

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

**КУРСОВАЯ РАБОТА**

Тема: «ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АРХИВАТОРА ДАННЫХ»

Дисциплина: «Теория информации»

Специальность: 10.05.01 Компьютерная безопасность

Специализация: Математические методы защит информации

Обозначение курсовой работы ТИ.990000.000 Группа ВКБ32

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.П. Ковалев

подпись, дата

Курсовая работа защищена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. преподаватель, И.А. Алферова

подпись, дата

Ростов-на-Дону

2025



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

**ЗАДАНИЕ**

на выполнение курсовой работы

Тема «ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АРХИВАТОРА ДАННЫХ»

Дисциплина: Теория информации

Обучающийся: Ковалев Данил Петрович

Обозначение курсовой работы ТИ.990000.000 Группа: ВКБ32

Срок представления работы к защите «\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_ г.

Исходные данные для курсовой работы:

1. Задание на выполнение курсовой работы

2. Горев, А И; Симаков А.А Обеспечение Информационной Безопасности / А Горев А.И; Симаков А. – Москва: Мир, 2005. – 844 c.

3. Герман, О. Н. Теоретико-числовые методы в криптографии / О.Н. Герман, Ю.В. Нестеренко. – М.: Академия, 2012. – 272 c.

|  |
| --- |
| **Содержание пояснительной записки** |
| Введение:  Описывается появление самых первых алгоритмов сжатия, а также актуальность использования этих методов. |
| Разделы основной части: |
| 1. В разделе "Обзор gzip" дается полное описание утилиты, включая основные принципы, алгоритмы и улучшенная версия – “pigz”, механизмы сжатия, а также применение и особенности использования. 2. В разделе “Обзор xz” дается полное описание алгоритма, включая основные принципы, механизмы сжатия, а также применение и особенности использования. 3. В разделе “Обзор модификаций LZ77: fastlz, lzf, lzjb, lzss” дается полное описание алгоритмов, включая основные принципы, механизмы сжатия, а также применение и особенности использования. 4. В разделе "Программная реализация архиватора" дается обоснование выбора языка программирования и среды разработки, выбора базы данных, S3 хранилища, архитектуры приложения, описываются основные методы и классы программы, а также показывается, как выглядит программное средство. 5. В разделе "Сравнительный анализ алгоритмов сжатия" проводится сравнительный анализ данных алгоритмов, демонстрируется их работоспособность, входные и выходные данные, а также рассматриваются преимущества и недостатки каждого алгоритма. |
| Заключение:  В рамках данной курсовой работы было разработано программное средство – архиватор.   |  |  |  |  | | --- | --- | --- | --- | | Руководитель работы | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | И.А. Алферова | |  |  |  | | Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Д.П. Ковалев | |

**Содержание**

[Введение 5](#_Toc167968069)

[1 Обзор криптосистемы Нидеррайтера 6](#_Toc167968070)

[1.1 Принцип работы и описание криптосистемы 6](#_Toc167968071)

[1.2 Математическая основа 7](#_Toc167968072)

[1.3 Достоинства и недостатки 8](#_Toc167968073)

[2 Обзор криптосистемы Мак-Элиса 10](#_Toc167968074)

[2.1 Принцип работы и описание криптосистемы 10](#_Toc167968075)

[2.2 Математическая основа 11](#_Toc167968076)

[2.3 Достоинства и недостатки 12](#_Toc167968077)

[3 Программная реализация криптосистемы Нидеррайтера и Мак-Элиса 14](#_Toc167968078)

[3.1 Выбор языка программирования и инструментов 14](#_Toc167968079)

[3.2 Реализация алгоритмов шифрования и дешифрования криптосистемы Нидеррайтера 14](#_Toc167968080)

[3.3 Реализация алгоритмов шифрования и дешифрования криптосистемы Мак-Элиса……………………………………………………………………...17](#_Toc167968081)

[4 Сравнительный анализ криптосистем 20](#_Toc167968082)

[4.1 Сравнение эффективности шифрования и дешифрования криптосистемы Мак-Элиса и Нидеррайтера 20](#_Toc167968083)

[4.2 Оценка устойчивости к атакам и вычислительным методам 21](#_Toc167968084)

[Заключение 24](#_Toc167968085)

[Перечень используемых информационных ресурсов 25](#_Toc167968086)

[Приложение А Листинг кода 26](#_Toc167968087)

Введение

В современном информационном обществе объемы данных растут с каждым днем, что делает эффективное управление и хранение информации одной из ключевых задач. Архивирование данных представляет собой важный инструмент, позволяющий уменьшить занимаемое пространство и упростить процесс передачи информации. В этом контексте разработка архиватора с интерфейсом командой строки (CLI) становится актуальной задачей, позволяющей таким специалистам, как DevOps эффективно сжимать такие данные: дамп базы данных, документы и т.п.

Целью данной курсовой работы является создание программного обеспечения для архивирования данных, которое будет реализовывать алгоритмы сжатия и обеспечивать удобный интерфейс для взаимодействия с потенциальным инженером.

В первой части будет представлен обзор теоретических основ сжатия информации, включая алгоритмы: LZ77 и его модификации, gzip, bzip2, xz. Далее будет описан процесс разработки архиватора, его архитектура, включая реализацию алгоритмов сжатия и распаковки. Также будет обоснован выбор базы данных для тестирования приложения, выбор S3 хранилища, а также технология Docker.

В рамках данной работы ставятся следующие цели:

1. Изучить теоретические основы алгоритмов fastlz, lzss, lzf, lzjb, gzip, bzip2, xz включая их математические принципы и особенности функционирования.
2. Разработать программное обеспечение для архиватора, включая алгоритмы сжатия, разжатия данных.
3. Провести сравнительный анализ эффективности на основе результатов программной реализации.
4. Выявить преимущества и недостатки каждого из алгоритмов с целью определения их пригодности для конкретных задач в различных областях применения.

1 Обзор утилиты gzip

1.1 Принцип работы и описание

gzip (GNU zip) – это утилита для сжатия и восстановления данных, использующая алгоритм Deflate. Она широко применяется для сжатия интернет – трафика и является стандартом для сжатия данных в ряде UNIX – систем. Как уже упоминалось, gzip основан на алгоритме Deflate – это алгоритм сжатия без потерь, который сочетает в себе методы LZ77 и кодирования Хаффмана. Утилита была разработана Жан – Лу Гайи и Марком Адлером, первая версия 0.1 была выпущена 31 октября 1992 года, а версия 1.0 – в феврале 1993 года.

