СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#__RefHeading___Toc12578_1470056244)

[1 Теоретическая часть 5](#__RefHeading___Toc12580_1470056244)

[1.1 Теоретические аспекты реализации шифра Атбаш 5](#__RefHeading___Toc12582_1470056244)

[1.2 Теоретические аспекты реализации шифра Аффинный 7](#__RefHeading___Toc12584_1470056244)

[1.3 Теоретические аспекты реализации шифра XOR 9](#__RefHeading___Toc12586_1470056244)

[2 Практическая часть 11](#__RefHeading___Toc12588_1470056244)

[2.1 Техническое задание 11](#__RefHeading___Toc12590_1470056244)

[2.2 Постановка задачи 16](#__RefHeading___Toc12592_1470056244)

[2.3 Алгоритм решения 17](#__RefHeading___Toc12594_1470056244)

[2.4 Руководство пользователя 18](#__RefHeading___Toc12596_1470056244)

[2.5 Руководство системного программиста 23](#__RefHeading___Toc12598_1470056244)

[2.6 Контрольный пример 25](#__RefHeading___Toc12600_1470056244)

[ЗАКЛЮЧЕНИЕ 41](#__RefHeading___Toc12602_1470056244)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#__RefHeading___Toc12604_1470056244)

[ПРИЛОЖЕНИЕ А Листинг программы 44](#__RefHeading___Toc12606_1470056244)

[ПРИЛОЖЕНИЕ Б Проверка теоретической части антиплагиатом 45](#__RefHeading___Toc12608_1470056244)

# ВВЕДЕНИЕ

С древнейших времён потребность в защите информации от несанкционированного доступа не только не утратила своей актуальности, но и возросла во много раз. В условиях колоссальных объемов данных и множества возможностей их передачи, постоянно возникают новые методы их хищения, модификации, фальсификации или просто перехвата. В связи с этим, использование современных методов обеспечения конфиденциальности и целостности данных является крайне важной задачей.

Данная работа направлена на изучение и практическую реализацию нескольких методов шифрования, имеющих историческую значимость и демонстрирующих фундаментальные принципы защиты информации. В качестве объектов исследования были выбраны три различных по своему принципу действия алгоритма: шифр Атбаш, шифр Аффинный и шифр XOR. Их изучение в рамках одной работы позволяет провести сравнительный анализ и продемонстрировать многообразие подходов к обеспечению конфиденциальности данных.

Главной целью данной Расчётно-Графической работы является разработка приложения, позволяющего зашифровать любой файл для безопасного его распространения, а также расшифровать этот файл конечным получателем. Для достижения этой цели были поставлены следующие задачи:

— Провести анализ работы выбранных алгоритмов и их математическую модель

— Продумать основные модули программы, необходимые для корректной работы

— Спроектировать взаимосвязь модулей программы между собой

— Разработать программу с учётом требований к соблюдению конвенций именования и коду

— Протестировать работоспособность программы «от А до Я», а также обработку ошибок

Ожидаемым результатом работы станет разработанное программное обеспечение, обладающее как практической ценностью для защиты данных, так и демонстрационной – для изучения принципов работы исторических криптографических алгоритмов.

# 1 Теоретическая часть

Теоретическая часть работы посвящена анализу выбранных для реализации алгоритмов шифрования: Атбаш, Аффинный и XOR. Целью данного анализа являлось выявление их ключевых особенностей, областей применения и криптографических свойств для последующего обоснования архитектурных решений. Исследование каждого алгоритма проводилось по следующим направлениям: исторический контекст и принцип действия, формальная математическая модель, а также анализ преимуществ и недостатков с точки зрения криптостойкости и практического применения.

# 1.1 Теоретические аспекты реализации шифра Атбаш

**История**

Шифр Атбаш — один из древнейших моноалфавитных шифров замены, известный по древнееврейской письменности. Его принцип состоит в зеркальном отображении алфавита: каждой букве соответствует буква, симметричная относительно центра. В данной работе шифр адаптирован для бинарных данных и работает с байтами 0–255.

Для байтового представления применяется отображение x → 255 − x. Ключ не используется, а операции шифрования и дешифрования совпадают.

**Математическая модель**

Основные элементы модели:

— Алфавит (A): Множество из 256 элементов, соответствующее байтам (0–255), используемым для представления данных в бинарных файлах.

— Ключ (K): Последовательность байтов K = {K0, K1, …, Kn-1}, где n – длина ключа, а каждый Kj [0, 255].

— Пространство преобразований: множество байтов 0–255; преобразование x → 255 − x.

— Функция шифрования: для открытого текста P = {P0, P1, …, Pm-1} шифрованный текст C вычисляется как Ci = 255 − Pi.

— Функция дешифрования: Pi = 255 − Ci (совпадает с шифрованием).

Например, нам необходимо зашифровать «HelloWorld!» с ключом «123». В байтовом представлении фраза выглядит как [72, 101, 108, 108, 111, 87, 111, 114, 108, 100, 33]. На каждый байт цифры ключа будут чередоваться: 1,2,3,1,2,3 и т.д. Таким образом, зашифрованный текст будет выглядеть как [73, 103, 111, 109, 113, 90, 112, 116, 111, 101, 35] в байтовом представлении, в текстовом представлении это выглядит как «IgomqZptoe#».

В программной реализации шифра добавлены проверки на пустой ключ и некорректные значения для повышения надёжности программы.

**Преимущества и недостатки**

Преимущества: предельно простая реализация и отсутствие ключа. Недостатки: моноалфавитность и низкая стойкость, поэтому шифр подходит только для учебных целей.

# 1.2 Теоретические аспекты реализации шифра Аффинный

**История**

Шифр Аффинный — классический моноалфавитный шифр замены, задаваемый линейным преобразованием C = (a·P + b) mod m, где a и b — ключевые параметры. Для корректного дешифрования a должно быть взаимно простым с m. В работе используется m = 256 для обработки байтовых данных.

**Математическая модель**

Основные элементы модели:

— Алфавит (A): Множество из 256 элементов, соответствующее байтам (0–255), для представления произвольных данных.

— Ключ (K): Последовательность байтов K = {K0, K1, …, Km-1}, где длина K = m равна длине открытого текста P, и каждый Ki – случайное значение из [0, 255].

— Пространство преобразований: Z\_256 (байты 0–255).

— Функция шифрования/дешифрования: C = (a·P + b) mod 256; P = a^{-1}·(C − b) mod 256.

Например, нам необходимо зашифровать «Linux» c ключом [230, 123, 010, 255, 000]. Представим текст и ключ в виде байтов в 2СС:

Текст – 01001100, 01101001, 01101110, 01110101, 01111000

Ключ – 11100110, 01111011, 00001010, 11111111, 00000000

Результат – 10101010, 00010010, 01100100, 10001010, 01111000

В реализации шифр использует одну функцию для обоих режимов, с обязательной проверкой равенства длин текста и ключа, а также возможностью генерации случайного ключа для шифрования.

**Преимущества и недостатки**

Преимущества: простота и высокая скорость вычислений, компактный ключ (два байта).

Недостатки: моноалфавитность и низкая криптостойкость; уязвим к частотному анализу.

# 1.3 Теоретические аспекты реализации шифра XOR

**История**

Шифр Скитала, также известный как ленточный шифр, представляет собой один из древнейших методов криптографии, использовавшийся в Древней Греции и Спарте около V века до н. э. для передачи секретных сообщений во время военных кампаний [4]. Название происходит от греческого слова "скитала" – палки или цилиндра, на который наматывалась лента с текстом. Для шифрования сообщение записывалось на ленту, намотанную на цилиндр определённого диаметра (ключ – диаметр или число витков), а при разматывании текст становился хаотичным [5]. Дешифровка требовала цилиндра такого же диаметра, чтобы лента наматывалась правильно и текст читался по строкам [4]. Этот шифр был эффективен в эпоху, когда криптоанализ был ручным, и получил применение в различных исторических контекстах, включая интерпретацию пророчеств Нострадамуса, где принцип Скитала использовался для расстановки катренов. В данной работе шифр адаптирован для обработки бинарных файлов, где ключ – число строк таблицы, а байты переставляются по столбцам для шифрования и обратно для дешифровки, что делает его универсальным для современных задач перестановочной криптографии.

**Математическая модель**

Основные элементы модели:

— Алфавит (A): Множество из 256 элементов, соответствующих байтам (0–255), для представления произвольных данных.

