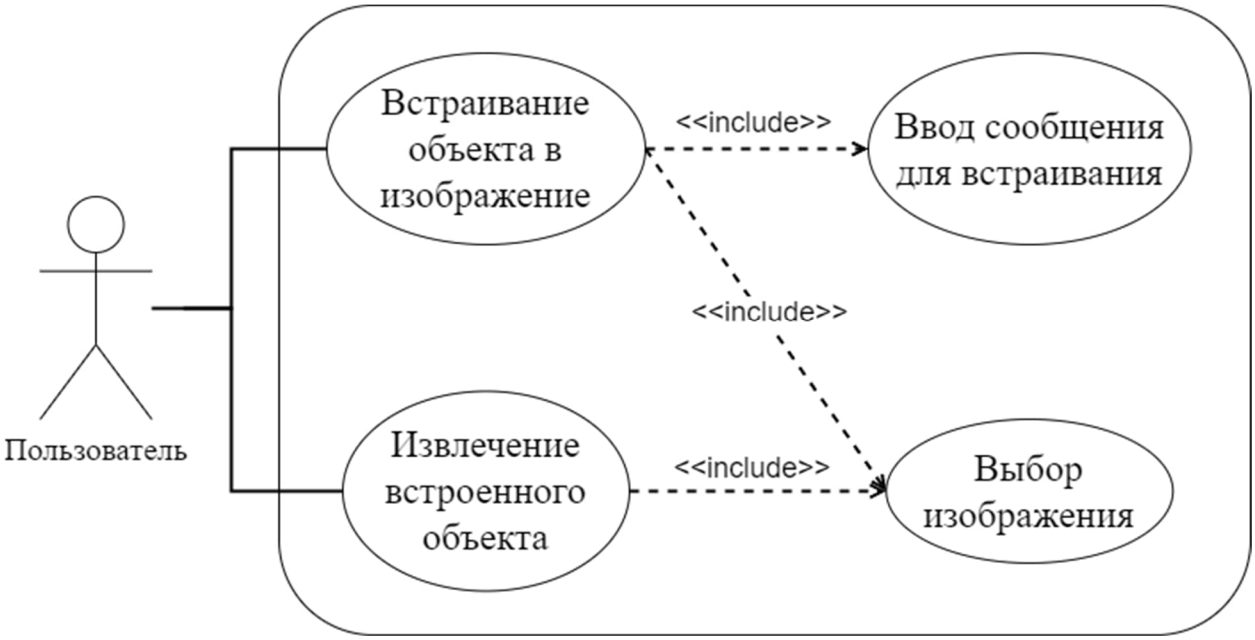

Рисунок 1 – Модель требований (вариантов использования) в нотации UML

Именно такие функции будут у подсистемы.

Второй шаг – проектирование функциональной структуры подсистемы: функционально-ориентированный подход.

Функционально-ориентированный подход рассматривает объект исследования, в данном случае подсистему, как набор функций, преобразующий поступающий поток информации в выходной поток.

Основной методикой реализации функционально-ориентированного подхода является методология IDEF0, которая с помощью наглядного

графического языка IDEF0 представляет структуру системы, её функции и процессы в виде набора взаимосвязанных функций (функциональных блоков), а также потоки материальных объектов и информации, которые преобразуются этими функциями [15].

В итоге была построена функциональная модель подсистемы в виде контекстной диаграммы в нотации IDEF0, представленная на рисунке 2. Также на основе данной диаграммы была создана диаграмма декомпозиции A0 на дочерние подпроцессы, которая показана на рисунке 3.

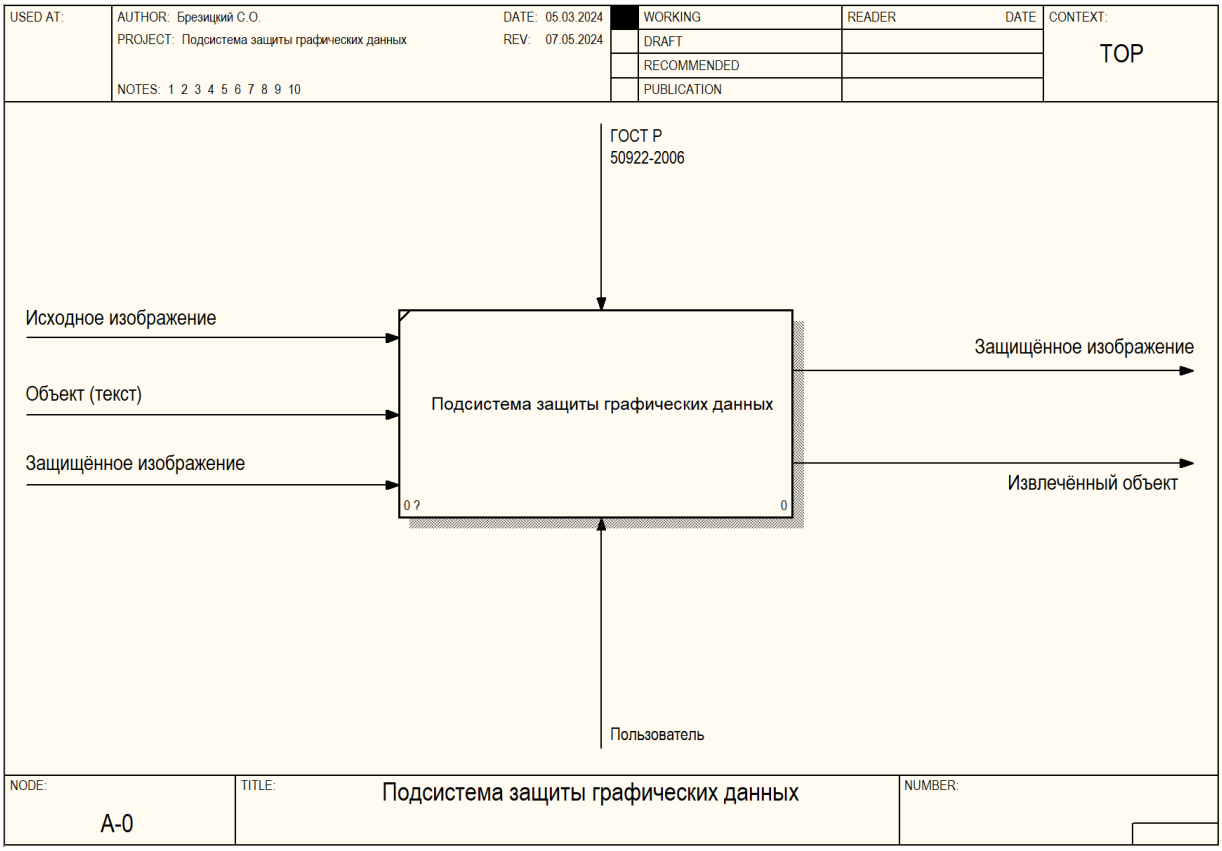

Рисунок 2 – Контекстная диаграмма в нотации IDEF0

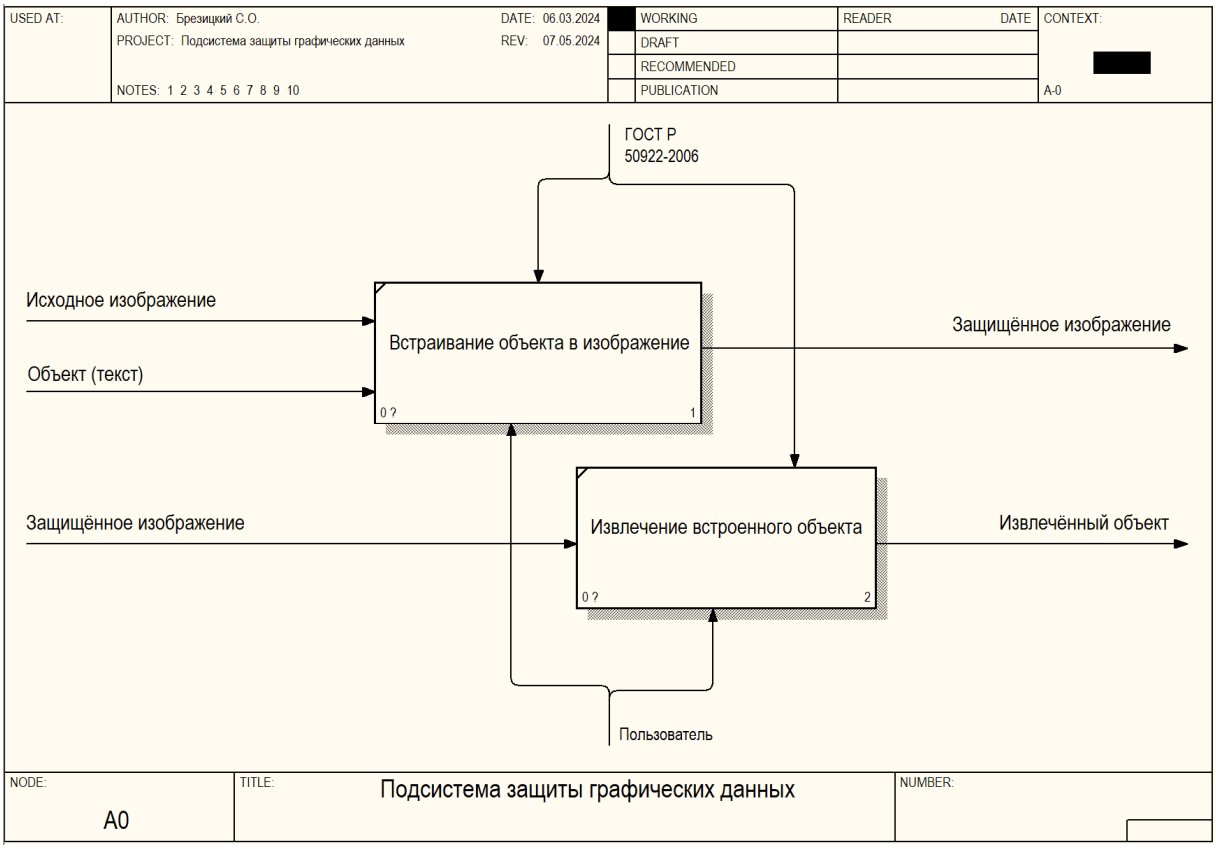

Рисунок 3 – Диаграмма декомпозиции A0

В таблице 2 представлена документация диаграммы декомпозиции A0 данными словарей Activity Dictionary и Arrow Dictionary.