Говоря проще, gzip – это практическое применение алгоритма Deflate. Опишем его процесс работы. Процесс сжатия начинается с анализа входных данных для выявления повторяющихся последовательностей. Алгоритм LZ77 заменяет эти повторяющиеся фрагменты ссылками на их предыдущие вхождения, что позволяет значительно сократить объем данных. Каждая ссылка состоит из пары значений: смещение (offset) и длины (length), которые указывают на позицию и длину повторяющейся последовательности в обработанных данных.

После применения алгоритма LZ77, полученные данные передаются на этап кодирования Хаффмана. Этот метод создает переменные длины кода для символов, основываясь на их частоте появления: более частые символы получают более короткие коды, а реже встречающиеся – более длинные. Это позволяет дополнительно уменьшить размер данных.

При распаковке данных процесс происходит в обратном порядке. Сначала декодируются коды Хаффмана, восстанавливая последовательности, а затем алгоритм LZ77 использует ссылки для восстановления оригинальных данных. Таким образом, алгоритм Deflate обеспечивает эффективное сжатие и восстановление данных, что делает его идеальным для использования в утилите gzip.

1.2 Достоинства и недостатки

Достоинства:

* + - 1. Эффективность сжатия: gunzip обеспечивает достаточно высокую степень сжатия.

1. Скорость работы: Утилита демонстрирует хорошую производительность как при сжатии, так и при распаковке данных. Алгоритм Deflate оптимизирован для быстрого выполнения операций, что делает gunzip подходящим для использования в реальном времени.
2. Отсутствие потерь: gunzip использует алгоритм сжатия без потерь, что означает, что оригинальные данные могут быть полностью восстановлены после распаковки. Это критически важно для многих приложений, где сохранение целостности данных является приоритетом.
3. Широкая поддержка: gunzip мало того, что является стандартной утилитой в Unix – подобных системах, но он также поддерживается в большинстве браузеров. Данная утилита активно применяется в Web серверах по типу Nginx.

Недостатки:

1. Неэффективность при малых файлах: при сжатии небольших файлов эффективность gunzip может быть ниже, чем при работе с большими объемами данных. Это связано с накладными расходными на обработку и метаданные.
2. Отсутствие встроенной защиты: gunzip не предоставляет никаких средств для шифрования или защиты данных, что может быть проблемой в ситуациях, где требуется безопасность информации.
   1. Потенциальные улучшения.

Одно из возможных улучшений – это использование многопоточности. Такая реализация существует и называется – pigz – сокращение от Parallel Implementation of gzip (параллельная реализация gzip), способная использовать несколько процессоров (нескольких ядер) при сжатии данных.

2 Обзор утилиты xz

2.1 Принцип работы и описание

xz – это утилита для сжатия и восстановления данных, использующая алгоритм LZMA (Lempel-Ziv-Markov chain algorithm). Она широко применяется для сжатия файлов в UNIX – подобных системах и становится все более популярной благодаря своей высокой эффективности. XZ обеспечивает сжатие без потерь, что позволяет полностью восстанавливать оригинальные данные после распаковки.

Говоря проще, xz – это практическое применения алгоритма LZMA. Опишем его процесс работы. Процесс сжатия начинается с анализа входных данных для выявления повторяющихся последовательностей, аналогично алгоритму LZ77. Однако LZMA использует более сложные методы, включая контекстное моделирование, что позволяет достичь более высокой степени сжатия.

Алгоритм LZMA сочетает в себе два основных этапа: сжатие с использованием LZ77 и кодирование с помощью арифметического кодирования. На первом этапе алгоритм заменяет повторяющиеся фрагменты ссылками на их предыдущие вхождения, как и в LZ77. На втором этапе применяется арифметическое кодирование, которое создает более компактные коды для символов, основываясь на их вероятности появления. Это позволяет значительно уменьшить размер данных по сравнению с другими методами сжатия.

При распаковке данных процесс происходит в обратном порядке. Сначала выполняется декодирование с использованием арифметического кодирования, а затем алгоритм LZMA восстанавливает оригинальные данные, используя ссылки на повторяющиеся последовательности. Таким образом, алгоритм LZMA обеспечивает эффективное сжатие и восстановление данных, что делает его идеальным для использования в утилите xz.

2.3 Достоинства и недостатки

Утилита xz имеет ряд достоинств и недостатков, которые следует учитывать при её использовании:

Достоинства:

1. Высокая степень сжатия: xz обеспечивает значительно более высокую степень сжатия по сравнению с другими утилитами, такими как gzip.
2. Сжатие без потерь: утилита использует алгоритм LZMA, который гарантирует, что оригинальные данные могут быть полностью восстановлены после распаковки, что критически важно для многих приложений.
3. Гибкость настроек: xz предлагает пользователю выбирать уровни сжатия. Но опять-таки это зависит от реализации xz, так как существует множество ПО, где нет такого выбора.

Недостатки:

1. Скорость работы: хотя xz обеспечивает высокую степень сжатия, скорость сжатия и распаковки может быть ниже по сравнению с другими утилитами, как gzip, что может быть проблемой в ситуациях, требующих быстрого выполнения операций.
2. Потребление ресурсов: xz может потреблять значительное количество оперативной памяти и процессорных ресурсов, особенно при использовании высоких уровней сжатия, что может стать ограничением на системах с ограниченными ресурсами.
3. Ограниченная поддержка и распространение: несмотря на свою популярность, xz может не поддерживаться в некоторых устаревших системах или программном обеспечении, что может ограничить его использование в определенных контекстах.

В целом, утилита xz представляет собой мощный инструмент для сжатия данных, обеспечивая высокую степень сжатия и гибкость настроек. Однако её использование требует учета возможных ограничений, связанных со скоростью работы и потреблением ресурсов.

3 Обзор алгоритма FastLZ

3.1 Принцип работы и описание

FastLZ — это высокоскоростной алгоритм сжатия данных без потерь, ориентированный на минимальные задержки и высокую производительность. Он был создан **Амитэбом Мукерджи (Amitabh Mukherjee)** и впервые представлен в **2005 году**. Алгоритм распространяется под лицензией MIT, что делает его свободным для использования в открытых и коммерческих проектах.