— Ключ (K): Целое число K [1, 255], определяющее количество строк таблицы (rows = K), где столбцы (columns) вычисляются как columns = [m/K] а m – длина текста.

— Пространство преобразований: побитовое XOR над байтами (Z\_2^8).

— Функция шифрования: C\_i = P\_i XOR K\_{i mod n}, где n — длина ключа.

— Функция дешифрования: P\_i = C\_i XOR K\_{i mod n}.

В реализации шифр использует массивы байтов, с математической формулой индекса для перестановки, что сохраняет исходную длину данных и упрощает обработку неполных строк.

**Преимущества и недостатки**

Преимущества: простота и высокая скорость.

Недостатки: при повторяющемся ключе стойкость низкая; абсолютная стойкость возможна только при одноразовом ключе той же длины, что и сообщение.

# 2 Практическая часть

# 2.1 Техническое задание

1. ВВЕДЕНИЕ

Настоящий документ, выполненный в соответствии с ГОСТ 19.201-78 «Единая система программной документации. Техническое задание. Требования к содержанию и оформлению», представляет собой пояснительную записку (техническое задание) к проекту «EncoDecs» (далее – Приложение). Документ определяет цели, назначение, технические требования, а также порядок контроля и приемки разрабатываемого программного средства. Разработка выполняется в рамках расчетно-графической работы (РГР).

2. НАИМЕНОВАНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

— Наименование программы: Программное средство «EncoDecs».

— Область применения: Приложение предназначено для использования в учебных и демонстрационных целях для наглядного изучения и применения различных алгоритмов шифрования и дешифрования. Оно позволяет пользователям шифровать/дешифровать любые файлы.

3. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

Основанием для разработки Приложения является учебное задание на выполнение расчетно-графической работы (РГР).

4. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Основным назначением разработки является:

— Разработка приложения, использующего алгоритмы шифрования/дешифрования.

— Предоставление пользователю интерфейса для выполнения операций шифрования/дешифрования файлов.

— Практическая демонстрация использования выбранных алгоритмов шифрования/дешифрования.

— Наглядное применение алгоритмов криптографической защиты информации в образовательных целях.

5. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

5.1. Требования к функциональным характеристикам

5.1.1. Общие функции приложения Приложение должно предоставлять пользователю возможность выбора следующих основных действий:

— Выбор процесса шифрования файла.

— Выбор процесса дешифрования файла.

5.1.4. Процесс шифрования и дешифрования файла При выборе данного процесса, Приложение должно обеспечивать:

— Выбор одного из алгоритмов шифрования/дешифрования.

— Ввод ключа, соответствующего выбранному алгоритму.

— Запрос пути к обрабатываемому файлу (исходному для шифрования или зашифрованному для дешифрования).

Если указанный пользователем путь к файлу не существует, приложение должно информировать об этом пользователя и, в случае ввода пути для нового (выходного) файла, предложить создать его (или необходимые директории).

— Запрос пути для результирующего файла (зашифрованного или дешифрованного).

— Поддержку работы с файлами через файловые потоки, допускается работа с бинарными файлами.

5.2. Требования к алгоритмам Алгоритмы шифрования и дешифрования должны удовлетворять следующим требованиям:

— Иметь унифицированный интерфейс (входные функции, типы данных) для обеспечения взаимозаменяемости и поддержки операций как шифрования, так и дешифрования.

— Обеспечивать поддержку широкого набора символов, включая кириллицу, латиницу, цифры и специальные символы (все символы, представленные на клавиатуре, соответствующей ГОСТ Р ISO 9241-4-2009), предпочтительно через обработку байтовых последовательностей или с использованием кодировки UTF-8.

— Важно: Реализация алгоритмов в библиотеках не должна использовать переменные, содержащие явное перечисление символов алфавита (например, string alphabet = "abc..."). Исключение составляют алгоритмы, где определение специфического алфавита является неотъемлемой частью самого метода (например, азбука Морзе). Обработка символов должна производиться на основе их кодовых значений.

5.3. Требования к архитектуре и качеству кода

— Архитектура: Код приложения должен быть структурирован с применением принципов чистой архитектуры для обеспечения гибкости, тестируемости и понятности.

— Именование: необходимо строго соблюдать общепринятые конвенции именования (например, CamelCase, snake\_case – в зависимости от выбранного стиля для C++) для переменных, функций, классов и файлов.

— Структура проекта: Проект должен быть многофайловым, с логическим разделением кода на модули (например, интерфейс пользователя, бизнес-логика, работа с библиотеками).

5.4. Требования к надежности (согласно ГОСТ Р ИСО/МЭК 25010-2015) Приложение должно обладать высокой степенью надежности, что включает:

— Завершенность: Корректное выполнение всех заявленных функций.

— Отказоустойчивость: Стабильная работа приложения без аварийных завершений. Обработка исключительных ситуаций должна осуществляться с использованием механизмов try-catch (или аналогичных для C++), с выводом информативных сообщений об ошибках пользователю в соответствии с ГОСТ 19.201-78.

— Восстанавливаемость: Корректное сохранение данных и состояния при возникновении ошибок, если это применимо.

— Приложение должно избегать состояния «не отвечает».

5.5. Требования к производительности (согласно ГОСТ Р ИСО/МЭК 25010-2015)

— Время отклика: Быстрый запуск приложения (не более 3 секунд). Операции шифрования/дешифрования должны выполняться с приемлемой скоростью.

— Использование ресурсов: Эффективное использование ресурсов устройства (ЦП, память, батарея).

5.6. Требования к безопасности (согласно ГОСТ 56939-2024)

— Проверка ввода: Проверка корректности ввода данных пользователем (пути к файлам, ключи) согласно пункту 5.17 ГОСТ 56939-2024 «Защита информации. Разработка безопасного программного обеспечения. Общие требования».

— Криптостойкость: Использование библиотек с большими типами данных для поддержки криптостойких алгоритмов.

5.7. Требования к интерфейсу и локализации (согласно ГОСТ Р ИСО 9241-110-2020 и ГОСТ Р ИСО/МЭК 25010-2015).

— Удобство использования (Usability): Интерфейс должен быть интуитивно понятным, соответствовать принципам диалогового взаимодействия, изложенным в ГОСТ Р ИСО 9241-110-2020.

— Язык интерфейса: Русский.

5.8. Требования к составу и параметрам технических средств (платформы) и используемым технологиям

— Планируемые платформы: Linux, MacOS.

— Язык программирования: C++.

— Работа с файлами: Использование файловых потоков для операций ввода-вывода; допускается работа с бинарными файлами для повышения универсальности файловых операций.

5.9. Требования к программной документации Состав программной документации, согласно ГОСТ 19.101-77 «Единая система программной документации. Виды программ и программных документов», включает настоящую пояснительную записку (техническое задание) и отчет о выполненной работе. Отчет о выполненной работе должен соответствовать требованиям, изложенным в исходном задании.

6. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ (согласно ГОСТ Р ИСО/МЭК 12207-2010) Разработка программного средства будет осуществляться в соответствии с процессами жизненного цикла, рекомендованными ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств», и включает следующие этапы:

Анализ требований и проектирование архитектуры: Уточнение функциональных и нефункциональных требований, проектирование общей архитектуры приложения и интерфейсов взаимодействия компонентов.

Разработка библиотек: Реализация алгоритмов шифрования/дешифрования.

Разработка основного приложения: Реализация пользовательского интерфейса и логики управления.

Интеграция и тестирование: Сборка компонентов, проведение функционального и нефункционального тестирования, отладка.

Подготовка отчетной документации.

7. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

Полностью готовый отчет, оформленный в соответствии с установленными требованиями, предоставляется на проверку преподавателю.

После получения положительного вердикта по отчету готовое приложение демонстрируется преподавателю. Приемка осуществляется на основании соответствия Приложения требованиям настоящего Технического задания (ГОСТ 19.201-78).

8. ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов.
2. ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению.
3. ГОСТ Р ISO 9241-4-2009 Эргономика взаимодействия человек-система. Часть 4. Физические устройства ввода.
4. ГОСТ 56939-2024 Защита информации. Разработка безопасного программного обеспечения. Общие требования.
5. ГОСТ Р ИСО/МЭК 12207-2010 Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств.
6. ГОСТ Р ИСО/МЭК 25010-2015 Информационная технология. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов.
7. ГОСТ Р ИСО 9241-110-2020 Эргономика взаимодействия человек-система. Часть 110. Принципы диалогового взаимодействия.