Таблица 2 – Описание элементов функциональной модели

Наименование стрелки	Источник стрелки	Тип стрелки источника	Приёмник стрелки	Тип стрелки приёмника
Исходное изображение	Внешняя граница	Input	Встраивание объекта в изображение	Input
Объект (текст)	Внешняя граница	Input	Встраивание объекта в изображение	Input
Защищённое изображение	Внешняя граница	Input	Извлечение встроенного объекта	Input
ГОСТ Р 50922-2006	Внешняя граница	Control	Встраивание объекта в изображение	Control

Продолжение таблицы 2

Наименование стрелки	Источник стрелки	Тип стрелки источника	Приёмник стрелки	Тип стрелки приёмника
ГОСТ Р 50922-2006	Внешняя граница	Control	Извлечение встроенного объекта	Control
Пользователь	Внешняя граница	Mechanism	Встраивание объекта в изображение	Mechanism
			Извлечение встроенного объекта	
Защищённое изображение	Встраивание объекта в изображение	Output	Внешняя граница	Output
Извлечённый объект	Извлечение встроенного объекта	Output	Внешняя граница	Output

Третий шаг – проектирование функциональной структуры подсистемы: объектно-ориентированный подход.

Объектно-ориентированный подход – это подход к проектированию функциональной структуры, который сочетает систему обозначения для представления логической и физической, а также статической и динамической моделей проектируемой системы и процесс объектно-ориентированной декомпозиции [16].

В результате была построена диаграмма деятельности для всей подсистемы, представленная на рисунке 4.

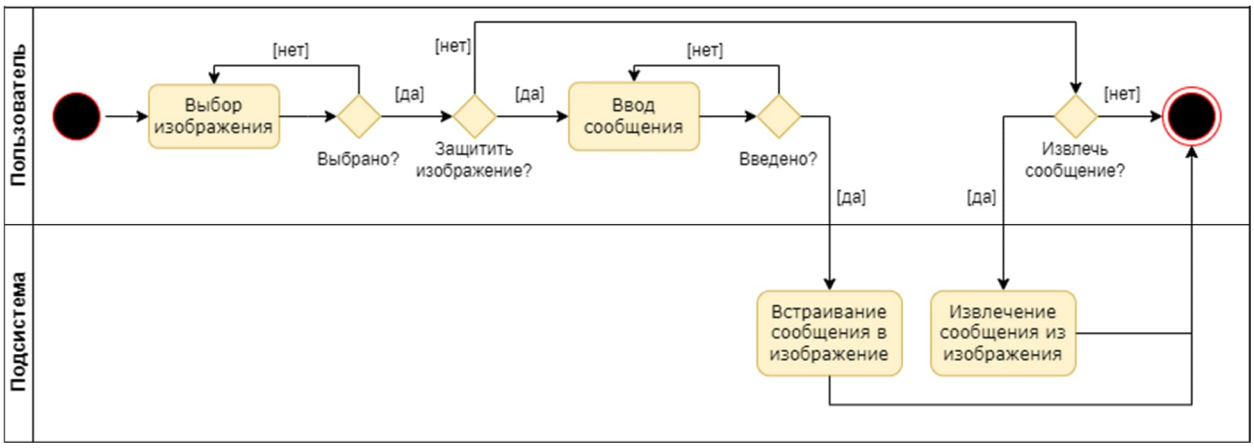

Рисунок 4 – Диаграмма деятельности подсистемы

Четвёртый шаг – проектирование пользовательского интерфейса.

Предлагаемый интерфейс проектируемой подсистемы показан на рисунке 5.

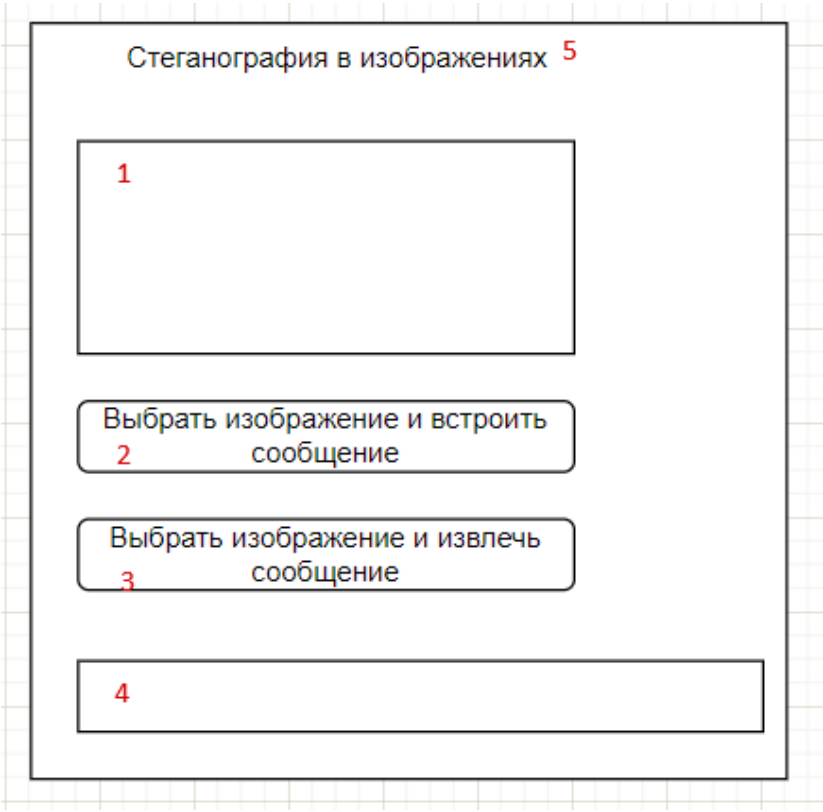

Рисунок 5 – Макет приложения

На данном макете цифрами обозначены:

1 – окно для ввода сообщения;

2 – кнопка для выбора изображения и встраивания сообщения;

3 – кнопка для выбора изображения и извлечения сообщения;

4 – прогресс-бар для отображения прогресса процесса обработки;

5 – заголовок приложения.

3 Разработка и тестирование подсистемы

3.1 Используемые модули

В целом, разработка приложений требует глубоких знаний в области программирования, дизайна, тестирования и управления проектами, а также гибкости и готовности к постоянному обучению и адаптации к новым вызовам.

Для установки Python на компьютер необходимо посетить официальный сайт Python по адресу python.org. На главной странице сайта представлена ссылка на загрузку, которая автоматически предложит версию Python, рекомендованную для операционной системы пользователя, будь то Windows, macOS или Linux.

После загрузки установочного файла его запуск приведет к появлению установщика Python, предлагающего ряд параметров настройки, включая возможность добавления Python в переменную среды PATH. Это позволяет запускать Python из командной строки любой директории. Рекомендуется выбрать данную опцию для удобства использования Python.

Следование инструкциям установщика завершит процесс установки. Проверка успешности установки производится путем открытия командной строки или терминала и ввода команды python --version или python3 --version, в зависимости от операционной системы и настроек. В ответ на эту команду система должна вывести версию установленного Python, что подтверждает успешность установки.

Для пользователей Windows, предпочитающих графический интерфейс для управления пакетами Python и виртуальными средами, рекомендуется установка Anaconda. Anaconda — это дистрибутив Python, включающий в себя обширный набор научных библиотек и инструментов, а также собственный менеджер пакетов Conda.

После завершения установки открываются широкие возможности для

программирования, разработки приложений, анализа данных и других задач, связанных с использованием Python.

Программа написана на языке Python. В ней используются следующие библиотеки:

tkinter – стандартная библиотека для создания графического пользовательского интерфейса (GUI) в Python. Используется для создания окон, диалоговых окон, различных виджетов (кнопок, лейблов, полей ввода и т.д.).