FastLZ основан на модификации алгоритма LZ77, но с упрощениями, которые ускоряют обработку данных. Вот ключевые этапы его работы:

Данные делятся на независимые блоки. Каждый блок сжимается отдельно. Дальше идет поиск повторяющихся последовательностей: алгоритм ищет совпадения в скользящем окне (обычно 8 – 32 КБ). Для ускорения поиска используется хэш-таблица, которая отслеживает позиции ранее встреченных 3-байтовых последовательностей (хэш – ключей). При обнаружении совпадения генерируется ссылка в формате (offset, length), где offset – расстояние до начала повторяющиеся последовательности, length – длина совпадения (от 3 до 258 байт). Далее ссылки переводятся в бинарный вид в зависимости от уровня FastLZ. В ином случае, литералы остаются также в обычно виде. В результате получается последовательность байтов с литералами.

3.2 Описание работы FastLZ Level 1

В данной конфигурации обычно используется хэш-таблица размером 16384 символа элементов, но это зависит от реализации. Смещения – 8192 байт. Максимальная длина совпадения – 264 байта.

В начале входной буфер данных разбивается на блоки до 65535 байт. Создается хэш таблица размером 16384 байта, заполненная нулями. Для текущей позиции в данных читаются 3 байта. Далее хэш – функция вычисляет индекс в хэш – таблице. Если же в хэш таблице есть предыдущая позиция с тем же хэшем, то проверяется совпадение данных: сравниваются 3 байта из текущей позиции и прошлой, которая находится в хэш-таблице. Если совпадение найдено, определяется длина совпадения (минимум 3 байта).

После того, как заполнилась хэш – таблица, то начинается кодирование. Тут есть несколько моментов, что не все символы могут быть закодированы, в таком случае выделяются литералы.

Дальше у нас появляется окно с размером 4 байт каждый раз. Для вычисления хэша от последовательности в данном случае используется 3 байта.

Для реализации программных кодов вполне целесообразно будет обратить внимание на Python. Это высокоуровневый, динамически типизированный язык программирования с высоким уровнем абстракции, обладающий лаконичным синтаксисом и имеющий обширную стандартную библиотеку функций. В отличие от C++ и C#, Python обеспечивает более удобную работу со структурами данных (например, возможность динамического изменения списков) и возможность приведения переменной к новому типу (например, получение целого числа из строки в двоичном виде). Также несомненным плюсом является форматирование строк и возможность работы с байтовыми строками. Таким образом, Python является наиболее оптимальным выбором для реализации поставленной задачи.

В кодах для криптосистем использовались библиотеки NumPy и random. NumPy обеспечивает эффективные операции с матрицами и векторами, что важно для криптографических алгоритмов. Модуль random используется для генерации случайных чисел, необходимых для создания случайных матриц и добавления случайности в процесс шифрования.

3.2 Реализация алгоритмов шифрования и дешифрования криптосистемы Нидеррайтера

Код Нидеррайтера — это разновидность кодов с открытым ключом, основанная на кодах коррекции ошибок. Основные этапы включают создание проверочной матрицы, генерацию матриц перестановки и замещения, шифрование и дешифрование. Рассмотрим основные шаги алгоритма шифрования и дешифрования, которые используются в моем коде.

**Основные этапы:**

1. Создание проверочной матрицы (H)
2. Генерация матриц перестановки (P) и замещения (S)
3. Генерация публичного ключа (Hpub) и закрытого ключа (priv\_key)
4. Шифрование текста (encrypt)
5. Дешифрование текста (decrypt)

Шифрование текста:

Шифрование начинается с преобразования открытого текста в двоичное представление. Для каждого бита текста, если бит равен '0', выбирается случайный индекс из четных индексов. Если бит равен '1', выбирается случайный индекс из нечетных индексов. Далее создается вектор m с единицей на выбранном индексе. Затем вычисляется вектор c путем умножения m на транспонированную матрицу Hpub. Этот вектор c преобразуется в строку и добавляется к шифрованному тексту. Реализация показана на рисунке 1.1.

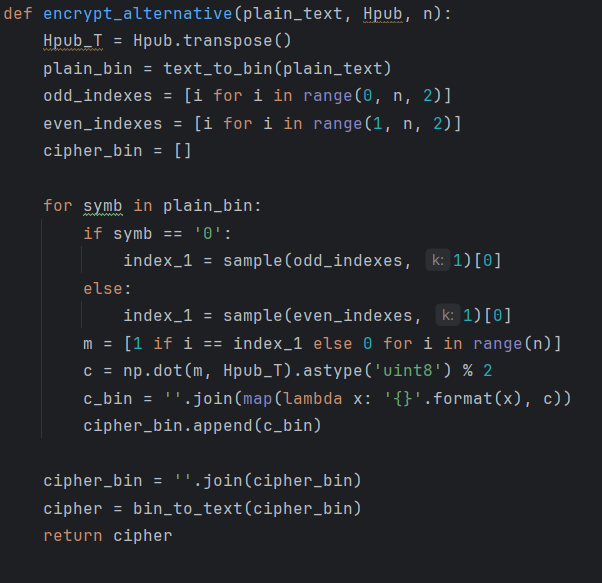


Рисунок 1.1 – Реализация Шифрования в коде Нидеррайтера

Дешифрование текста:

Дешифрование начинается с преобразования шифрованного текста в двоичное представление. Для каждого блока размера r преобразуется блок в вектор c, после чего вычисляется синдром s путем умножения c на транспонированную обратную матрицу S. Затем осуществляется поиск индекса синдрома в транспонированной проверочной матрице H\_T. Создается вектор m\_ с единицей на найденном индексе, который преобразуется с помощью обратной матрицы P\_T\_inv. Исходный бит определяется на основе индекса единицы в m. Реализация показана на рисунке 1.2.