# 2.2 Постановка задачи

Необходимо разработать программу, которая должна шифровать и расшифровывать исходные данные одним из представленных алгоритмов. Программа должна выполнять следующие основные действия:

— Предусмотреть ввод исходного текста с клавиатуры или загрузку данных из файла;

— Реализовать шифрование данных и демонстрацию полученных результатов;

— Реализовать дешифрование (расшифрование) данных и демонстрацию полученных результатов (консоль, файл).

Методы шифрования реализовать в виде отдельных функций/модулей.

# 2.3 Алгоритм решения

Ниде представлен алгоритм взаимодействия пользователя с программой

1. Запуск программы: Очистка экрана, вывод главного меню
2. Вывод выбора режима: На экран выводится меню с выбором режима:
   1. Шифрование
   2. Дешифрование
   3. Выход
3. Выбор режима: Пользователь вводит номер желаемого режима.
4. Вывод выбора алгоритма: На экран выводится меню с выбором алгоритма шифрования для последующей работы:
   1. Шифр XOR основан на операции исключающего ИЛИ: для каждого байта вычисляется C\_i = P\_i XOR K\_i.
   2. В данной работе используется ключ фиксированной длины, который циклически повторяется по всему файлу. Дешифрование выполняется той же операцией XOR.
   3. Симметричность операции XOR позволяет использовать один и тот же алгоритм для шифрования и дешифрования.
5. Выбор алгоритма: Пользователь вводит номер желаемого алгоритма.
6. Запрос пути до файла: Программа запрашивает у пользователя путь до файла для последующей его обработки.
7. Ввод пути до файла: Пользователь вводит путь до желаемого файла. Программа проверяет корректность пути, корректность файла, и после успешных проверок считывает файл в массив байт.
8. Вывод выбора источника ключа: На экран выводится меню с выбором источника ключа:
   1. Ввести
   2. Из файла
   3. Сгенерировать (Недоступно при дешифровании)
9. Выбор источника ключа: Пользователь выбирает желаемый источник ключа.
10. Получение ключа: В зависимости от выбранного источника ключ вводится, читается из файла, либо генерируется с соблюдением ограничений, индивидуальных для каждого шифра.
11. Запрос подтверждения: Когда выбран режим и шифр, а также получен массив байт текста и ключ, программа выводит имеющиеся данные о предстоящей задаче и запрашивает у пользователя подтверждение начала работы.
12. Подтверждение: Пользователь подтверждает или отклоняет начало работы алгоритма.
13. Выполнение: Если подтверждение получено, программа начинает работу по преобразованию исходного файла, отображая прогресс.
14. Запрос сохранения результата: После завершения обработки файла, программа предлагает сохранить результат в отдельный файл.
15. Сохранение результата: Результат работы записывается в файл.
16. Повтор: После успешно выполненной работы программа возвращается в главное меню.

# 2.4 Руководство пользователя

2.4.1 Общие сведения

Программа «cipher\_app» предназначена для выполнения операций шифрования и дешифрования данных с использованием криптоалгоритмов.

Программа предоставляет пользователю возможности:

— Ввод текста для шифрования/дешифрования из файла

— Генерация ключей для алгоритмов

— Вывод результатов операций на экран или в файл

Программа реализована на языке C++. Работает в среде совместимой с ОС Linux. Дисковой памяти для запуска программы требуется не менее 0.123 МБ. Оперативной памяти для нормальной работы требуется не менее 4.6 МБ.

2.4.2 Подготовка к работе

Убедитесь, что файл для обработки существует, а на устройстве достаточно оперативной памяти.

2.4.3 Порядок работы

Шаг 1: Запуск программы: Выводится главное меню с заголовком, приветствием и опциями.

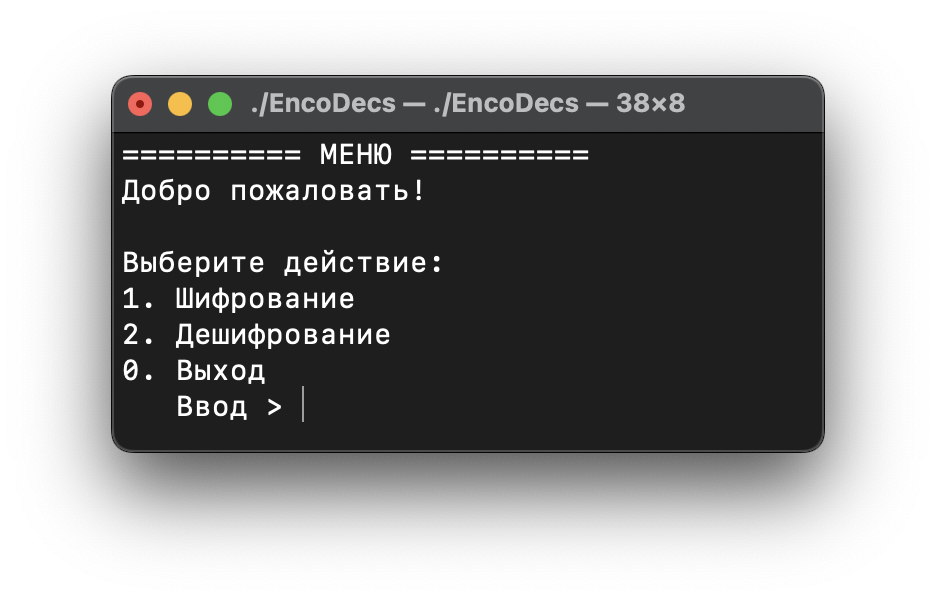


Рисунок 2.4.1 – Главное меню программы

Шаг 2: Выбор шифра: Выводится меню с выбором шифра для работы.

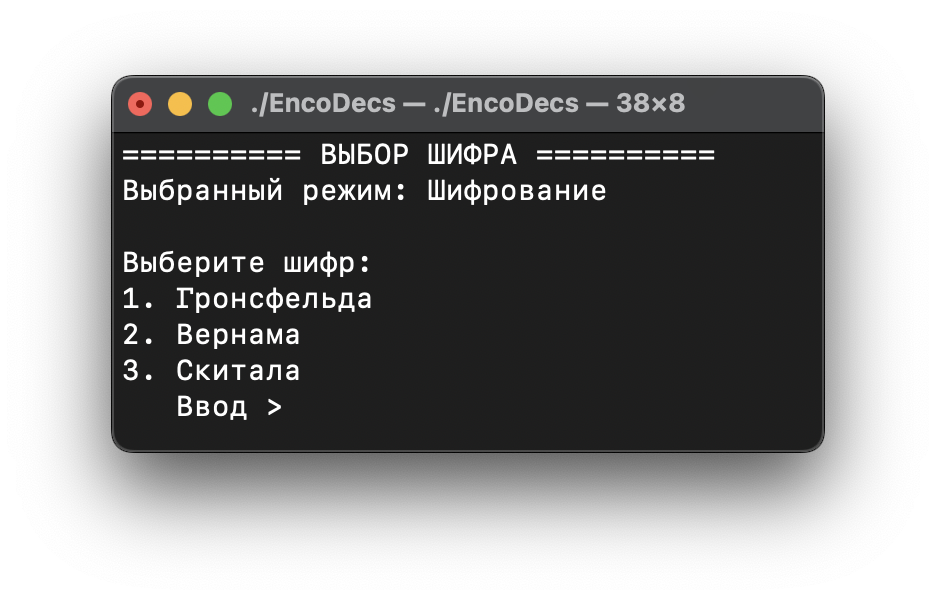


Рисунок 2.4.1 – Меню выбора шифра

Шаг 3: Чтение файла: Запрашивается путь до файла.