PIL (Python Imaging Library), доступная через модуль Pillow (from PIL import Image) – библиотека для работы с изображениями, включая открытие, манипулирование и сохранение многих форматов изображений. Используется для обработки изображений в контексте стеганографии или других задач, связанных с графикой.

numpy – пакет для научных вычислений с Python, предоставляющий поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (математических) функций для операций с этими массивами [17].

threading – модуль для высокоуровневой работы с потоками. Позволяет выполнять код параллельно, что может быть использовано для улучшения отклика GUI или для выполнения задач в фоновом режиме.

В Python модули устанавливаются с использованием менеджера пакетов pip. Pip является стандартным инструментом, который поставляется с Python начиная с версии 3.4 для Python 3 и Python 2.7.9 для Python 2. Он позволяет устанавливать, обновлять и удалять пакеты из репозитория PyPI (Python Package Index), который содержит тысячи пакетов, разработанных сообществом.

Процесс установки модулей с помощью pip включает выполнение команды в командной строке или терминале. Команда для установки модуля имеет следующий формат:

```
pip install имя_модуля
```

Для установки конкретной версии модуля используется команда:

```
pip install имя_модуля==версия
```

Для обновления уже установленного модуля до последней версии применяется команда:

```
pip install --upgrade имя_модуля
```

Pip также позволяет устанавливать пакеты, указывая файл зависимостей (обычно requirements.txt), который содержит список модулей с указанием версий. Команда для установки пакетов из файла зависимостей выглядит следующим образом:

```
pip install -r requirements.txt
```

Этот инструмент обеспечивает удобный способ управления зависимостями в проектах на Python, автоматизируя процесс установки и обновления библиотек. Важно отметить, что для выполнения команд pip может потребоваться соединение с интернетом для доступа к репозиторию PyPI, если только пакеты не устанавливаются из локального источника [18].

3.2 Описание программного обеспечения подсистемы

3.2.1 Моделирование физических аспектов

Для моделирования физических аспектов подсистемы были использованы два вида диаграмм: диаграмма компонентов и диаграмма развёртывания.

Диаграмма компонентов используется для визуализации статического аспекта физических компонентов программного обеспечения и их отношений. Диаграмма компонентов разработанной подсистемы представлена на рисунке 6.

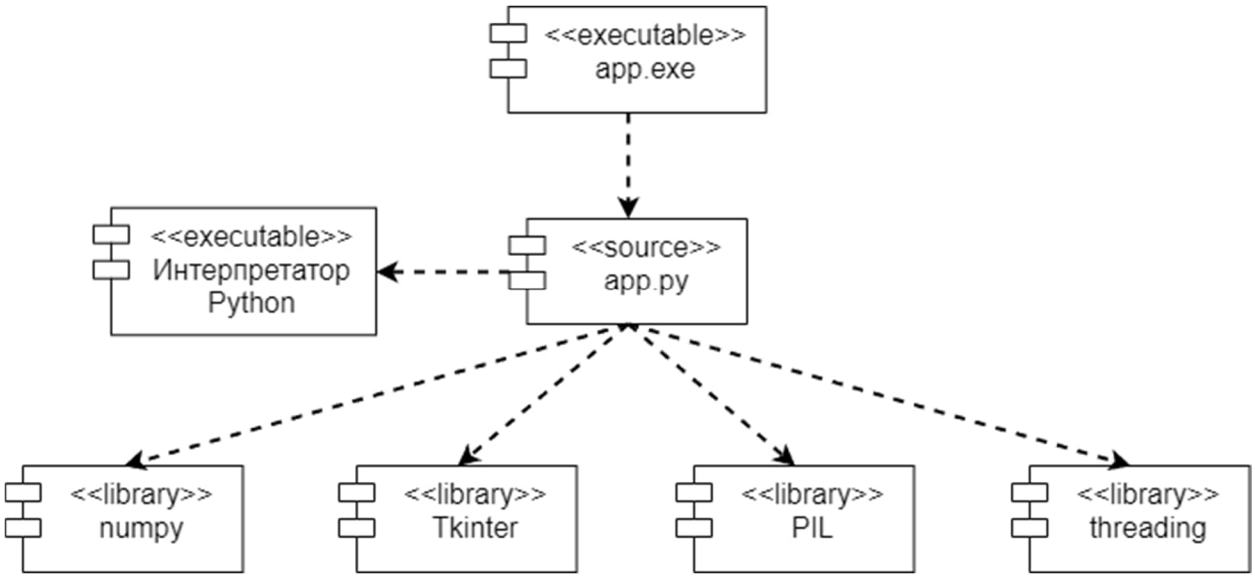

Рисунок 6 – Диаграмма компонентов подсистемы

Диаграмма развёртывания показывает конфигурацию узлов, где производится обработка информации, и то, какие компоненты размещены на каждом узле.

Диаграмма развёртывания для данной подсистемы представлена на рисунке 7.

Рисунок 7 – Диаграмма развёртывания подсистемы

3.2.2 Описание программного кода

Целиком код программы приведен в приложении.

Функция `update_progress_bar(value)` в данной программе необходима, чтобы производить обновление значение параметра value прогресс-бара. Данная функция имеет всего лишь один аргумент, но он является принципиально важным. Это параметр под названием value – т.е. то значение, на которое теперь должен быть обновлен при ближайшей отрисовке прогресс-бар.

Тело данной функции построено из двух строчек кода. Первая строчка имеет следующий незамысловатый вид:

```
progress_bar['value'] = value
```

Смысл данной строки заключается в том, что в ней устанавливается новое значение параметра value. Подразумевается, что `progress_bar` является экземпляром виджета прогресс-бара из Tkinter.

Вторая строка имеет такой вид:

```
root.update_idletasks()
```

Эта строка является вызовом функции, назначение которой – заставить главное окно приложения (которое хранится в переменной root) обновить задачи, которые сейчас являются в режиме ожидания. Благодаря этому можно немедленно отобразить изменения в прогресс-баре.

Следующая функция – это функция `text_to_binary`. Назначение данной функции следующее: она принимает на вход сообщение в виде текстовой строки, а затем преобразует эту строку с помощью специального алгоритма в строку с двоичными данными. Текст на входе в кодировке UTF-8. А на выходе получается двоичный код. Кроме того, обязательно добавляется к текстовому содержимому в начале сообщения длина будущего сообщения. В конце сообщения еще добавляется завершающая последовательность. Завершающая последовательность необходимо для точного определения конца строкового сообщения.

Строки данной функции с краткими пояснениями:

– `binary_message = message.encode('utf-8')`: занимается преобразованием непосредственно исходного текстового сообщения в байтовую строку. Как результат – на выходе получается последовательность байтов. Причем необходимо обязательно отметить тот факт, что поддерживается двухбайтовая кодировка (в частности, UTF-8);

– `message_length = len(binary_message)`: в данной строке программного кода вычисляется длина полученной строки байтов. Таким образом определяется количество байтов, необходимое для хранения закодированного сообщения. В дальнейшей работе программы это число будет использоваться для формирования двоичного представления длины кодируемого сообщения;

– `message_length_binary = format(message_length, '016b')`: применяется для преобразования длины сообщения в бинарную (двоичную) строку. Значение бинарной строки будет состоять из 2 байтов (или 16 бит). Так, использование этого формата '016b' гарантирует дополнение нулями слева до достижения длины в 16 бит. А делается это со следующей целью: это поможет обеспечить необходимое единообразие представления длины сообщения;

– `binary_data = ''.join(format(byte, '08b') for byte in binary_message)`: эта строка программного кода помогает алгоритму выполнить следующие действия: сначала каждый байт из байтовой строки binary_message преобразуется в его бинарное представление (строку из 8 бит), затем все такие бинарные строки (представляющие отдельные байты сообщения) объединяются в одну длинную бинарную строку;

– `return message_length_binary + binary_data + '1111111111111110'`: содержит возврат результата. А результат склеивается из трех подстрок. Первая часть – это бинарное представление длины сообщения (`message_length_binary`). Вторая часть – это бинарные данные самого сообщения (`binary_data`). А третья, последняя часть, является завершающей

последовательностью '1111111111111110'. Завершающая последовательность выбрана случайным образом именно такой. По большому счету, не принципиально, какой она будет. Вместо нее может быть любой фиксированный набор битов. Другое дело, что малая длина последовательности будет приводить к сбоям в работе алгоритма (т.к. может наблюдаться ситуация, когда в исходном файле изображения будет изначально содержаться уже такая последовательность битов – а это, в свою очередь, приведет к досрочному определению завершающей последовательности, и алгоритм даст сбой). Двухбайтовая последовательность завершающая тоже имеет риск быть повторенной, однако такая вероятность составляет примерно 2^{-16}, т.е. 0,0015%. Для еще большего увеличения надежности алгоритма в этом плане можно использовать завершающую последовательность еще большей длины (например, четырехбайтовую).