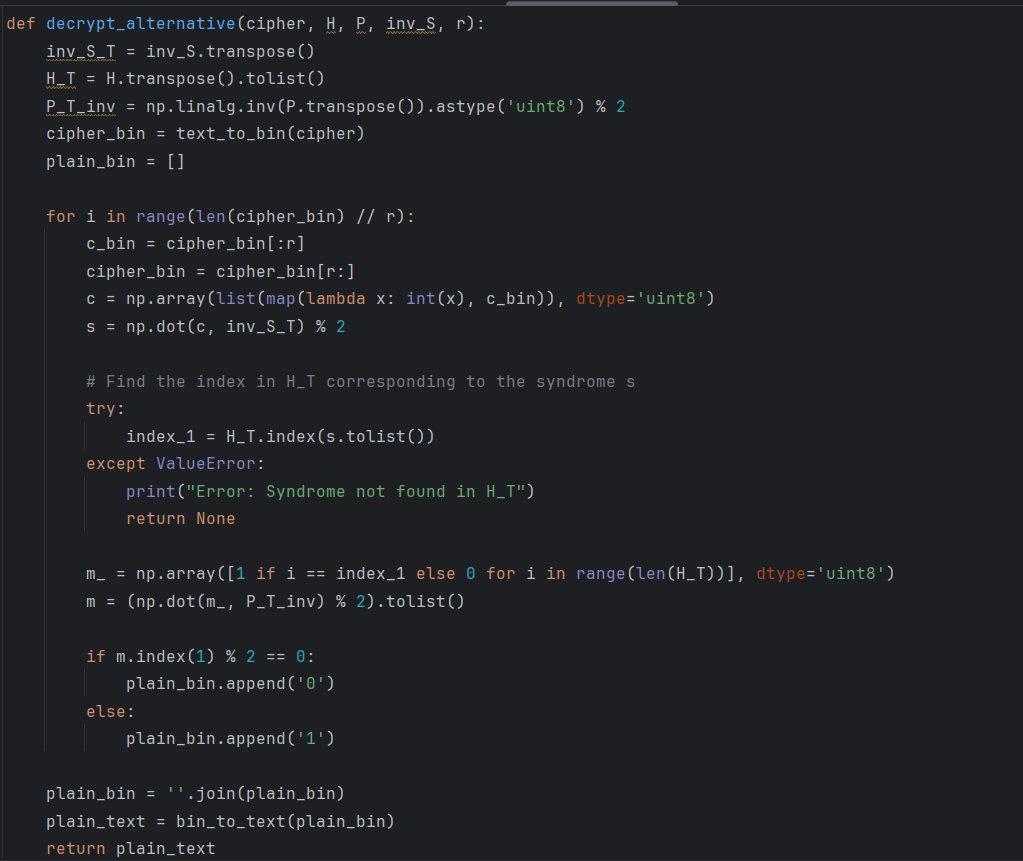


Рисунок 1.2 - Реализация дешифрования

Тестирование на тестовых данных:

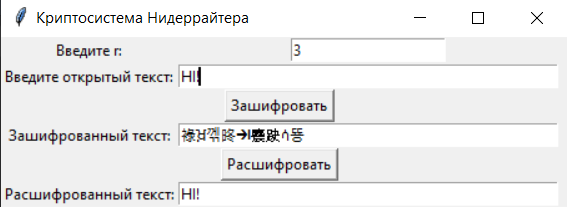


Рисунок 1.3 - Работа кода и графический интерфейс

Таким образом получилось реализовать программный код для реализации кодирования и декодирования криптосистемы Нидеррайтера, работу которого можно увидеть на рисунке 1.3.

3.3 Реализация алгоритмов шифрования и дешифрования криптосистемы Мак-Элиса

Шифрование в коде Мак-Элиса происходит следующим образом:

1. Получение параметра r: Пользователь вводит параметр r, который определяет размерность матриц для шифрования.
2. Генерация матриц: генерируются три матрицы: матрица проверки H, матрица S, и матрица перестановки P.
3. Создание открытого ключа: Открытый ключ формируется путем умножения матрицы S на систематическую матрицу Gsys​.
4. Преобразование открытого текста: Открытый текст преобразуется в двоичное представление.
5. Разбиение на блоки: Двоичная строка разбивается на блоки длиной k бит.
6. Шифрование каждого блока: для каждого блока выполняются следующие действия: генерируется случайный вектор ошибки, который складывается с произведением информационного блока на открытый ключ. Полученный вектор шифротекста добавляется к общему результату. Шифрование можно увидеть на рисунке 1.4.

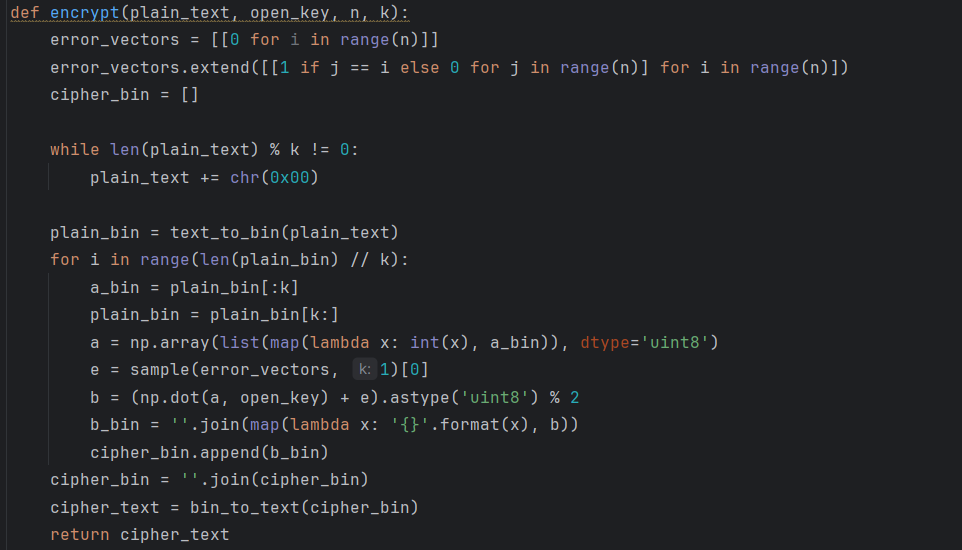


Рисунок 1.4 - Шифрование в коде Мак-Эллиса

Дешифрование в коде Мак-Элиса происходит обратным образом:

1. Получение зашифрованного текста: вводится зашифрованный текст.
2. Расшифровка: для расшифровки используется приватный ключ, который включает в себя матрицы S и P, а также систематическую матрицу Gsys. Сначала шифротекст преобразуется в двоичное представление, а затем разбивается на блоки длиной n бит. Для каждого блока выполняется дешифрование: вычисляется вектор b′ путем умножения шифроблока на обратную матрицу P, затем декодируется с помощью систематической матрицы Gsys и матрицы S. Полученные информационные блоки объединяются, и восстанавливается исходный открытый текст. Выполнение данного метода можно заметить на рисунке 1.5.