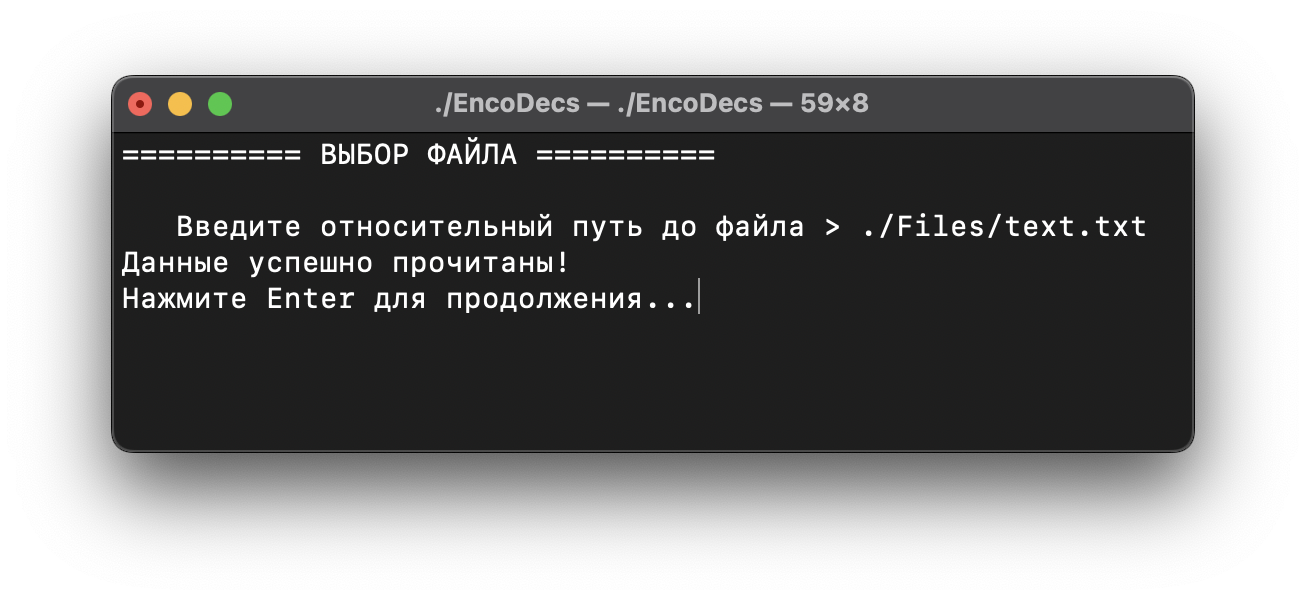


Рисунок 2.4.3 – Запрос пути и чтение файла

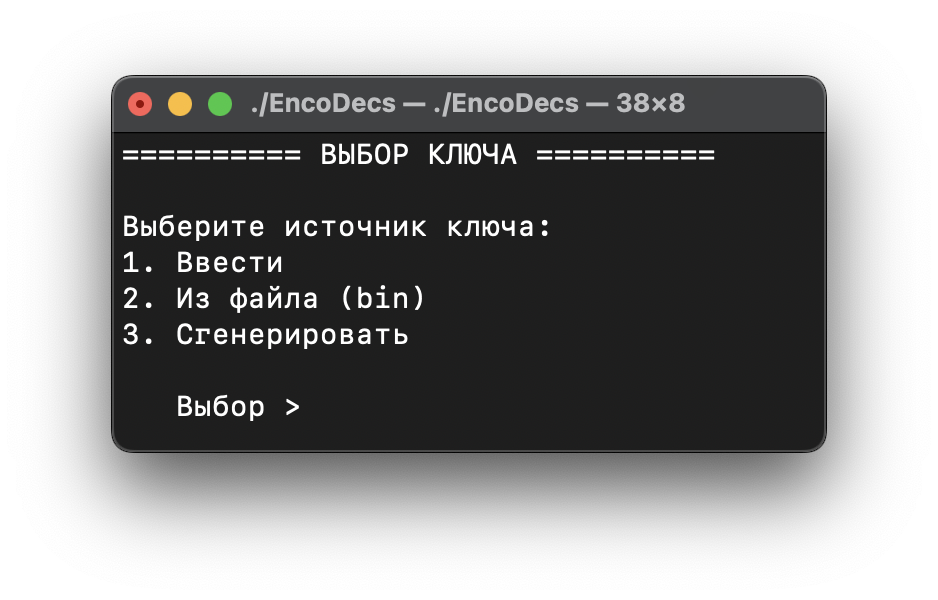


Рисунок 2.4.4 – Запрос источника ключа

Шаг 5.1: Ввод ключа: Запрашивается ввод ключа в терминал

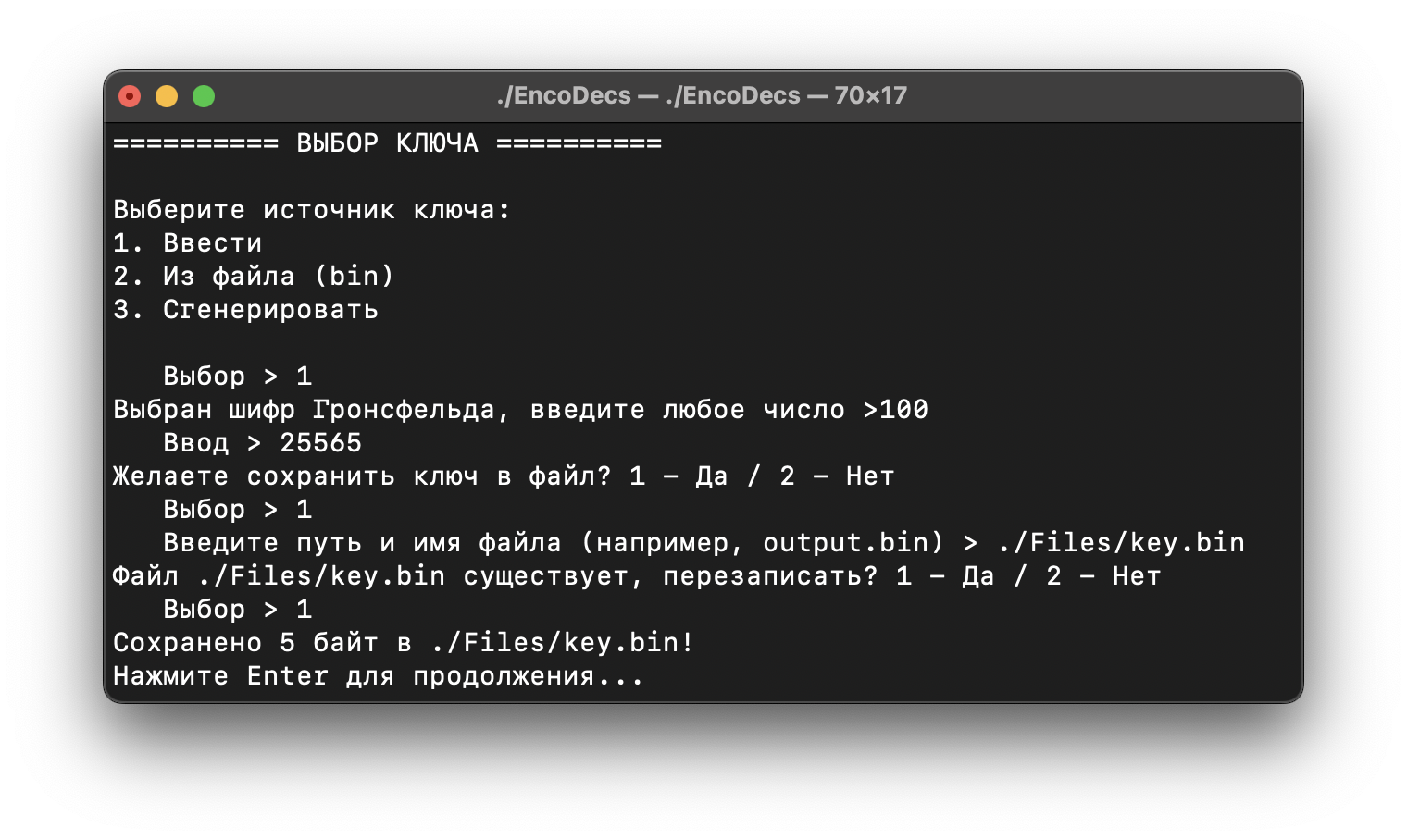


Рисунок 2.4.5.1 – Ввод ключа в терминал

Шаг 5.2: Чтение ключа из файла: Запрашивается путь до файла, содержащего ключ

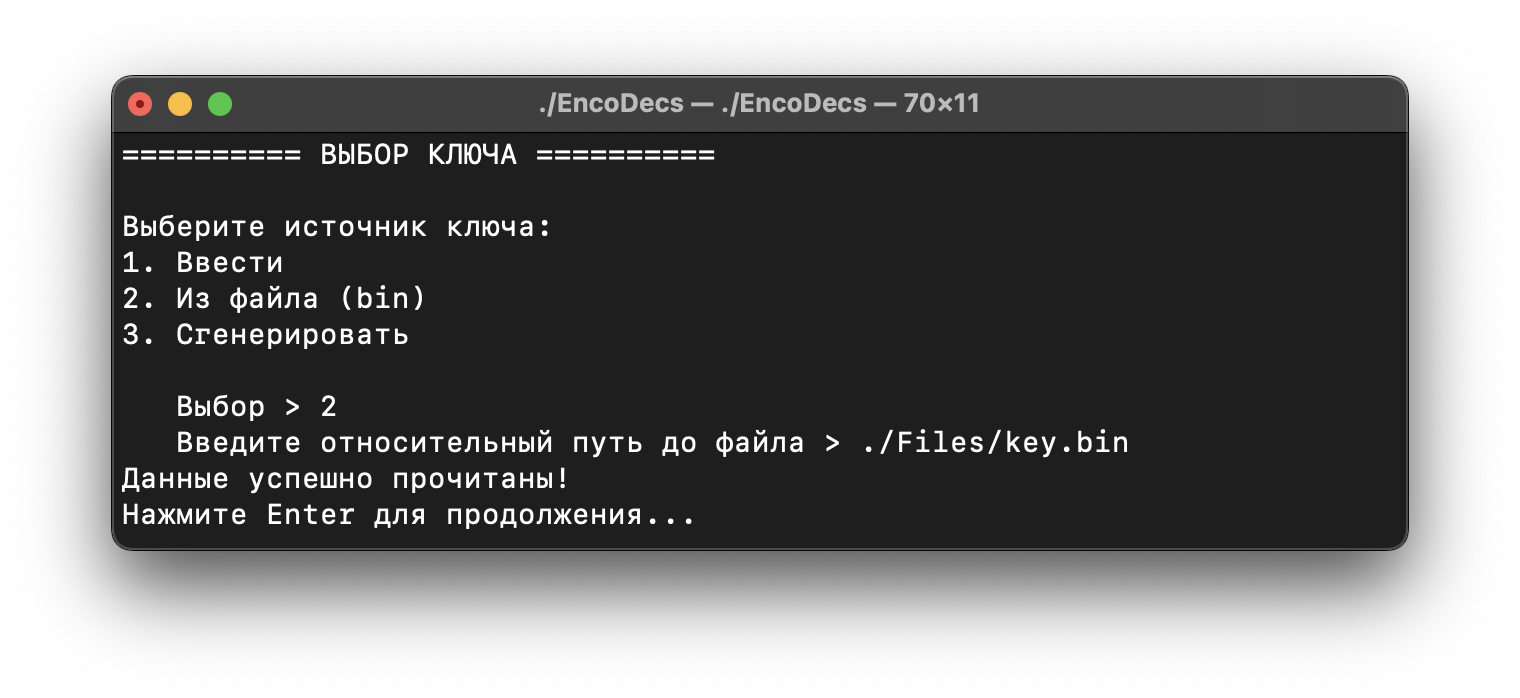


Рисунок 2.4.5.2 – Чтение ключа из файла

Шаг 5.3: Генерация ключа: Ключ генерируется под требования каждого шифра

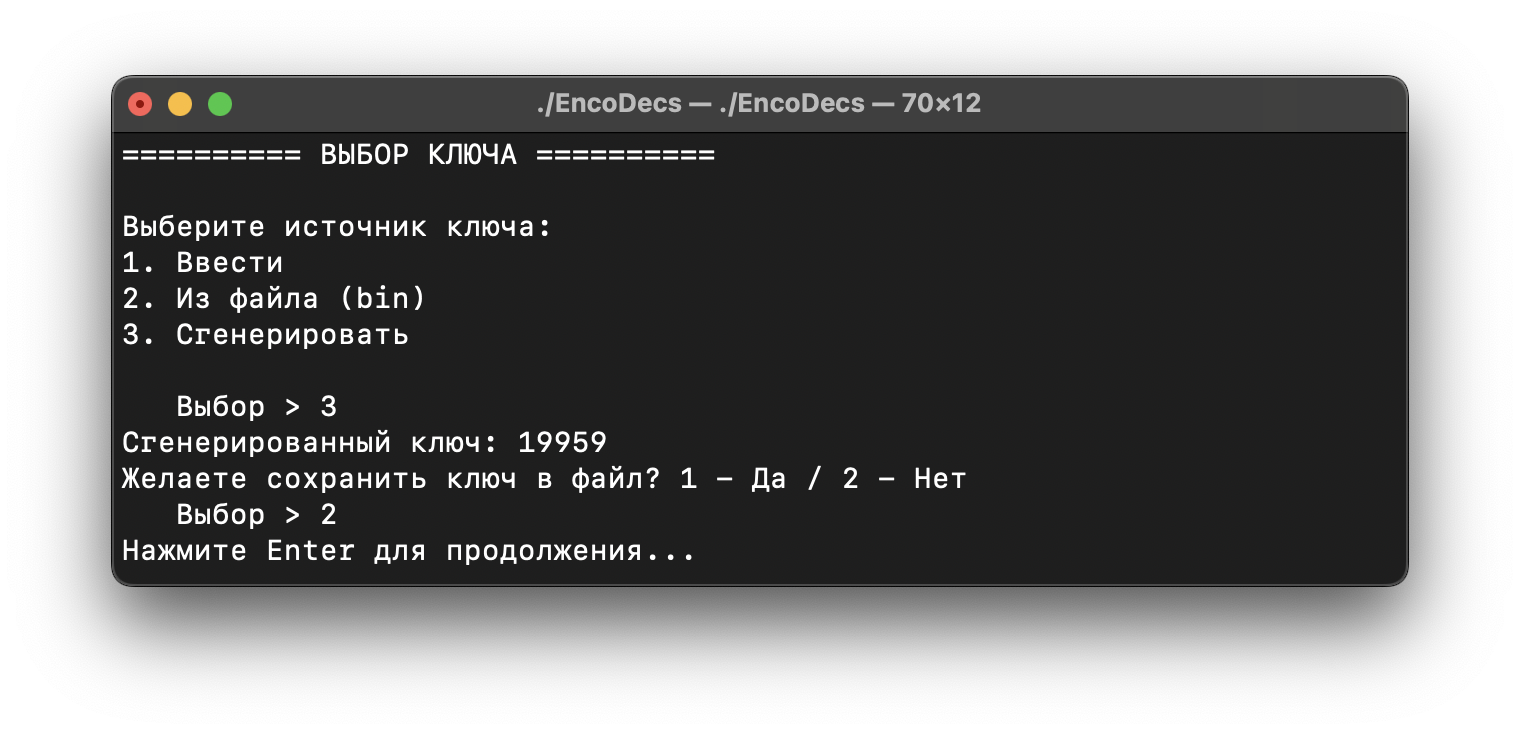


Рисунок 2.4.5.3 – Генерация ключа

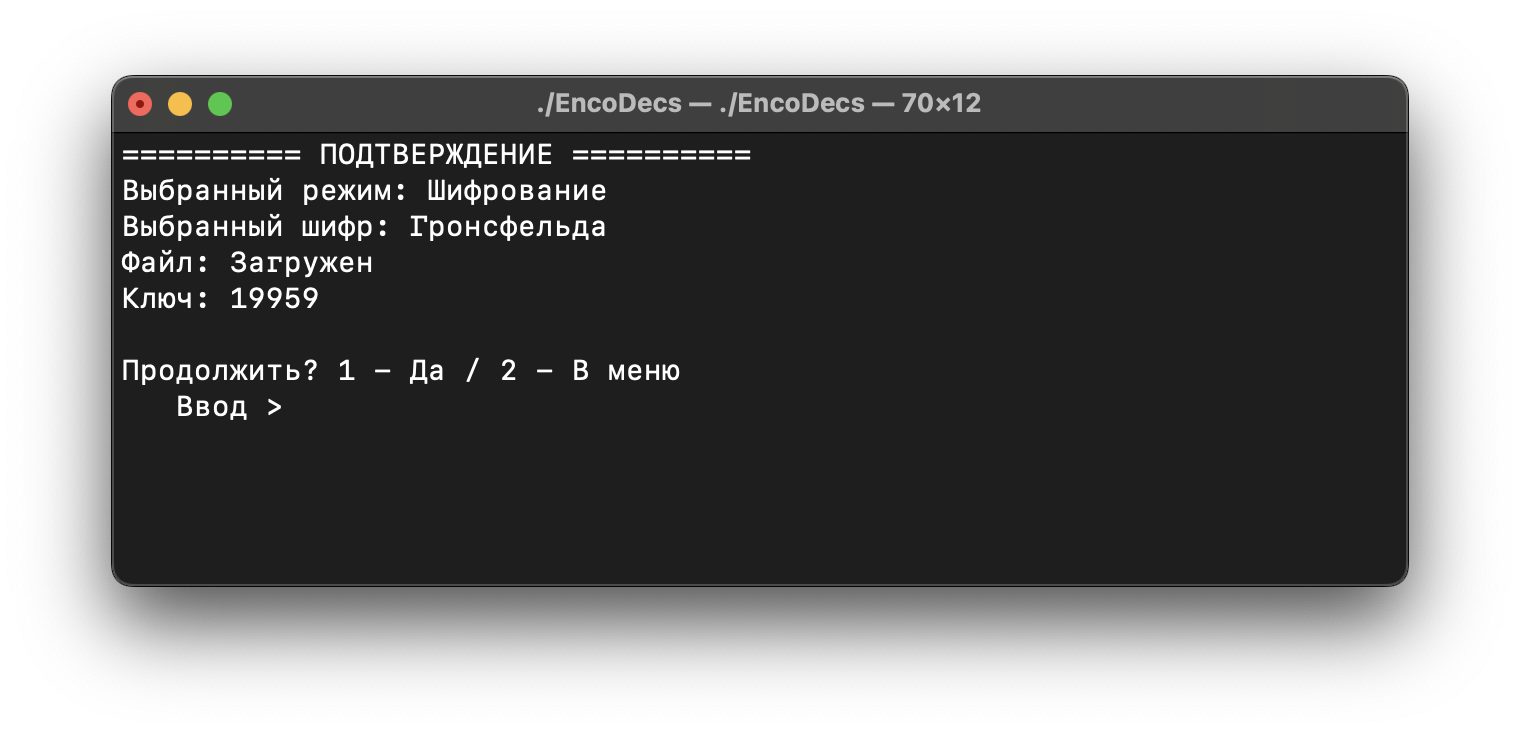


Рисунок 2.4.6 – Окно сверки и подтверждения данных

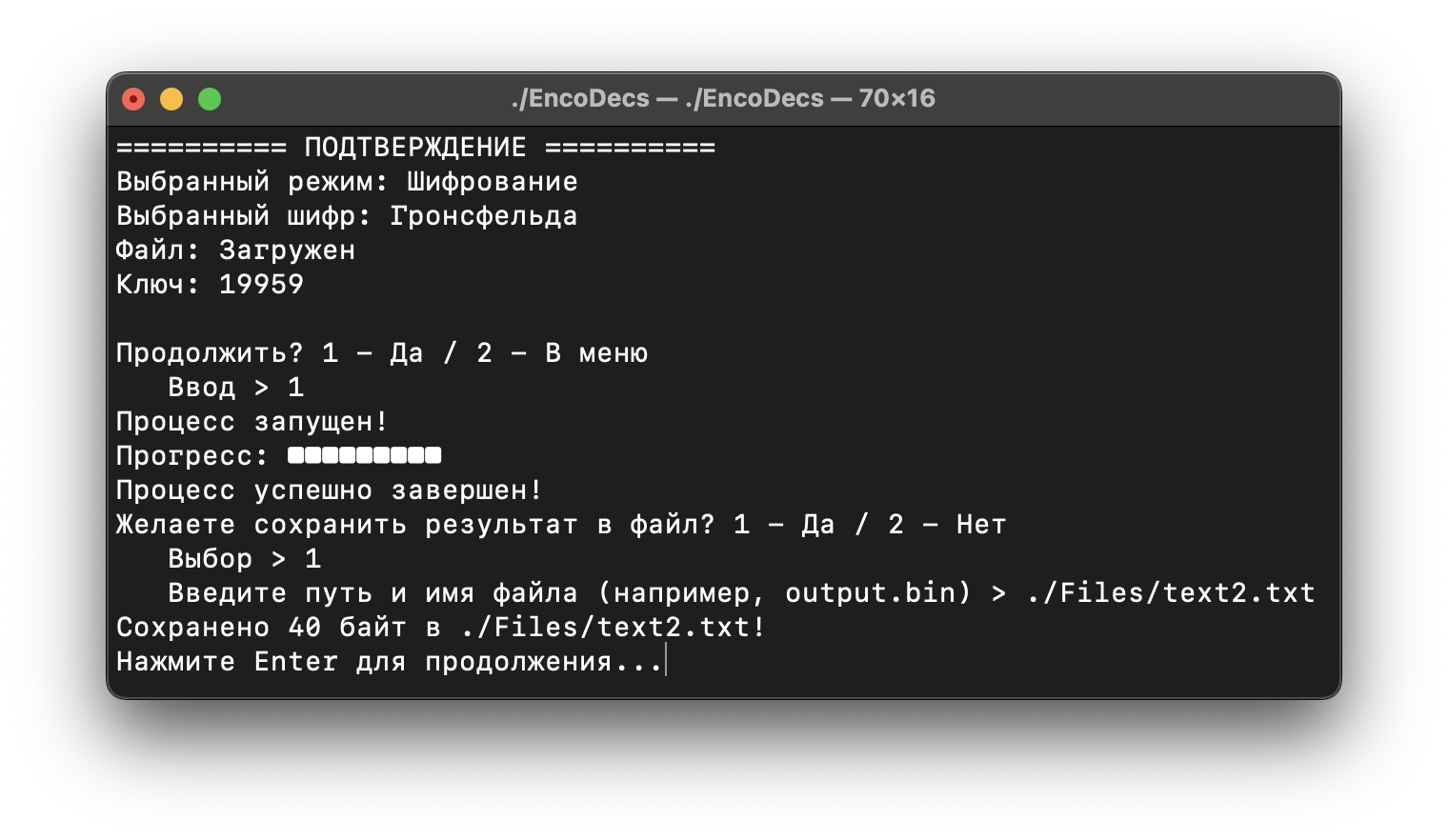


Рисунок 2.4.7 – Алгоритм, завершивший работу