Функция `binary_to_text` принимает в качестве аргумента строку, представляющую бинарные данные сообщения, и преобразует её обратно в текстовую форму в кодировке UTF-8. Данная функция работает по следующему алгоритму:

– `message_length = int(binary_message[:16], 2)`: данная строка программного кода находит (выделяет) первые два байта (или 16 бит) входной двоичной строки, чтобы найти длину закодированного сообщения. Далее применяется функция `int()`. Она преобразовывает число из двоичной системы счисления в десятичную. Как видно из программного кода, в нем присутствует второй параметр функции int – число 2. Это число является признаком того, что исходное число надо считать как число в двоичной системе счисления;

– `binary_data = binary_message[16:16 + 8 * message_length]`: эта строка кода находит (или, можно сказать, извлекает) бинарные данные сообщения, начиная с 17-го бита (после информации о длине) и заканчивая битом, соответствующим вычисленной длине сообщения. В данном случае

произведение 8 и message_length необходимо для преобразования двоичного кода в однобайтовый (восьмибитный);

– `message = bytearray(int(binary_data[i:i+8], 2) for i in range(0, len(binary_data), 8)).decode('utf-8')`: занимается преобразованием извлеченных двоичных данных обратно в текст. Здесь происходит следующее: во-первых, сначала строка разбивается на части по восемь бит (каждая из этих частей далее преобразовывается в целое число (иными словами, в один байт), во-вторых, затем, все эти полученные байты объединяются вместе в один объект `bytearray`, в-третьих, в самом конце, полученный объект `bytearray` конвертируется в строку символов с использованием кодировки UTF-8;

– `return message`: данная строка программного кода занимается простым делом – она просто возвращает декодированное текстовое сообщение.

Можно сказать, что функция `binary_to_text` выполняет обратное преобразование к функции `text_to_binary`. Она восстанавливает исходное текстовое сообщение из его бинарного представления.

Функция `embed_message` занимается преобразованием (точнее, вставкой) текстового сообщения в изображение. При этом используется стеганография – специальный метод скрытия объекта. В данном случае стеганография основывается на модификации младших битов пикселей изображения. Для понимания сути данной функции необходимо описать аргументы функции и её действия:

– `img`: исходное изображение, в которое будет встроено сообщение;

– `message`: текстовое сообщение для встраивания;

– `progress_callback`: функция обратного вызова для отслеживания прогресса встраивания сообщения;

– `on_complete`: функция обратного вызова, которая вызывается по завершении процесса встраивания.

Детальное описание кода:

– `binary_message = text_to_binary(message)`: преобразует текстовое сообщение в бинарную строку с помощью функции `text_to_binary`;

– `img_array = np.array(img)`: преобразует изображение img в массив numpy для удобства работы с пикселями;

– `total_pixels = img_array.shape[0] * img_array.shape[1]`: вычисляет общее количество пикселей в изображении на основе его размеров;

– `data_index = 0`: инициализирует индекс, который будет использоваться для итерации по бинарному сообщению;

– цикл `for index, (i, j) in enumerate(np.ndindex(img_array.shape[0], img_array.shape[1]))`: итерирует по всем пикселям изображения;

– `pixel = img_array[i, j]`: получает текущий пиксель изображения.

– Вложенный цикл `for n in range(3)`: итерирует по каждому из трех каналов пикселя (RGB);

– `if data_index < len(binary_message)`: проверяет, остались ли ещё биты сообщения для встраивания;

– `pixel[n] = pixel[n] & ~1 | int(binary_message[data_index])`: модифицирует младший бит текущего канала пикселя, встраивая в него бит сообщения;

– `data_index += 1`: увеличивает индекс бинарного сообщения;

– `if data_index == len(binary_message)`: проверяет, все ли биты сообщения встроены;

– `progress_callback(100)`: вызывает функцию обратного вызова для обновления прогресса до 100%;

– `on_complete(Image.fromarray(img_array))`: создает изображение из модифицированного массива и вызывает функцию обратного вызова on_complete;

– `if index % 100 == 0`: регулярно обновляет прогресс встраивания через каждые 100 пикселей;

– `progress = (index / total_pixels) * 100`: вычисляет текущий прогресс встраивания;

– `progress_callback(progress)`: вызывает функцию обратного вызова `progress_callback` с текущим значением прогресса.

По завершении цикла, если сообщение полностью встроено, вызывается `progress_callback(100)` и `on_complete(Image.fromarray(img_array))` для обновления прогресса до 100% и возврата модифицированного изображения.

Таким образом, функция `embed_message` позволяет встроить скрытое сообщение в изображение, изменяя младшие биты пикселей, и обеспечивает возможность отслеживать прогресс и обрабатывать завершение процесса через функции обратного вызова.

Функция `save_image` предназначена для сохранения изображения на диск пользователя с использованием графического интерфейса для выбора местоположения файла. Детальное описание функции:

– `def save_image(img)`: определяет функцию save_image, которая принимает один аргумент:

– `img`: объект изображения, который необходимо сохранить;

– `file_path = filedialog.asksaveasfilename(defaultextension=".png")`: вызывает диалоговое окно для сохранения файла, позволяя пользователю выбрать путь и имя файла для сохранения изображения. Аргумент `defaultextension=".png"` указывает, что по умолчанию предлагаемое расширение файла — `.png`;

– `if file_path`: проверяет, был ли выбран путь файла. Если пользователь закрывает диалоговое окно или отменяет операцию сохранения, `file_path` будет пустым, и следующий блок кода не выполнится;

– `img.save(file_path)`: сохраняет изображение по указанному пути file_path. Метод save является частью API объекта изображения (например, объекта из библиотеки PIL (Python Imaging Library)) и позволяет записать изображение на диск;

– `messagebox.showinfo("Успех", "Изображение успешно сохранено!")`: выводит информационное сообщение о том, что изображение успешно сохранено. Это диалоговое окно содержит заголовок "Успех" и текст сообщения "Изображение успешно сохранено!", информируя пользователя об успешном завершении операции.

Таким образом, функция `save_image` обеспечивает интерфейс для сохранения изображения с возможностью выбора места сохранения через графический интерфейс пользователя и информирует пользователя о результате сохранения с помощью всплывающего сообщения.

Функция `select_image_to_embed` предназначена для выбора изображения, в которое будет встроено сообщение. Она использует диалоговое окно для выбора файла, открывает выбранное изображение, считывает сообщение из текстового поля и запускает процесс встраивания сообщения в отдельном потоке. Детальное описание работы функции:

– `filepath = filedialog.askopenfilename()`: вызывает диалоговое окно для выбора файла, позволяя пользователю выбрать изображение, в которое будет встроено сообщение. Путь к выбранному файлу сохраняется в переменной filepath;

– `if filepath:`: проверяет, был ли выбран файл. Если путь к файлу существует (то есть пользователь выбрал файл и не отменил операцию), выполнение кода продолжается;

– `img = Image.open(filepath)`: открывает выбранное изображение с использованием библиотеки PIL (Python Imaging Library). Созданный объект изображения сохраняется в переменной img;

– `message = text_entry.get("1.0", "end-1c")`: считывает текстовое сообщение, которое будет встроено в изображение, из текстового поля интерфейса. При этом применяется такой метод, как get. Данный метод применяется для того, чтобы извлечь текст из виджета `text_entry`. Причем "`1.0`" означает начало текста, а "`end-1c`" указывает на конец текста за вычетом последнего символа (обычно перевода строки);

– `progress_bar['value'] = 0`: данная строка занимается тем, что она сбрасывает значение прогресс-бара на 0, тем самым она готовит его к отображению прогресса встраивания сообщения;

– `threading.Thread(target=lambda: embed_message(img, message, update_progress_bar, save_image)).start()`: эта строка создает и запускает новый поток, который выполняет функцию `embed_message` с передачей ей изображения, сообщения, функции обновления прогресс-бара и функции сохранения изображения в качестве аргументов. В этом случае применяется специальная функция – а именно лямбда-функция. С ее помощью осуществляется передача аргументов в функцию `embed_message`. Использование многопоточности позволяет обеспечить отзывчивость интерфейса, особенно в случае длительного процесса зашифровки сообщений (иначе была бы блокировка главного потока приложения и интерфейс программы зависал бы).

Таким образом, функция `select_image_to_embed` выполняет следующие функции:

– интегрирует процесс выбора изображения, ввода сообщения и встраивания сообщения в изображение;

– обеспечивает асинхронное выполнение кодирования сообщения;

– поддерживает отзывчивость графического интерфейса пользователя.

Функция `extract_message` выполняет действия, которые направлены на извлечение скрытого текстового сообщения из изображения. Аргументы данной функции нуждаются в некоторых комментариях:

– `img` – это изображение, из которого нужно извлечь сообщение;

– `progress_callback` – функция обратного вызова для отслеживания прогресса извлечения сообщения;

– `on_complete` – функция обратного вызова, вызываемая после успешного извлечения сообщения.

Строка кода `img_array = np.array(img)` занимается тем, что преобразует изображение img в массив NumPy. Тем самым обеспечивается в дальнейшем удобный доступ к массиву пикселей изображения.

`binary_message = ""` – этот код производит действия по инициализации строки, в которой будет собрано бинарное представление сообщения.

Строка программы `total_pixels = img_array.shape[0] * img_array.shape[1]` занимается следующим:

– вычисляет общее количество пикселей в изображении;

– причем использует для этого размеры массива изображения.

Цикл `for index, (i, j) in enumerate(np.ndindex(img_array.shape[0], img_array.shape[1]))` выполняет функции:

– происходит обход всех пикселей изображения;

– генерирует индексы для доступа к каждому пикселю;

– получает текущий пиксель изображения.

Программный код `for n in range(3)` – это вложенный цикл, который проходит по каждому из трех каналов пикселя (RGB – red, green, blue, т.е. красный, зеленый и синий канал).