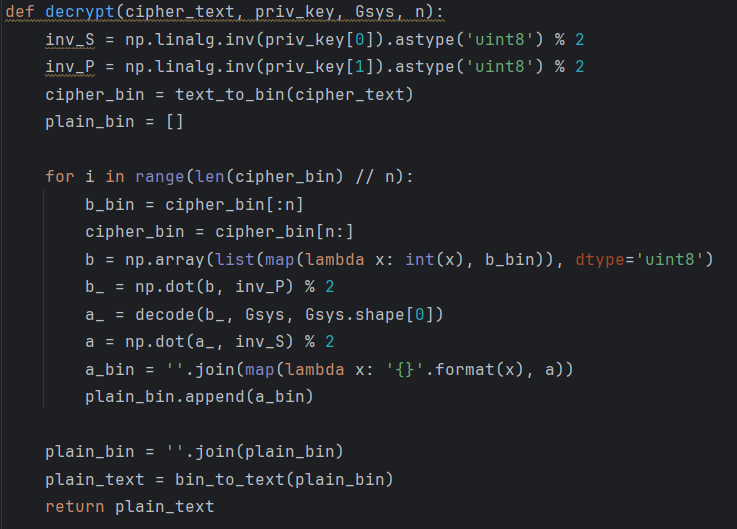


Рисунок 1.5 - Реализация дешифрования в коде криптосистемы Мак-Эллиса

Тестирование на тестовых данных:

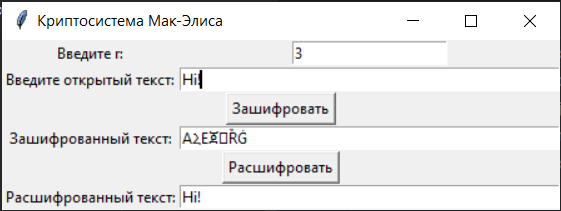


Рисунок 1.6 - Работа кода и графический интерфейс

Таким образом получилось реализовать программный код для реализации кодирования и декодирования криптосистемы Нидеррайтера, работу которого можно увидеть на рисунке 1.6.

4 Сравнительный анализ криптосистем

4.1 Сравнение эффективности шифрования и дешифрования криптосистемы Мак-Элиса и Нидеррайтера

Криптосистемы Мак-Элиса и Нидеррайтера используют линейные блоковые коды для шифрования и дешифрования сообщений, но их эффективность различается в зависимости от конкретных реализаций и параметров кодов.

В криптосистеме Мак-Элиса процесс шифрования начинается с представления сообщения в виде вектора длиной *k*. Этот вектор умножается на публичный ключ *G*′, а затем к результату добавляется вектор ошибок *e*. Шифрование требует умножения вектора на матрицу размером *k*×*n*, что делает этот процесс относительно быстрым с вычислительной сложностью порядка *O*(*kn*).

Дешифрование в криптосистеме Мак-Элиса включает умножение шифротекста на обратную перестановочную матрицу , декодирование с использованием алгоритма коррекции ошибок для исходного кода, представленного матрицей *G*, и применение обратного преобразования с использованием матрицы . Основная вычислительная нагрузка приходится на процесс декодирования кода Гоппы, который имеет сложность порядка *O*(*n*2) или больше, в зависимости от реализации алгоритма декодирования.

В криптосистеме Нидеррайтера процесс шифрования начинается с выбора сообщения в виде вектора длиной *n* с весом *t*. Этот вектор умножается на публичный ключ *H*′, что делает процесс шифрования быстрым с вычислительной сложностью порядка *O*((*n*−*k*)*n*). Дешифрование включает умножение шифротекста на обратную перестановочную матрицу , декодирование с использованием алгоритма коррекции ошибок для исходного кода с матрицей *H*, и применение обратных преобразований с использованием матриц *S* и *P*. Основная вычислительная нагрузка снова падает на декодирование кода Гоппы, с аналогичной сложностью порядка *O*(*n*2) или больше.

Обе криптосистемы имеют сопоставимую скорость шифрования, поскольку требуют умножения вектора на матрицу. В криптосистеме Мак-Элиса матрица *G*′ имеет размер *k*×*n*, тогда как в Ниддеррайтере матрица *H*′ имеет размер (*n*−*k*)×*n*. Если длина сообщения *k* значительно меньше *n*, то шифрование в Мак-Элисе может быть быстрее, хотя в большинстве случаев различия минимальны.

При дешифровании основная вычислительная нагрузка в обеих криптосистемах заключается в декодировании кода Гоппы, что делает их сложность примерно одинаковой. В криптосистеме Нидеррайтера используется матрица *H* размером (*n*−*k*)×*n*, что теоретически может повлиять на требования к памяти, но на практике это влияние незначительно.

Таким образом, обе криптосистемы предлагают эффективные методы шифрования и дешифрования с основной нагрузкой на этапе декодирования. Криптосистема Мак-Элиса может быть немного более эффективной при шифровании, если длина сообщения *k* значительно меньше *n*, однако различия в эффективности между ними обычно незначительны и зависят от конкретной реализации и параметров используемых кодов.

4.2 Оценка устойчивости к атакам и вычислительным методам

Оценка устойчивости криптосистем Мак-Элиса и Нидеррайтера к атакам и вычислительным методам является важным аспектом их применения, особенно в контексте постквантовой криптографии. Оба метода основаны на сложности проблем, связанных с декодированием линейных кодов, однако их устойчивость к различным типам атак и эффективность защиты от них могут различаться.

Устойчивость к атакам криптосистемы Мак-Элиса

Сложность декодирования: Основной принцип безопасности криптосистемы Мак-Элиса базируется на сложности задачи декодирования случайного линейного кода, которая является NP-полной задачей. Это означает, что для злоумышленника восстановление исходного сообщения без знания приватного ключа требует решения крайне сложной задачи.