2.4.4 Сообщения пользователю

При работе с программой могут выводиться сообщения, представленные в таблице ниже.

Таблица 2.4.1 – Сообщения программы

2.4.5 Аварийные ситуации

К аварийным ситуациям относятся: нехватка оперативной памяти для создания окна программы. Если исполняемый модуль программы не запускается, либо не выполнены требования условия работы программы, либо один из файлов поврежден, необходимо обратиться к разработчику программы.

# 2.5 Руководство системного программиста

2.5.1 Общие сведения

Программа реализована на языке С++ в среде Visual Studio Code. Интерфейс создан программно, поэтому весь функционал полностью отражен в списке функций и листинге программы.

2.5.2 Структура программы

Программа имеет модульную структуру, что обеспечивает ее гибкость и простоту в сопровождении.

— types.hpp: Заголовочный файл с базовыми типами. Содержит определение byte\_array и enum ModeType, CipherType.

— utils.hpp/cpp: Модуль с утилитарными функциями. Предоставляет инструменты для очистки ввода, паузы, минимума, имен шифров, отображения ключей и валидации ввода.

— file\_io.hpp/cpp: Заголовочный и исходный файл для работы с файловой системой. Содержит функции для чтения и записи бинарных данных.

— get\_keys.hpp/get\_keys.cpp: Модуль для получения ключей. Содержит функции InputKey (ввод), ReadKey (чтение из файла), GenKey (генерация) и GetKey (связующая функция).

— ciphers.hpp/cpp: Модуль с алгоритмами шифрования. Содержит унифицированную функцию EncryptDecrypt.

— main.cpp: Ядро приложения. Основная функция — основной цикл с меню, выбором, чтением, получением ключа, шифрованием и сохранением.