Код программы `binary_message += str(pixel[n] & 1)` выполняет функции:

– извлекает младший бит каждого канала пикселя;

– добавляет его к строке `binary_message`;

– формирует бинарное представление скрытого сообщения.

Программный код в виде строки `if len(binary_message) > 16 and binary_message[-16:] == '1111111111111110'` занимается специальной проверкой того факта, а не достигнута ли завершающая последовательность, ведь она будет указывать на конец сообщения.

Код `progress_callback(100)` – обновляет прогресс до 100%. Т.е. установка этого значения показывает, что процесс дешифровки завершен.

Код `on_complete(binary_to_text(binary_message))` выполняет действия:

– передает извлеченное текстовое сообщение, преобразованное из бинарной строки в текст с помощью функции `binary_to_text`;

– вызывает функцию обратного вызова `on_complete`.

`if index % 100 == 0`: регулярно обновляет прогресс извлечения через каждые 100 обработанных пикселей.

`progress = (index / total_pixels) * 100`: вычисляет текущий прогресс извлечения сообщения.

`progress_callback(progress)`: вызывает функцию обратного вызова `progress_callback` с текущим значением прогресса.

Таким образом, функция `extract_message` анализирует младшие биты пикселей изображения для восстановления скрытого бинарного сообщения и преобразует его в текстовую форму, обеспечивая возможность отслеживания прогресса и обработки завершения процесса через функции обратного вызова.

Функция `select_image_to_extract` предназначена для выбора изображения пользователем, из которого будет извлечено скрытое сообщение. Она использует графический интерфейс пользователя (GUI) для выбора файла и многопоточность для асинхронного извлечения сообщения. Вот пошаговое описание работы функции:

– выбор файла изображения: с помощью `filedialog.askopenfilename()`, функция открывает стандартное диалоговое окно выбора файла, позволяя пользователю выбрать изображение для извлечения сообщения. Выбранный путь к файлу сохраняется в переменной `filepath`;

– проверка выбора файла: если пользователь выбрал файл (`if filepath:`), выполнение кода продолжается. В противном случае (если диалог был закрыт без выбора файла), выполнение функции завершается;

– открытие изображения: с помощью `Image.open(filepath)`, функция открывает изображение по указанному пути. Этот объект изображения (img) будет использоваться для извлечения скрытого сообщения;

– сброс прогресс-бара: значение прогресс-бара устанавливается в 0 (`progress_bar['value'] = 0`), чтобы инициализировать или сбросить отображение прогресса операции извлечения сообщения;

– запуск процесса извлечения в отдельном потоке: для предотвращения блокировки или зависания GUI во время выполнения потенциально длительной операции извлечения сообщения используется многопоточность. Создается и запускается новый поток с помощью `threading.Thread`, в который передается функция `extract_message` с необходимыми аргументами: объект изображения img, функция обратного вызова для обновления прогресс-бара `update_progress_bar`, и лямбда-функция, вызывающаяся по завершению извлечения и отображающая извлеченное сообщение в диалоговом окне с помощью `messagebox.showinfo`.

Использование лямбда-функции в аргументе `target` позволяет передать дополнительные аргументы в функцию `extract_message`, обеспечивая таким образом гибкость в определении поведения по завершению операции. Такой подход позволяет асинхронно извлекать сообщение из изображения, обновлять прогресс выполнения в прогресс-баре и отображать результат пользователю, не нарушая отзывчивость интерфейса.

После всех функций идут строки:

– `root = tk.Tk()`: создает корневое окно приложения, которое служит основой для всех остальных элементов GUI;

– `root.title("Стеганография в изображениях")`: устанавливает заголовок корневого окна;

– `embed_frame = tk.Frame(root)`: создает фрейм (Frame) внутри корневого окна для группировки виджетов. Фреймы используются для улучшения организации элементов интерфейса;

– `embed_frame.pack(padx=10, pady=10)`: размещает фрейм в корневом окне, задавая отступы по горизонтали (padx) и вертикали (pady);

– `text_entry = tk.Text(embed_frame, height=10, width=50)`: создает текстовое поле (Text) для ввода сообщения, которое будет встроено в изображение. Размер текстового поля задается параметрами height и width;

– `text_entry.pack()`: размещает текстовое поле внутри фрейма;

– `embed_button = tk.Button(embed_frame, text="Выбрать изображение и встроить сообщение", command=select_image_to_embed)`: создает кнопку (Button), которая при нажатии вызывает функцию select_image_to_embed для выбора изображения и встраивания в него текстового сообщения;

– `embed_button.pack(pady=10)`: размещает кнопку встраивания сообщения в фрейме, задавая отступ по вертикали;

– `extract_button = tk.Button(embed_frame, text="Выбрать изображение и извлечь сообщение", command=select_image_to_extract)`: создает вторую кнопку для выбора изображения и извлечения из него текстового сообщения. Функция select_image_to_extract вызывается при нажатии на кнопку;

– `extract_button.pack(pady=10)`: размещает кнопку извлечения сообщения в фрейме, аналогично кнопке встраивания;

– `progress_bar = ttk.Progressbar(embed_frame, orient="horizontal", length=400, mode="determinate")`: создает прогресс-бар (Progressbar) для отображения прогресса операций встраивания или извлечения сообщения. Прогресс-бар имеет горизонтальную ориентацию, фиксированную длину и детерминированный режим;

– `progress_bar.pack(pady=10)`: размещает прогресс-бар в фрейме;

– `root.mainloop()`: запускает главный цикл обработки событий Tkinter, который обеспечивает отображение окна приложения и реагирование на действия пользователя.

Этот код демонстрирует создание простого, но функционального интерфейса для стеганографического приложения, позволяя пользователю

взаимодействовать с программой через графический интерфейс для встраивания и извлечения сообщений в изображениях.

3.3 Демонстрация работы подсистемы

Для запуска приложения необходимо запустить файл app.exe. При этом отобразится главное окно приложения, показанное на рисунке 8.

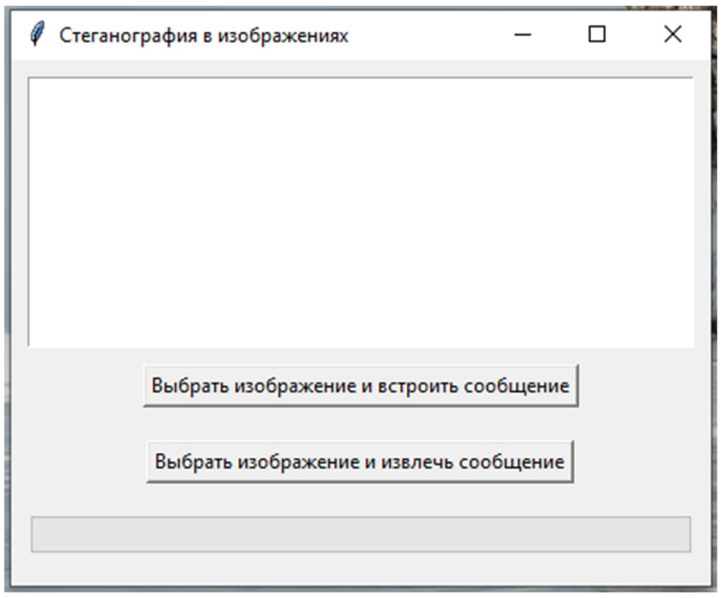

Рисунок 8 – Главное окно приложения

Далее можно ввести текст сообщения в отведенное для этого окно ввода, показанное на рисунке 9.

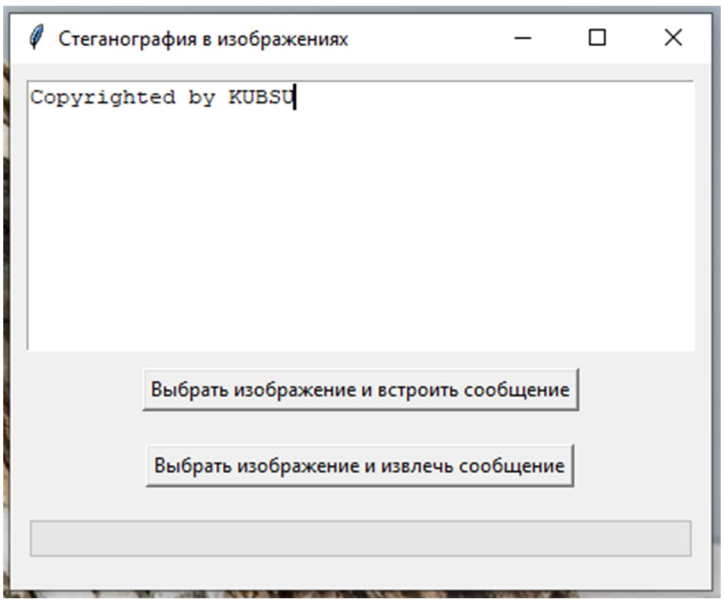

Рисунок 9 – Ввод текста

Далее необходимо нажать кнопку «Выбрать изображение и встроить сообщение». При этом откроется диалоговое окно выбора файла (рисунок 10).

