МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Информационных систем и технологий
Специальность 1-98 01 03 Программное обеспечение информационной безопасности мобильных систем

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА:

по дисциплине «Защита информации и надежность информационных систем»
Тема: «Сравнительный анализ производительности и безопасности блочно-симметричных алгоритмов шифрования DES и Speck»

Исполнитель
Студент 4 курса группы 7 Ероховец И. А.
(Ф.И.О.)

Руководитель работы асс. Сазонова Д. В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Минск 2024

## Содержание

# Введение

С развитием цифровых технологий и увеличением объема передаваемой информации вопросы безопасности данных становятся всё более актуальными. Шифрование данных — это один из главных методов защиты, который помогает предотвратить несанкционированный доступ к конфиденциальной информации. Среди множества алгоритмов особое внимание привлекают симметричные алгоритмы, использующие один и тот же ключ как для шифрования, так и для расшифрования.

В этой работе будет проведён сравнительный анализ производительности и безопасности двух симметричных алгоритмов шифрования — DES и Speck. Оба алгоритма относятся к категории симметричного шифрования, что подразумевает использование единого ключа для выполнения обоих процессов. Это делает их удобными для реализации в системах, где необходимо быстрое и эффективное управление ключами.

DES, один из первых широко используемых алгоритмов, известен своей надежностью, хотя и страдает от ограниченной длины ключа. В то же время Speck, разработанный для современных устройств, предлагает высокую скорость работы и гибкость в выборе длины ключа, что делает его эффективным для различных приложений.

Результатом данной работы станет проект, содержащий реализацию обоих алгоритмов, что позволит сравнить их по ряду характеристик.

## 1 Постановка задачи

В рамках данного исследования мы сосредоточимся на сравнительном анализе двух популярных алгоритмов симметричного шифрования: классического DES и современного Speck. Наша цель - оценить их эффективность в контексте современных требований к безопасности данных, сравнив скорость шифрования и расшифрования, а также уровень защищенности, который они обеспечивают.

Для реализации проекта мы выбрали язык программирования JavaScript, который идеально подходит для создания интерактивных веб-приложений. Среда разработки MS Visual Studio Code обеспечит нас удобными инструментами для написания и отладки кода. JavaScript, как высокоуровневый язык, поддерживает асинхронное программирование, что делает его особенно полезным для создания отзывчивых пользовательских интерфейсов. Мы планируем разработать приложение, которое будет осуществлять шифрование и расшифрование текстовых сообщений, что является важной функцией любого инструмента для шифрования в условиях цифровой безопасности.

Ключевым аспектом нашего приложения станет вывод времени, затраченного на шифрование и расшифровку. Это позволит пользователям не только получать результаты работы алгоритмов, но и анализировать их эффективность в зависимости от объема обрабатываемых данных. Также мы уделим внимание созданию удобного интерфейса, который будет включать возможность выбора алгоритма шифрования, что обеспечит доступность приложения для пользователей с различным уровнем подготовки и опыта.

В нашем проекте будет использовано множество библиотек и инструментов, таких как Node.js и Express.js, что позволит создать надежную серверную часть и обеспечить качественное взаимодействие с клиентской стороной. Удобство интерфейса играет важную роль в восприятии приложения, и мы стремимся сделать его максимально интуитивно понятным и функциональным. Кроме того, мы планируем провести детальное сравнение результатов тестирования по производительности и безопасности, чтобы понять, как каждый алгоритм проявляет себя в реальных условиях.

Анализ результатов тестирования будет учитывать не только скорость обработки, но и такие критически важные факторы, как устойчивость к атакам и потребление ресурсов. Мы проведем оценку криптостойкости каждого алгоритма, чтобы определить, какой из них лучше защищает данные от потенциальных угроз. В конечном итоге, мы сформулируем выводы о том, какой алгоритм более подходящий для конкретных условий и задач, что поможет пользователям выбирать наиболее эффективные решения для своих нужд.

Таким образом, данный проект не только углубляет наше понимание работы шифровальных алгоритмов, но и предоставляет практическое приложение для их оценки и использования. Результаты нашего анализа могут стать основой для дальнейших исследований в области криптографии и разработки новых методов защиты данных. Важно подчеркнуть, что правильный выбор алгоритма может существенно повлиять на безопасность и производительность приложений в условиях постоянно меняющейся информационной среды.

### 1.1 Вывод

В рамках данного курсового проекта мы провели всесторонний сравнительный анализ производительности и безопасности симметричных алгоритмов шифрования DES и Speck, используя язык программирования JavaScript в удобной среде MS Visual Studio Code. Разработанное приложение обладает возможностью шифрования и расшифрования текстовых сообщений, а также фиксирует время, затраченное на выполнение этих операций. Это позволяет не только оценить производительность каждого из алгоритмов, но и выявить их сильные и слабые стороны.

Удобный и интуитивно понятный интерфейс приложения обеспечивает простоту ввода данных, что делает его доступным для пользователей с разным уровнем подготовки и опыта. Благодаря этому, проект подчеркивает важность выбора подходящего алгоритма шифрования, который соответствует специфическим требованиям к безопасности и скорости обработки данных в различных условиях.

Таким образом, результаты нашего исследования могут служить ценным ориентиром для практического применения алгоритмов шифрования, а также для дальнейшего изучения вопросов, связанных с безопасностью информации.

## 2 Описание методов

Выбранные для нашего анализа алгоритмы шифрования принадлежат к категории блочно-симметричных шифров. Эти алгоритмы представляют собой группу криптографических методов, которые обрабатывают информацию, разбивая ее на блоки фиксированной длины, такие как 128, 192 или 256 бит. Важно отметить, что они используют симметричное шифрование, что означает, что один и тот же ключ применяется как для процесса шифрования, так и для последующего дешифрования.