Связь модулей программы между собой представлена на рисунке 2.5.1

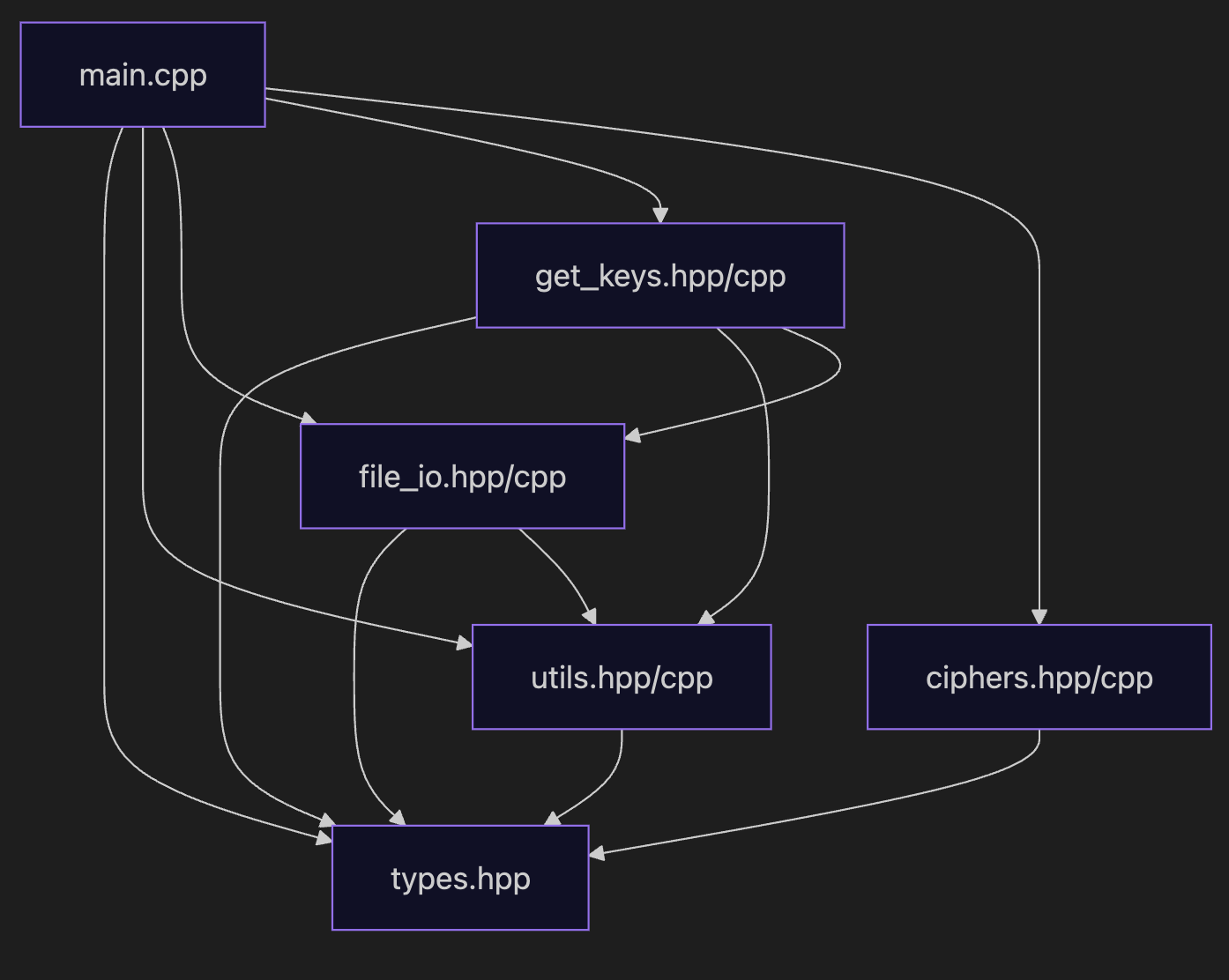


Рисунок 2.5.1 – Связь модулей программы между собой

# 2.6 Контрольный пример

Для проверки корректности работы программы попробуем зашифровать фразу «Мурашко Евгений, АИ-42!» сначала с помощью программы, а затем вручную, и сравним результаты.

Так как программа работает с файлами, текст для шифрования был написан в отдельном документе:

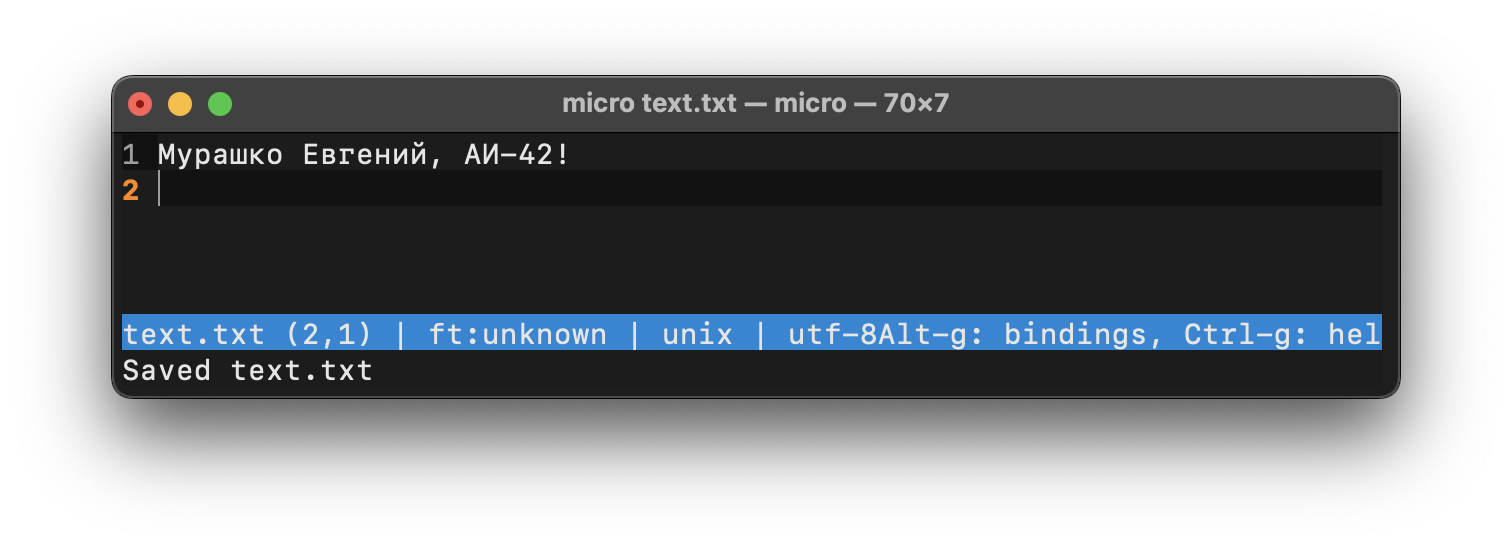


Рисунок 2.6 – Файл с контрольным текстом.

2.6.1 Шифр Атбаш

2.6.1.1 Пример программного преобразования.

В программе выберем режим «Шифрование», алгоритм «Шифр Атбаш», файл по относительному пути «./Files/text.txt» и выполним шифрование без ключа.

2.6.1.2 Пример ручного преобразования.

Формула: Ci = 255 − Pi. Для первых трёх байтов: P1=208 → C1=47; P2=156 → C2=99; P3=209 → C3=46.

2.6.1.3 Сверка результатов.

Результаты программного и ручного расчёта совпадают, следовательно реализация корректна.

2.6.2 Шифр Аффинный

2.6.2.1 Пример программного преобразования.

В программе выберем режим «Шифрование», алгоритм «Шифр Аффинный», файл по относительному пути «./Files/text.txt» и зададим ключ (a, b).

2.6.2.2 Пример ручного преобразования.

Формула: C = (a·P + b) mod 256; P = a^{-1}·(C − b) mod 256. Ручной расчёт выполняется для нескольких байтов и сверяется с программой.

2.6.2.3 Сверка результатов.

Результаты программного и ручного расчёта совпадают, следовательно реализация корректна.

2.6.3 Шифр XOR

2.6.3.1 Пример программного преобразования.

В программе выберем режим «Шифрование», алгоритм «Шифр XOR», файл по относительному пути «./Files/text.txt» и введём ключ из 3 байтов «1 2 3».

2.6.3.2 Пример ручного преобразования.

Формула: C\_i = P\_i XOR K\_{i mod n}. Для первых трёх байтов: 208^1=209; 156^2=158; 209^3=210.

2.6.3.3 Сверка результатов.