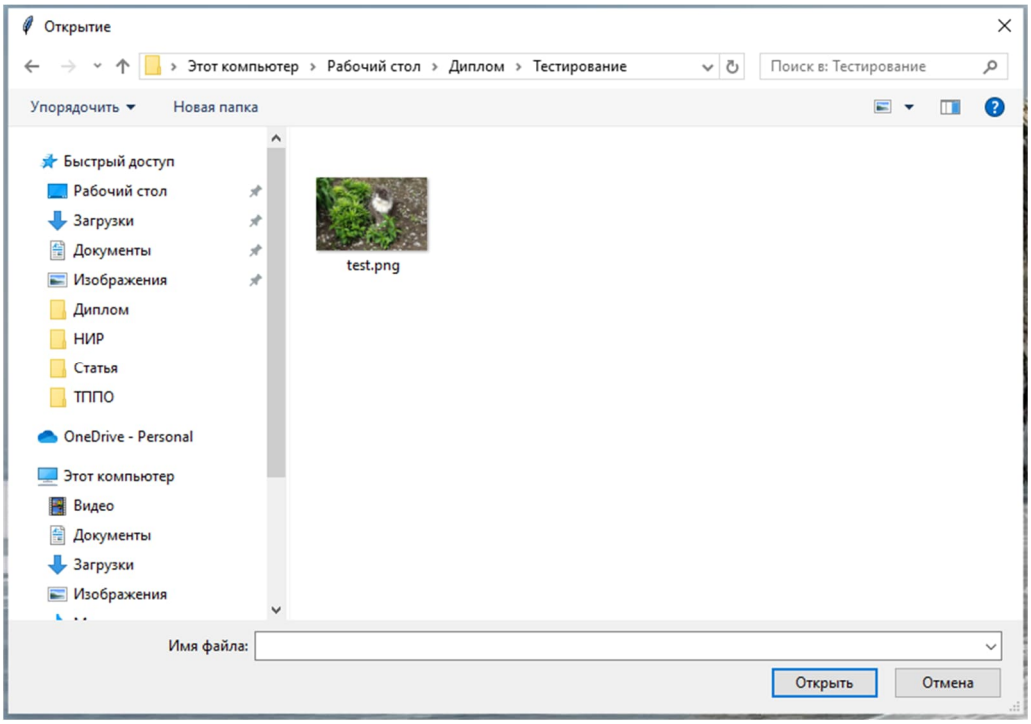

Рисунок 10 – Выбор изображения

Выбрано изображение test.png, как показано на рисунке 11. На нем изображен кот Кузьма.

Рисунок 11 – Тестовое изображение

После этого откроется диалоговое окно с выбором места сохранения выходного файла, как показано на рисунке 12.

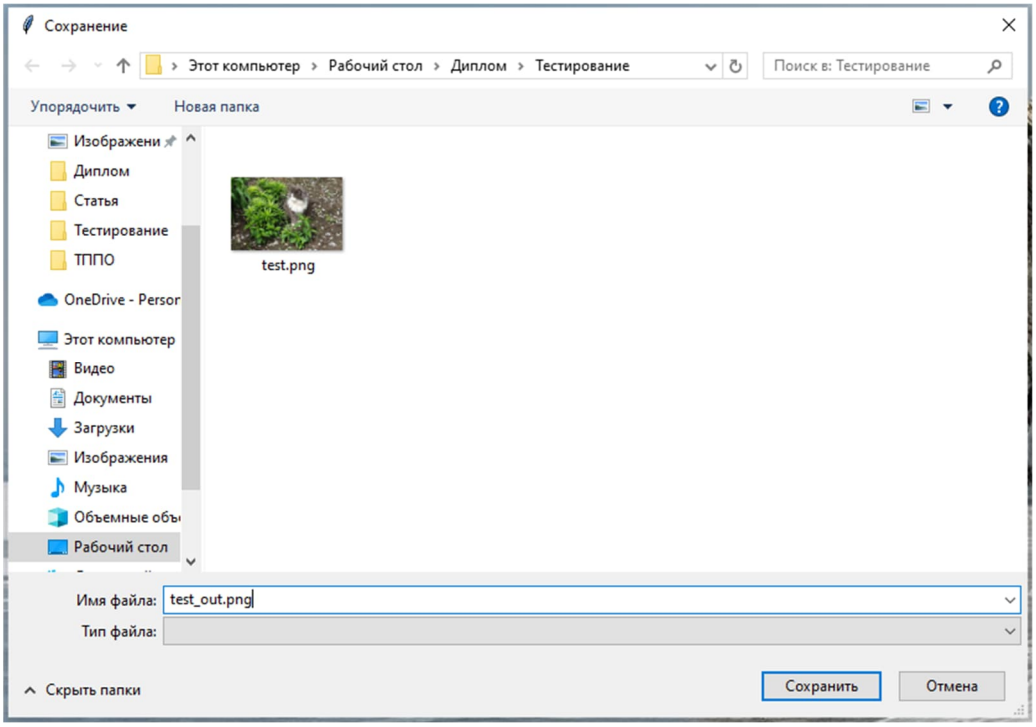

Рисунок 12 – Выбор места сохранения выходного файла

После обработки отображается сообщение об успехе, как показано на рисунке 13.

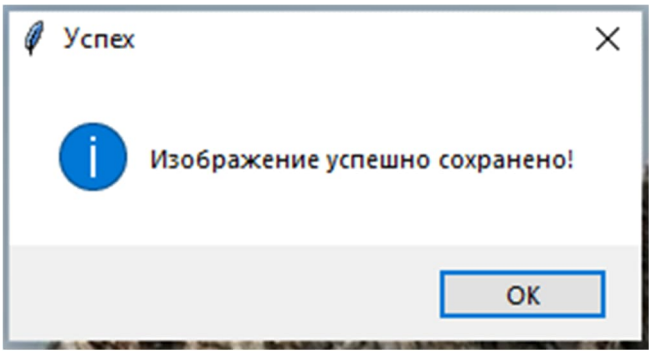

Рисунок 13 – Сообщение после обработки изображения

На рисунке 14 представлено изображение со встроенным сообщением. Можно сравнить два изображения на рисунках 11 и 14 (файлы test.png и test_out.png).

Рисунок 14 – Выходное изображение

По итогу сравнения изображений на рисунках 11 и 14 можно сделать вывод: сообщение было внедрено таким образом, что визуальные изменения невозможно обнаружить невооружённым глазом.

Для восстановления зашифрованного в изображении текстового сообщения необходимо нажать на кнопку «Выбрать изображение и извлечь сообщение». Результат показан на рисунке 15.

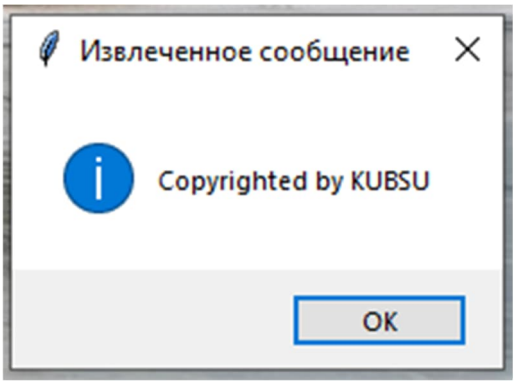

Рисунок 15 – Извлеченное сообщение

Легко видеть, что текстовое сообщение успешно прочитано из файла.

3.4 Экспериментальные исследования

Исходя из особенностей работы приложения, оно позволяет сохранять изображения только в формате png. Поэтому удобно открывать и изучать сохраненные изображения именно в этом формате.

Размер (количество знаков) текстового сообщения, которое скрывается, может быть достаточно большим. Можно провести тестирование на большом объеме текста. Пусть тестовым образцом будет весь программный код подсистемы. Размер сообщения равен размеру файла app.py (5136 байт).

Внедрение показано на рисунке 16. Изображение будет то же, что в подразделе 3.3.

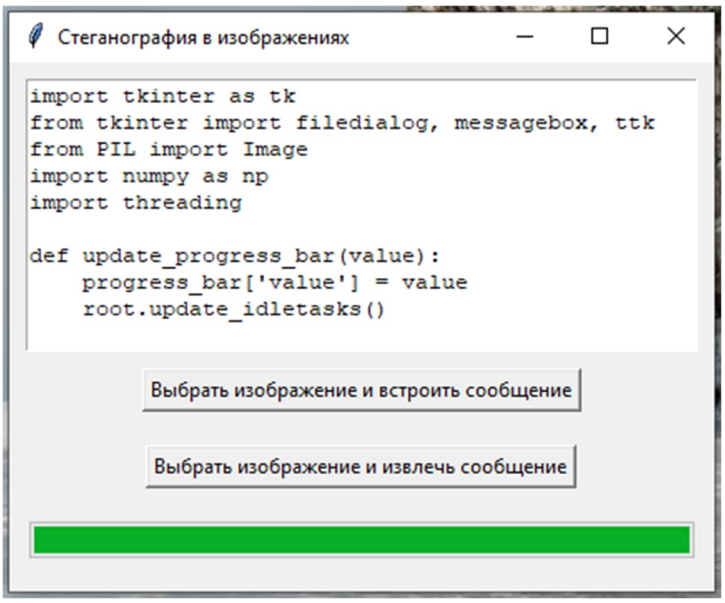

Рисунок 16 – Внедрение длинного текста

Можно сравнить два изображения (файлы test.png и test_out.png), как показано на рисунке 17.