Атаки на структуру: на протяжении многих лет было предложено несколько типов атак на криптосистему Мак-Элиса, которые пытаются использовать структурные особенности используемых кодов Гоппы. В частности, атаки могут быть направлены на восстановление структуры генераторной матрицы или на выявление ошибок. Однако коды Гоппы были выбраны именно за их хорошие криптографические свойства, включая стойкость к таким атакам.

Квантовые атаки: С появлением квантовых компьютеров безопасность традиционных криптосистем подвергается сомнению. Криптосистема Мак-Элиса, однако, считается устойчивой к атакам квантовых компьютеров, поскольку декодирование случайного линейного кода остаётся сложной задачей даже для квантовых алгоритмов.

Устойчивость к атакам криптосистемы Нидеррайтера

Сложность декодирования: как и в случае с Мак-Элисом, безопасность криптосистемы Нидеррайтера основана на сложности задачи декодирования линейного кода, что является NP-полной задачей. Однако в криптосистеме Нидеррайтера используется другая структура кода, что может оказывать влияние на её устойчивость к атакам.

Атаки на структуру: Криптосистема Нидеррайтера также подвержена атакам, направленным на выявление структуры порождающей матрицы и на восстановление приватного ключа. Применение кодов Гоппы в этой системе обеспечивает высокую устойчивость к таким атакам, аналогично криптосистеме Мак-Элиса.

Квантовые атаки: Криптосистема Нидеррайтера также считается устойчивой к атакам квантовых компьютеров. Задача декодирования линейного кода остаётся сложной и для квантовых алгоритмов, что делает эту систему подходящей для использования в постквантовой криптографии.

Стойкость к атакам на структуру: Обе криптосистемы используют коды Гоппы, что обеспечивает высокую устойчивость к атакам на структуру. Тем не менее, специфические атаки могут по-разному влиять на каждую из систем, что требует тщательной оценки безопасности при выборе параметров кодов.

Квантовая стойкость: Обе системы считаются квантово-устойчивыми, поскольку задача декодирования линейного кода остаётся сложной даже для квантовых компьютеров. Это делает обе криптосистемы перспективными кандидатами для использования в условиях развития квантовых вычислений.

Практическая стойкость: на практике обе криптосистемы демонстрируют высокую стойкость к современным вычислительным атакам, благодаря тщательно подобранным параметрам кодов Гоппы. Однако криптосистема Мак-Элиса имеет более долгую историю исследований и применения, что может служить дополнительным фактором доверия к её безопасности.

Таким образом, и криптосистема Мак-Элиса, и криптосистема Нидеррайтера обеспечивают высокую степень устойчивости к атакам, основываясь на сложности задачи декодирования линейных кодов. Оба метода предлагают надёжную защиту в условиях современных и будущих вычислительных угроз, включая угрозы со стороны квантовых компьютеров.

Заключение

В рамках данной работы было разработано программное средство, выполняющее сжатие и декомпрессию файлов с помощью криптосистемы Нидеррайтера и криптосистемы Мак-Элиса.

Также в ходе курсовой работы были выполнены следующие поставленные задачи:

1. Изучена литература по теме исследования.
2. Проанализирована и описана информация о криптосистемах Нидеррайтера и Мак-Элиса.
3. Изучены возможные варианты использования данных криптосистем.
4. Приведены примеры реализации кода для обеих криптосистем.
5. Реализовано программное средство для шифрования и дешифрования с использованием криптосистемы Нидеррайтера.
6. Реализовано программное средство для шифрования и дешифрования с использованием криптосистемы Мак-Элиса.
7. Проведен сравнительный анализ эффективности и устойчивости обеих криптосистем.

В результате выполнения курсовой работы была достигнута поставленная цель, заключающаяся в анализе особенностей криптосистем Нидеррайтера и Мак-Элиса, а также их программной реализации и сравнительном анализе на практике.

Перечень используемых информационных ресурсов

1. Адаменко, Михаил Основы классической криптологии. Секреты шифров и кодов / Михаил Адаменко. – Москва: Машиностроение, 2014. – 256 c.
2. Бабаш, А. В. История криптографии. Часть I / А.В. Бабаш, Г.П. Шанкин. – М.: Гелиос АРВ, 2002. – 240 c.
3. Баричев, С. Г. Основы современной криптографии / С.Г. Баричев, В.В. Гончаров, Р.Е. Серов. – Москва: СИНТЕГ, 2011. – 176 c.
4. Герман, О. Н. Теоретико-числовые методы в криптографии / О.Н. Герман, Ю.В. Нестеренко. – М.: Академия, 2012. – 272 c.
5. Горев, А И; Симаков А А Обеспечение Информационной Безопасности / А Горев А И; Симаков А. – Москва: Мир, 2005. – 844 c.
6. Деундяк В.М., Маевский А.Э., Могилевская Н.С. Методы помехоустойчивой защиты данных: учебник / В.М. Деундяк, А.Э Маевский, Н.С. Могилевская. – Ростов н/Д: Издательство Южного федерального университета, 2014. – 309 с.
7. Чижов, И.В. Пространство ключей криптосистемы Мак-Элиса – Сидельникова: автореферат / И.В. Чижов. – М.: МГУ, 2010. – 20 с.

Приложение А Листинг кода

import numpy as np  
from random import sample  
import tkinter as tk  
from tkinter import messagebox  
def get\_check\_matrix(r, n):  
H\_T = []  
for i in range(1, n + 1):  
H\_T.append([int(elem) for elem in '{:0{}b}'.format(i, r)])  
return np.array(H\_T, dtype='uint8').transpose()  
def get\_S\_matrix(dim):  
while True:  
S\_matrix = np.random.randint(0, 2, (dim, dim))  
if np.linalg.det(S\_matrix) % 2 == 1:  
return S\_matrix.astype('uint8')  
def get\_P\_matrix(n):  
P\_matrix = []  
indexes = np.random.permutation(n)  
for index in indexes:  
P\_matrix.append([1 if i == index else 0 for i in range(n)])  
return np.array(P\_matrix, dtype='uint8')  
def text\_to\_bin(text):