Результаты программного и ручного расчёта совпадают, следовательно реализация корректна.

2.6.4 Вывод

Все контрольные расчёты сошлись, следовательно, программа работает корректно.

# ЗАКЛЮЧЕНИЕ

В результате выполнения расчётно-графической работы достигнута основная цель – разработано приложение «EncoDecs», обеспечивающее шифрование и дешифрование произвольных файлов с использованием трёх исторических алгоритмов: Атбаш, Аффинный и XOR.

Рассмотрены ключевые вопросы криптографической защиты информации, включая исторический контекст, принципы действия и математические модели выбранных алгоритмов. Изучены их адаптации для обработки бинарных данных в диапазоне байтов 0–255: отражение байтов 0–255 для Атбаш, аффинное преобразование C = (a·P + b) mod 256 для Аффинный, а также XOR с повторяющимся ключом для XOR. Проведён анализ преимуществ (простота реализации, скорость обработки) и недостатков (низкая криптостойкость при повторяющемся ключе, моноалфавитность).

Разработано консольное приложение на языке C++ с модульной архитектурой, унифицированным интерфейсом для алгоритмов, поддержкой ввода/генерации ключей, потоковой работой с файлами, обработкой ошибок и отображением прогресса. Функциональность подтверждена тестированием, включая контрольный пример шифрования текста, где результаты программного и ручного расчётов полностью совпали.

Достоинства разработки: высокая скорость обработки, универсальность для бинарных файлов, соблюдение принципов чистой архитектуры, надёжность и образовательная ценность. Недостатки: ограниченная криптостойкость алгоритмов, консольный интерфейс, поддержка только платформ Linux и macOS, отсутствие интеграции с современными стандартами безопасности.

По сравнению с аналогичными образовательными инструментами, настоящее приложение превосходит в обработке произвольных файлов и модульности, но уступает профессиональным библиотекам в стойкости и функциональности.

Результаты работы рекомендуется использовать в учебных целях: для лабораторных занятий по криптографии, демонстрации базовых принципов шифрования и тестирования на небольших объёмах данных.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шифр Атбаш // Wikipedia. URL: https://en.wikipedia.org/wiki/Atbash
2. Affine cipher // University of Rhode Island. URL: https://cs.uri.edu/cryptography/classical-ciphers/affine/article.html
3. Affine cipher // Wikipedia. URL: https://en.wikipedia.org/wiki/Affine\_cipher
4. XOR cipher // Wikipedia. URL: https://en.wikipedia.org/wiki/XOR\_cipher
5. XOR Cipher (повторяющийся ключ) // Boxentriq. URL: https://www.boxentriq.com/ciphers/xor-cipher

# ПРИЛОЖЕНИЕ А Листинг программы

Main.cpp – Главный модуль программы

#include <iostream>

#include "types.hpp"

#include "utils.hpp"

#include "file\_io.hpp"

#include "get\_keys.hpp"

#include "ciphers.hpp"

using namespace std;

int main() {

while (true) {

system("clear");

cout << "========== МЕНЮ ==========" << endl;

cout << "Добро пожаловать!" << endl << endl;

cout << "Выберите действие:" << endl;

cout << "1. Шифрование" << endl;

cout << "2. Дешифрование" << endl;

cout << "0. Выход" << endl;

int input\_mode = GetIntInput("Ввод", 0, 2);

auto mode = static\_cast<ModeType>(input\_mode);

if (mode == ModeType::EXIT) {return 0;}

system("clear");

cout << "========== ВЫБОР ШИФРА ==========" << endl;

cout << "Выбранный режим: " << (mode == ModeType::ENCRYPT ? "Шифрование" : "Дешифрование") << endl << endl;

cout << "Выберите шифр:" << endl;

cout << "1. Атбаш" << endl;

cout << "2. Аффинный" << endl;

cout << "3. XOR" << endl;

int input\_cipher = GetIntInput("Ввод", 1, 3);

auto cipher = static\_cast<CipherType>(input\_cipher);

system("clear");

cout << "========== ВЫБОР ФАЙЛА ==========" << endl << endl;

byte\_array file = ReadFileToBytearray();

byte\_array key = GetKey(mode, cipher, file.size());

system("clear");

cout << "========== ПОДТВЕРЖДЕНИЕ ==========" << endl;

cout << "Выбранный режим: " << (mode == ModeType::ENCRYPT ? "Шифрование" : "Дешифрование") << endl;

cout << "Выбранный шифр: " << NameOfCipher(cipher) << endl;

cout << "Файл: Загружен" << endl;

cout << "Ключ: " << DisplayKey(cipher, key) << endl << endl;

cout << "Продолжить? 1 - Да / 2 - В меню" << endl;

int confirm = GetIntInput("Ввод",1,2);

byte\_array result;

if (confirm == 1) {

result = EncryptDecrypt(mode, cipher, file, key);

SaveBytearrayToFile(result, "результат");

WaitForEnter();

}

}

}

Ссылка на GitHub репозиторий проекта:

https://github.com/Lipt4n/Sem2\_RGR/

# ПРИЛОЖЕНИЕ Б Проверка теоретической части антиплагиатом

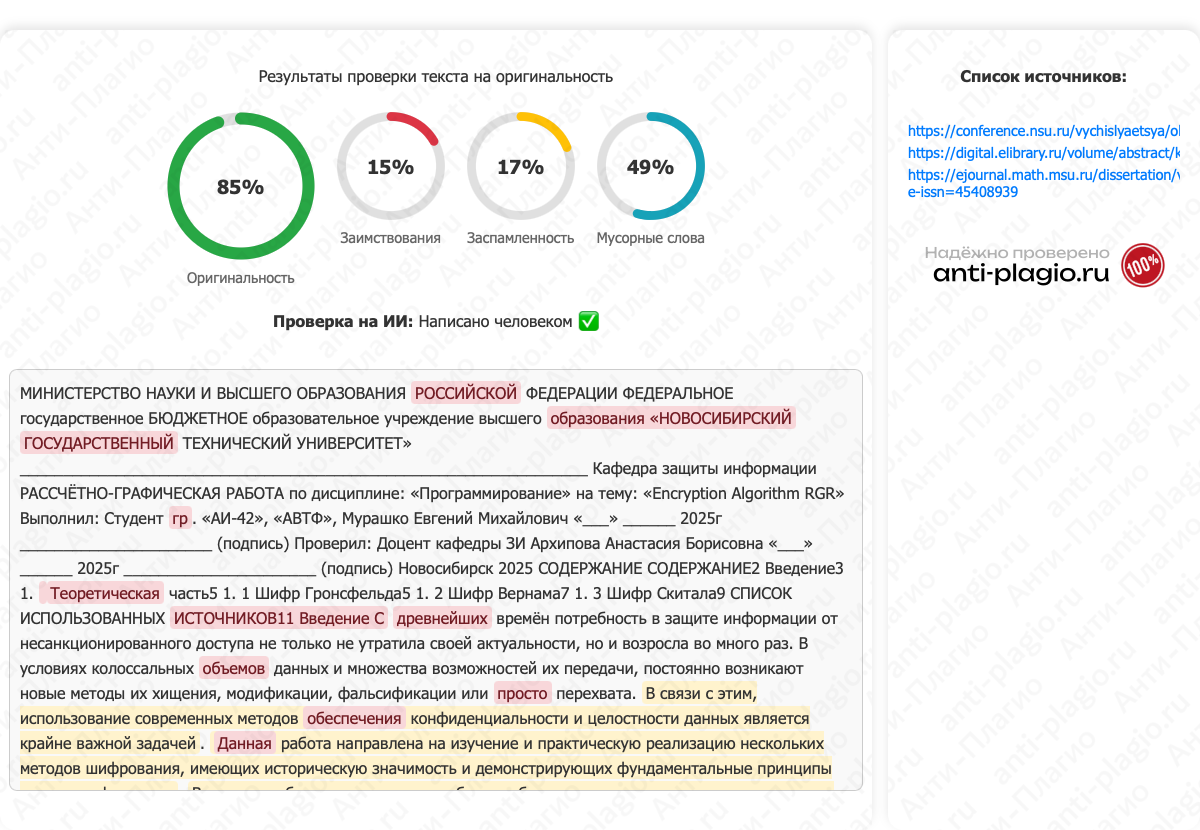


Рисунок Б.1 – Проверка теоретической части в системе «Антиплагиат»