Размер исходного файла изображения составил 369055 байт.

Размер выходного файла изображения составил 374512 байт.

Разница, на которую увеличился объем файла, составила

374512-369055 = 5457 байт.

а

б

Рисунок 17 – Сравнение изображений (а – входное, б – выходное)

Сравнение изображений (сверху – входное, снизу – выходное). Видно, что визуальных отличий никаких не наблюдается.

Таким образом, даже при сокрытии достаточно крупных текстовых сообщений проблем с отображением изображения не наблюдается, что подтверждает высокую степень пригодности данного алгоритма. Стеганографический метод позволяет скрывать текстовые сообщения большого размера без риска чтения зашифрованного послания, т.к.

злоумышленник даже не заподозрит, что изображение содержит шифрованную информацию.

Каждый бит текста сохраняется в младший значащий бит (LSB) пикселя изображения. В данном случае изменяется только один бит на канал в каждом пикселе, поэтому зависимость между длиной текста и размерами изображения будет следующей:

Допустим, имеется изображение с размерами $W \times H$ (ширина W, высота H). У каждого пикселя есть три цветовых канала (RGB), в каждом из которых можно изменить один бит для внедрения данных.

Если сохранять один бит информации на каждый канал пикселя, то максимальное количество бит, которое можно внедрить в изображение, будет:

$$\textit{Максимальное количество бит} = W * H * 3 \quad (5)$$

Поскольку в данном случае каждый символ в UTF-8 занимает 2 байта, что равно 16 битам, максимальное количество символов, которое можно внедрить в изображение, будет:

$$\textit{Максимальное количество символов} = \frac{W * H * 3}{16} \quad (6)$$

Расчёты максимального количества символов для изображений разного размера представлены в таблице 3.

Таблица 3 – Расчёт максимального количества символов

Длина*Ширина	Максимальное количество символов
0*0	0
50*50	468
100*100	1875
150*150	4218
200*200	7500
250*250	11718

Продолжение таблицы 3

Длина*Ширина	Максимальное количество символов
300*300	16875
350*350	22968
400*400	30000
450*450	37968
500*500	46875
550*550	56718
600*600	67500
650*650	79218
700*700	91875
750*750	105468
800*800	120000
850*850	135468
900*900	151875
950*950	169218
1000*1000	187500
1050*1050	206718
1100*1100	226875
1150*1150	247968
1200*1200	270000
1250*1250	292968
1300*1300	316875
1350*1350	341718
1400*1400	367500
1450*1450	394218
1500*1500	421875
1550*1550	450468
1600*1600	480000
1650*1650	510468
1700*1700	541875
1750*1750	574218
1800*1800	607500
1850*1850	641718
1900*1900	676875
1950*1950	712968
2000*2000	750000
2050*2050	787968
2100*2100	826875
2150*2150	866718
2200*2200	907500
2250*2250	949218
2300*2300	991875
2350*2350	1035468
2400*2400	1080000
2450*2450	1125468
2500*2500	1171875

Продолжение таблицы 3

Длина*Ширина	Максимальное количество символов
2550*2550	1219218
2600*2600	1267500
2650*2650	1316718
2700*2700	1366875
2750*2750	1417968
2800*2800	1470000
2850*2850	1522968
2900*2900	1576875
2950*2950	1631718
3000*3000	1687500
3050*3050	1744218
3100*3100	1801875
3150*3150	1860468
3200*3200	1920000
3250*3250	1980468
3300*3300	2041875
3350*3350	2104218
3400*3400	2167500
3450*3450	2231718
3500*3500	2296875

График зависимости максимального количества символов от размеров изображения показан на рисунке 18.

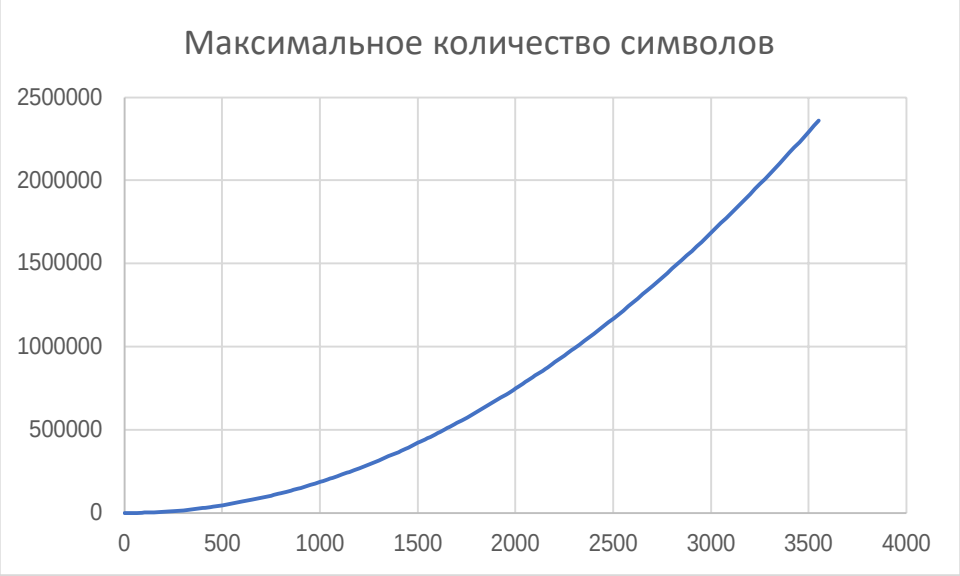

Рисунок 18 – Зависимость максимального количества символов от размеров изображения

По оси абсцисс показан размер (ширина равна длине и равна значению по оси x) изображения (т.е., по сути, это для квадратных изображений). А по оси ординат показано максимальное количество символов. Поэтому графиком является парабола. Легко видеть, что в изображении размером около 3500 на 3500 пикселей можно разместить такой объемный текст, как, например, «Война и мир» Л. Н. Толстого.

Таким образом, при известных размерах изображения можно использовать эту формулу для расчета максимальной длины текста, который можно безопасно зашифровать в изображении, не выходя за его пределы.

ЗАКЛЮЧЕНИЕ

В заключении данной выпускной квалификационной работы можно констатировать, что поставленная цель разработки и анализа подсистемы защиты графических данных, интегрирующей в себя современные методы стеганографии и криптографии для создания надежного и эффективного механизма защиты, была успешно достигнута. Основное внимание было уделено изучению возможностей скрытого внедрения идентификационных данных в графические объекты, не ухудшая их визуальных характеристик, и обеспечению возможности их последующей верификации.

В результате выполнения выпускной квалификационной работы все поставленные задачи выполнены, а именно:

– проанализированы современные методы защиты графических данных, рассмотрена их классификация, что позволило оценить текущее состояние сферы информационной безопасности и выделить наиболее перспективные направления для дальнейшего исследования;

– исследованы принципы работы и методы внедрения скрытого идентификационного слоя в графические объекты, что позволило определить оптимальные подходы для интеграции защитных механизмов, не влияющих на первоначальное качество и восприятие изображений;

– разработан алгоритм и программное обеспечение для реализации подсистемы защиты графических данных, с применением современных технологий программирования и математического моделирования, что обеспечило гибкость и модульность предложенной подсистемы;

– проведено тестирование, выполнен анализ эффективности предложенной подсистемы, которые подтвердили эффективность и высокую надежность разработанного решения в условиях реального применения. На нескольких примерах было продемонстрировано, что стеганография успешно работает и является высокоэффективной в сфере сокрытия информации;

– разработаны рекомендации по применению подсистемы защиты в различных областях для внедрения и использования разработанной системы в условиях, где защита графической информации является критически важной.

Таким образом, в результате выполнения всех поставленных задач удалось создать полноценную подсистему защиты графических данных. Данная система может быть эффективно интегрирована в существующие информационные системы для повышения их безопасности.

Стоит отметить, что результаты работы имеют важное теоретическое и прикладное значение. Дело в том, что технология стеганографии сейчас, как никогда актуальна в эпоху современных компьютерных технологий из-за необходимости защиты цифровой информации, поскольку стеганография позволяет шифровать информацию в неочевидном виде. Злоумышленник даже может не подозревать наличие шифрованных сообщений в изображениях, несмотря на то что они там, как раз-таки будут находиться. Тем самым легко скрыть сам факт шифрования. И тем самым злоумышленник никогда не сможет получить доступ к данным. Применять данную методику можно, например, для обеспечения сохранности паролей.

В целом, данная выпускная квалификационная работа представляет собой значительный вклад в область информационной безопасности, поскольку она демонстрирует успешное сочетание теоретических исследований и практической разработки. Результаты работы могут найти применение в различных сферах, требующих защиты графических данных, и стать основой для дальнейших исследований и разработок в этой области.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ Р 50922-2006. Защита информации. Основные термины и определения : национальный стандарт Российской Федерации : издание официальное : утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 27 декабря 2006 г. № 373-ст : взамен ГОСТ Р 50922-96 : дата введения 2008–02–01 / подготовлен Федеральным государственным учреждением «Государственный научно-исследовательский испытательный институт проблем технической защиты информации Федеральной службы по техническому и экспортному контролю» // Электронный фонд правовых и нормативно-технических документов : [сайт]. – URL: https://docs.cntd.ru/document/1200058320 (дата обращения: 02.04.2024).