return ''.join(map(lambda x: '{:016b}'.format(ord(x)), text))  
def bin\_to\_text(bin):  
symbols = []  
for i in range(len(bin) // 16):  
symb = bin[:16]  
symbols.append(chr(int(symb, 2)))  
bin = bin[16:]  
return ''.join(symbols)  
def encrypt\_alternative(plain\_text, Hpub, n):  
Hpub\_T = Hpub.transpose()  
plain\_bin = text\_to\_bin(plain\_text)  
odd\_indexes = [i for i in range(0, n, 2)]  
even\_indexes = [i for i in range(1, n, 2)]  
cipher\_bin = []  
for symb in plain\_bin:  
if symb == '0':  
 index\_1 = sample(odd\_indexes, 1)[0]  
 else:  
 index\_1 = sample(even\_indexes, 1)[0]  
 m = [1 if i == index\_1 else 0 for i in range(n)]  
 c = np.dot(m, Hpub\_T).astype('uint8') % 2  
 c\_bin = ''.join(map(lambda x: '{}'.format(x), c))  
 cipher\_bin.append(c\_bin)  
  
 cipher\_bin = ''.join(cipher\_bin)  
 cipher = bin\_to\_text(cipher\_bin)

return cipher  
def decrypt\_alternative(cipher, H, P, inv\_S, r):  
 inv\_S\_T = inv\_S.transpose()  
 H\_T = H.transpose().tolist()  
 P\_T\_inv = np.linalg.inv(P.transpose()).astype('uint8') % 2  
 cipher\_bin = text\_to\_bin(cipher)  
 plain\_bin = []  
  
 for i in range(len(cipher\_bin) // r):  
 c\_bin = cipher\_bin[:r]  
 cipher\_bin = cipher\_bin[r:]  
 c = np.array(list(map(lambda x: int(x), c\_bin)), dtype='uint8')  
 s = np.dot(c, inv\_S\_T) % 2  
  
 # Find the index in H\_T corresponding to the syndrome s  
 try:  
 index\_1 = H\_T.index(s.tolist())  
 except ValueError:  
 print("Error: Syndrome not found in H\_T")  
 return None  
  
 m\_ = np.array([1 if i == index\_1 else 0 for i in range(len(H\_T))], dtype='uint8')  
 m = (np.dot(m\_, P\_T\_inv) % 2).tolist()  
  
 if m.index(1) % 2 == 0:  
 plain\_bin.append('0')  
 else:  
 plain\_bin.append('1')  
 plain\_bin = ''.join(plain\_bin)

plain\_text = bin\_to\_text(plain\_bin)  
 return plain\_text  
  
  
def encrypt\_callback():  
 try:  
 r = int(entry\_r.get())  
 n = 2 \*\* r - 1  
 k = 2 \*\* r - 1 - r  
 plain\_text = entry\_plain\_text.get()  
  
 H = get\_check\_matrix(r, n)  
 S = get\_S\_matrix(n - k)  
 P = get\_P\_matrix(n)  
 Hpub = np.dot(S, H) % 2  
 Hpub = np.dot(Hpub, P) % 2  
 inv\_P = np.linalg.inv(P).astype('uint8') % 2  
 inv\_S = np.linalg.inv(S).astype('uint8') % 2  
 priv\_key = (inv\_S, H, inv\_P)  
  
 cipher\_text = encrypt\_alternative(plain\_text, Hpub, n)  
 entry\_cipher\_text.delete(0, tk.END)  
 entry\_cipher\_text.insert(0, cipher\_text.encode('utf-8', errors='replace').decode())  
  
 global stored\_params  
 stored\_params = (r, H, P, inv\_S)  
 except Exception as e:  
 messagebox.showerror("Ошибка", str(e))  
def decrypt\_callback():  
 try:

r, H, P, inv\_S = stored\_params  
 cipher\_text = entry\_cipher\_text.get()  
 plain\_text = decrypt\_alternative(cipher\_text, H, P, inv\_S, r)  
 entry\_decrypted\_text.delete(0, tk.END)  
 entry\_decrypted\_text.insert(0, plain\_text)  
 except Exception as e:  
 messagebox.showerror("Ошибка", str(e))  
  
  
stored\_params = None  
  
root = tk.Tk()  
root.title("Криптосистема Нидеррайтера")  
  
tk.Label(root, text="Введите r:").grid(row=0, column=0)  
entry\_r = tk.Entry(root)  
entry\_r.grid(row=0, column=1)  
  
tk.Label(root, text="Введите открытый текст:").grid(row=1, column=0)  
entry\_plain\_text = tk.Entry(root, width=50)  
entry\_plain\_text.grid(row=1, column=1)  
  
tk.Button(root, text="Зашифровать", command=encrypt\_callback).grid(row=2, column=0, columnspan=2)  
  
tk.Label(root, text="Зашифрованный текст:").grid(row=3, column=0)  
entry\_cipher\_text = tk.Entry(root, width=50)  
entry\_cipher\_text.grid(row=3, column=1)  
  
tk.Button(root, text="Расшифровать", command=decrypt\_callback).grid(row=4, column=0, columnspan=2)  
  
tk.Label(root, text="Расшифрованный текст:").grid(row=5, column=0)  
entry\_decrypted\_text = tk.Entry(root, width=50)  
entry\_decrypted\_text.grid(row=5, column=1)  
  
root.mainloop()

import numpy as np  
from random import sample  
import tkinter as tk  
from tkinter import messagebox  
  
def get\_check\_matrix(r, n):  
 H\_T = []  
 for i in range(1, n + 1):  
 H\_T.append([int(elem) for elem in '{:0{}b}'.format(i, r)])  
 return np.array(H\_T, dtype='uint8').transpose()  
  
def get\_S\_matrix(dim):  
 while True:  
 S\_matrix = np.random.randint(0, 2, (dim, dim))  
 if np.linalg.det(S\_matrix) % 2 == 1:  
 return S\_matrix.astype('uint8')  
  
def get\_P\_matrix(n):  
 P\_matrix = []  
 indexes = np.random.permutation(n)  
 for index in indexes:

P\_matrix.append([1 if i == index else 0 for i in range(n)])  
 return np.array(P\_matrix, dtype='uint8')  
  
def get\_Hsys(H, n, k):  
 H\_T\_list = H.transpose().tolist()  
 for i in range(n - k):  
 H\_T\_list.append(H\_T\_list.pop(H\_T\_list.index([1 if j == i else 0 for j in range(n - k)])))  
 return np.array(H\_T\_list, dtype='uint8').transpose()  
  
def get\_Gsys(Hsys, n, k):  
 Hsys\_T\_list = Hsys.transpose().tolist()  
 Hsys\_T\_list = Hsys\_T\_list[:k]  
 P\_part = np.array(Hsys\_T\_list, dtype='uint8')  
 E\_part = np.eye(k, dtype='uint8')  
 return np.hstack((E\_part, P\_part))  
  
def text\_to\_bin(text):  
 return ''.join(map(lambda x: '{:016b}'.format(ord(x)), text))  
  
def bin\_to\_text(bin):  
 symbols = []  
 for i in range(len(bin) // 16):  
 symb = bin[:16]  
 symbols.append(chr(int(symb, 2)))  
 bin = bin[16:]  
 return ''.join(symbols)  
  
def get\_code\_words(k, G):  
 code\_words = []  
 for i in range(2\*\*k):  
 info\_word = '{:0{}b}'.format(i, k)

info\_word = tuple(map(lambda x: int(x), info\_word))  
 code\_words.append(np.dot(info\_word, G).astype('uint8') % 2)  
 return code\_words  
  
def decode(v, G, k):  
 code\_words = get\_code\_words(k, G)  
 index = 0  
 d\_min = ((v + code\_words[0]) % 2).tolist().count(1)  
 for i in range(1, len(code\_words)):  
 d = ((v + code\_words[i]) % 2).tolist().count(1)  
 if d < d\_min:  
 d\_min = d  
 index = i  
 info\_word = '{:0{}b}'.format(index, k)  
 info\_word = list(map(lambda x: int(x), info\_word))  
 return np.array(info\_word, dtype='uint8')  
  
def encrypt(plain\_text, open\_key, n, k):  
 error\_vectors = [[0 for i in range(n)]]  
 error\_vectors.extend([[1 if j == i else 0 for j in range(n)] for i in range(n)])  
 cipher\_bin = []  
  
 while len(plain\_text) % k != 0:  
 plain\_text += chr(0x00)  
  
 plain\_bin = text\_to\_bin(plain\_text)  
 for i in range(len(plain\_bin) // k):  
 a\_bin = plain\_bin[:k]  
 plain\_bin = plain\_bin[k:]  
 a = np.array(list(map(lambda x: int(x), a\_bin)), dtype='uint8')

e = sample(error\_vectors, 1)[0]  
 b = (np.dot(a, open\_key) + e).astype('uint8') % 2  
 b\_bin = ''.join(map(lambda x: '{}'.format(x), b))  
 cipher\_bin.append(b\_bin)  
 cipher\_bin = ''.join(cipher\_bin)  
 cipher\_text = bin\_to\_text(cipher\_bin)  
 return cipher\_text  
  
def decrypt(cipher\_text, priv\_key, Gsys, n):  
 inv\_S = np.linalg.inv(priv\_key[0]).astype('uint8') % 2  
 inv\_P = np.linalg.inv(priv\_key[1]).astype('uint8') % 2  
 cipher\_bin = text\_to\_bin(cipher\_text)  
 plain\_bin = []  
  
 for i in range(len(cipher\_bin) // n):  
 b\_bin = cipher\_bin[:n]  
 cipher\_bin = cipher\_bin[n:]  
 b = np.array(list(map(lambda x: int(x), b\_bin)), dtype='uint8')  
 b\_ = np.dot(b, inv\_P) % 2  
 a\_ = decode(b\_, Gsys, Gsys.shape[0])  
 a = np.dot(a\_, inv\_S) % 2  
 a\_bin = ''.join(map(lambda x: '{}'.format(x), a))  
 plain\_bin.append(a\_bin)  
  
 plain\_bin = ''.join(plain\_bin)  
 plain\_text = bin\_to\_text(plain\_bin)  
 return plain\_text  
def encrypt\_callback():  
 try:  
 r = int(entry\_r.get())  
 n = 2 \*\* r - 1  
 k = 2 \*\* r - 1 - r

plain\_text = entry\_plain\_text.get()  
  
 H = get\_check\_matrix(r, n)  
 Hsys = get\_Hsys(H, n, k)  
 Gsys = get\_Gsys(Hsys, n, k)  
 S = get\_S\_matrix(k)  
 P = get\_P\_matrix(n)  
 priv\_key = (S, P)  
 open\_key = np.dot(S, Gsys) % 2  
 np.dot(open\_key, P, out=open\_key)  
 open\_key %= 2  
  
 cipher\_text = encrypt(plain\_text, open\_key, n, k)  
 entry\_cipher\_text.delete(0, tk.END)  
 entry\_cipher\_text.insert(0, cipher\_text)  
  
 global stored\_params  
 stored\_params = (r, priv\_key, Gsys, n)  
 except Exception as e:  
 messagebox.showerror("Ошибка", str(e))  
  
def decrypt\_callback():  
 try:  
 r, priv\_key, Gsys, n = stored\_params  
 cipher\_text = entry\_cipher\_text.get()  
 plain\_text = decrypt(cipher\_text, priv\_key, Gsys, n).rstrip(chr(0x00))  
 entry\_decrypted\_text.delete(0, tk.END)  
 entry\_decrypted\_text.insert(0, plain\_text)  
 except Exception as e:  
 messagebox.showerror("Ошибка", str(e))  
  
  
stored\_params = None  
  
root = tk.Tk()  
root.title("Криптосистема Мак-Элиса")  
  
tk.Label(root, text="Введите r:").grid(row=0, column=0)  
entry\_r = tk.Entry(root)  
entry\_r.grid(row=0, column=1)  
  
tk.Label(root, text="Введите открытый текст:").grid(row=1, column=0)  
entry\_plain\_text = tk.Entry(root, width=50)  
entry\_plain\_text.grid(row=1, column=1)  
  
tk.Button(root, text="Зашифровать", command=encrypt\_callback).grid(row=2, column=0, columnspan=2)  
  
tk.Label(root, text="Зашифрованный текст:").grid(row=3, column=0)  
entry\_cipher\_text = tk.Entry(root, width=50)  
entry\_cipher\_text.grid(row=3, column=1)  
  
tk.Button(root, text="Расшифровать", command=decrypt\_callback).grid(row=4, column=0, columnspan=2)  
  
tk.Label(root, text="Расшифрованный текст:").grid(row=5, column=0)  
entry\_decrypted\_text = tk.Entry(root, width=50)  
entry\_decrypted\_text.grid(row=5, column=1)  
root.mainloop()