2. Ефремов, А. Понятие и виды конфиденциальной информации / А. Ефремов // Право и Интернет : [сайт]. – 2000. – URL: https://www.russianlaw.net/law/confidential_data/a90/ (дата обращения: 03.04.2024).

3. Аутентификация. Теория и практика обеспечения безопасного доступа к информационным ресурсам : учебное пособие / А. А. Афанасьев, Л. Т. Веденьев, А. А. Воронцов, Э. Р. Газизова ; под редакцией А. А. Шелупанова [и др.]. – 2-е изд., стер. – Москва : Горячая линия-Телеком, 2012. – 550 с. – URL: https://e.lanbook.com/book/5114 (дата обращения: 03.04.2024). – ISBN 978-5-9912-0257-2.

4. Стеганографические системы. Критерии и методическое обеспечение : учебно-методическое пособие / В. Г. Грибунин, В. Е. Костюков, А. П. Мартынов [и др.] ; под редакцией В. Г. Грибунина. – Саров : РФЯЦ-ВНИИЭФ, 2016. – 324 с. – URL: https://e.lanbook.com/book/243479 (дата обращения: 05.04.2024). – ISBN 978-5-9515-0317-6.

5. Стеганографические системы. Цифровые водяные знаки : учебно-методическое пособие / В. Г. Грибунин, В. Е. Костюков, А. П. Мартынов

[и др.] ; под редакцией В. Г. Грибунина. – Саров : РФЯЦ-ВНИИЭФ, 2016. – 210 с. – URL: https://e.lanbook.com/book/243482 (дата обращения: 07.04.2024). – ISBN 978-5-9515-0330-5.

6. Криптография и безопасность цифровых систем : учебное пособие / В. Г. Грибунин, А. П. Мартынов, Д. Б. Николаев, В. Н. Фомченко ; под редакцией А. И. Астайкина. – Саров : РФЯЦ-ВНИИЭФ, 2011. – 411 с. – URL: https://e.lanbook.com/book/243461 (дата обращения: 09.04.2024). – ISBN 978-5-9515-0166-0.

7. Васильев, А. Программирование на C# для начинающих. Основные сведения / А. Васильев. – Москва : Эксмо, 2018. – 592 с. – ISBN 978-5-04-092519-3.

8. Никитина, Т. П. Программирование. Основы Python / Т. П. Никитина, Л. В. Королев. – Санкт-Петербург : Лань, 2023. – 156 с. – URL: https://e.lanbook.com/book/302714 (дата обращения: 11.04.2024). – ISBN 978-5-507-45283-5.

9. Kumawat, G. Learn PHP Website Backend Development: Shaping the Future of Dynamic Websites / G. Kumawat. – Independently published, 2023. – 139 p. – ISBN 979-8872903406.

10. Gregoire, M. Professional C++, 6th Edition / M. Gregoire. – New Jersey : John Wiley & Sons, Inc., 2024. – 1376 p. – ISBN 978-1-394-19318-9.

11. Курбатова, И. В. Основы программирования на языке Java : учебное пособие для вузов / И. В. Курбатова, А. В. Печкуров. – Санкт-Петербург : Лань, 2024. – 348 с. – URL: https://e.lanbook.com/book/385928 (дата обращения: 11.04.2024). – ISBN 978-5-507-48515-4.

12. Das, S. Steganography and Steganalysis: Different Approaches / S. Das, B. Bandyopadhyay, S. Sanyal // arXiv : [сайт]. – 2011. – URL: https://arxiv.org/abs/1111.3758 (дата обращения: 05.04.2024).

13. Николаева, И. А. Исследование цифровой информации методами стеганографии : учебное пособие / И. А. Николаева, И. А. Мартынова, Э. В. Запонов. – Саров : РФЯЦ-ВНИИЭФ, 2021. – 215 с. – URL:

https://e.lanbook.com/book/243461 (дата обращения: 10.04.2024). – ISBN 978-5-9515-0500-2.

14. Вейцман, В. М. Проектирование информационных систем : учебное пособие для вузов / В. М. Вейцман. – 2-е изд., стер. – Санкт-Петербург : Лань, 2022. – 316 с. – URL: https://e.lanbook.com/book/208946 (дата обращения: 15.04.2024). – ISBN 978-5-8114-9982-3.

15. Гвоздева, Т. В. Проектирование информационных систем. Методы и средства структурно-функционального проектирования. Практикум : учебное пособие для спо / Т. В. Гвоздева, Б. А. Баллод. – 3-е изд., стер. – Санкт-Петербург : Лань, 2024. – 148 с. – URL: https://e.lanbook.com/book/388976 (дата обращения: 15.04.2024). – ISBN 978-5-507-47555-1.

16. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. А. Максимчук, М. У. Энгл [и др.] ; под редакцией Д. А. Илюшина. – 3-е изд : Пер. с англ – Москва : ООО «И.Д. Вильямс», 2008. – 720 с. : ил. – ISBN 978-5-8459-1401-9.

17. Bhasin, H. Python Programming Using Problem Solving / H. Bhasin. – Dulles : Mercury Learning and Information, 2023. – 601 p. – ISBN 978-1- 68392-862-1.

18. Olsen, D. Mastering Python: 50 Specific Tips for Writing Better Code / D. Olsen. – Mumbai : Ziyob Publishers, 2023. – 345 p. – ISBN 9798865196815.

ПРИЛОЖЕНИЕ

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
from PIL import Image
import numpy as np
import threading

def update_progress_bar(value):
    progress_bar['value'] = value
    root.update_idletasks()

def text_to_binary(message):
    binary_message = message.encode('utf-8')
    message_length = len(binary_message)
    message_length_binary = format(message_length, '016b')
    binary_data = ''.join(format(byte, '08b') for byte in
binary_message)
    return message_length_binary + binary_data +
'1111111111111110'

def binary_to_text(binary_message):
    message_length = int(binary_message[:16], 2)
    binary_data = binary_message[16:16 + 8 * message_length]
    message = bytearray(int(binary_data[i:i+8], 2) for i in
range(0, len(binary_data), 8)).decode('utf-8')
    return message

def embed_message(img, message, progress_callback, on_complete):
    binary_message = text_to_binary(message)
    img_array = np.array(img)
    total_pixels = img_array.shape[0] * img_array.shape[1]
    data_index = 0
```

```
        for index, (i, j) in enumerate(np.ndindex(img_array.shape[0],
img_array.shape[1])):
            pixel = img_array[i, j]
            for n in range(3):
                if data_index < len(binary_message):
                    pixel[n]     =     pixel[n]     &     ~1     |
int(binary_message[data_index])
                    data_index += 1
                if data_index == len(binary_message):
                    progress_callback(100)
                    on_complete(Image.fromarray(img_array))
                    return
            if index % 100 == 0:
                progress = (index / total_pixels) * 100
                progress_callback(progress)
        progress_callback(100)
        on_complete(Image.fromarray(img_array))

    def save_image(img):  # Определение функции save_image
        file_path                                                    =
filedialog.asksaveasfilename(defaultextension=".png")
        if file_path:
            img.save(file_path)
            messagebox.showinfo("Успех",     "Изображение     успешно
сохранено!")

    def select_image_to_embed():
        filepath = filedialog.askopenfilename()
        if filepath:
            img = Image.open(filepath)
            message = text_entry.get("1.0", "end-1c")
            progress_bar['value'] = 0
            threading.Thread(target=lambda:        embed_message(img,
message, update_progress_bar, save_image)).start()
```

```
    def extract_message(img, progress_callback, on_complete):
        img_array = np.array(img)
        binary_message = ""
        total_pixels = img_array.shape[0] * img_array.shape[1]
        for index, (i, j) in enumerate(np.ndindex(img_array.shape[0],
img_array.shape[1])):
            pixel = img_array[i, j]
            for n in range(3):
                binary_message += str(pixel[n] & 1)
            if len(binary_message) > 16 and binary_message[-16:] ==
'1111111111111110':
                progress_callback(100)
                on_complete(binary_to_text(binary_message))
                return
            if index % 100 == 0:
                progress = (index / total_pixels) * 100
                progress_callback(progress)

    def select_image_to_extract():
        filepath = filedialog.askopenfilename()
        if filepath:
            img = Image.open(filepath)
            progress_bar['value'] = 0
            threading.Thread(target=lambda:      extract_message(img,
update_progress_bar, lambda message: messagebox.showinfo("Извлеченное
сообщение", message))).start()

    root = tk.Tk()
    root.title("Стеганография в изображениях")
    embed_frame = tk.Frame(root)
    embed_frame.pack(padx=10, pady=10)
    text_entry = tk.Text(embed_frame, height=10, width=50)
    text_entry.pack()
```

```
    embed_button = tk.Button(embed_frame, text="Выбрать изображение и встроить сообщение", command=select_image_to_embed)
    embed_button.pack(pady=10)
    extract_button = tk.Button(embed_frame, text="Выбрать изображение и извлечь сообщение", command=select_image_to_extract)
    extract_button.pack(pady=10)
    progress_bar = ttk.Progressbar(embed_frame, orient="horizontal", length=400, mode="determinate")
    progress_bar.pack(pady=10)
    root.mainloop()
```