Суть работы блочно-симметричных шифров заключается в том, что данные разделяются на равные блоки. Если размер исходной информации не делится на размер блока, используется метод дополнения, который позволяет корректно обработать все данные. В ходе каждого раунда алгоритма ключ модифицируется с помощью специальной функции, что обеспечивает дополнительный уровень безопасности. Каждый блок данных проходит через несколько раундов, в которых осуществляются операции замены, перестановки и смешивания, что делает процесс шифрования более сложным и защищенным от потенциальных атак.

Таким образом, блочно-симметричные шифры представляют собой мощный инструмент для обеспечения безопасности информации, используя тщательно продуманные принципы работы и алгоритмические процессы.

DES (Data Encryption Standard) — это классический алгоритм, реализующий блочное симметричное шифрование и широко использовавшийся в криптографии с момента его создания в 1970-х годах. Он работает с блоками данных фиксированного размера — 64 бита, и использует 56-битный ключ для шифрования и дешифрования. Алгоритм DES основан на концепции сети Фейстеля, что позволяет ему эффективно обрабатывать данные.

Процесс шифрования в DES включает несколько раундов, обычно 16, в ходе которых данные подвергаются различным операциям, таким как замена и перестановка. Эти операции зависят от ключа, который проходит через несколько преобразований, что значительно увеличивает уровень безопасности. Каждое изменение в данных усложняет задачу атакующим, что делает DES эффективным средством защиты информации на протяжении многих лет.

Несмотря на свои достоинства, такие как относительно высокая скорость работы и простота реализации, алгоритм DES в последние десятилетия стал уязвимым для современных методов криптоанализа. Это связано с коротким ключом, который легко поддается атакам методом перебора. В результате, DES был заменен более современными алгоритмами, такими как AES, но он по-прежнему остается важной частью истории криптографии и основой для многих последующих разработок.

Таким образом, DES представляет собой значимый шаг в развитии шифрования, обеспечивая надежную защиту данных на протяжении своего времени, и его принципы продолжают влиять на современные криптографические технологии.

Алгоритм строится на основе сети Фейстеля.

Само название конструкции Фейстеля означает ее ячеистую топологию. Формально одна ячейка сети соответствует одному раунду зашифрования или

расшифрования сообщения. При зашифровании сообщение разбивается на блоки одинаковой и фиксированной длины как правило – 64 или 128 битов.

Полученные блоки называются входными. В случае если длина входного блока меньше, чем выбранный размер, блок удлиняется установленным способом.

Каждый входной блок шифруемого сообщения изначально делится на два подблока одинакового размера: левый ($L_0$) и правый ($R_0$). Далее в каждом i-м раунде выполняются преобразования в соответствии с формальным представлением ячейки сети Фейстеля:

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} + f(R_{i-1}, K_i) \end{cases}$$

По какому-либо математическому правилу вычисляется раундовый ключ $K_i$. В приведенном выражение знак «+» соответствует поразрядному суммированию на основе «XOR». На рис. 2.1 приведено графическое отображение сети Фейстеля.

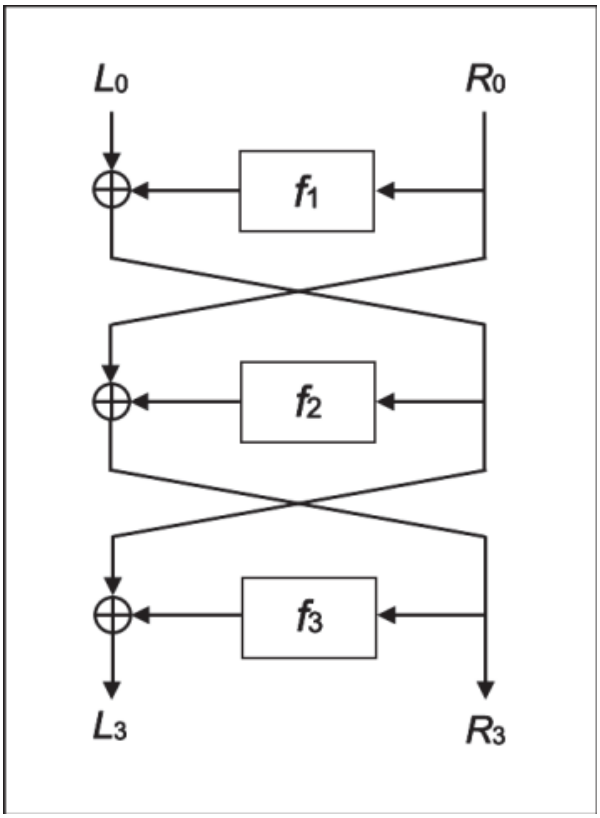


Рисунок 2.1 – Графическое отображение сети Фейстеля

Расшифрование происходит так же, как и зашифрование, с той лишь разницей, что раундовые ключи будут использоваться в обратном порядке по отношению к зашифрованию.

Входной блок данных, состоящий из 64 битов, преобразуется в выходной блок идентичной длины. В алгоритме широко используются рассеивания (подстановки) и перестановки битов текста, о которых мы упоминали выше. Комбинация двух указанных методов преобразования образует фундаментальный строительный блок DES, называемый раундом или циклом.

Один блок данных подвергается преобразованию (и при зашифровании, и при расшифровании) в течение 16 раундов.

После первоначальной перестановки и разделения 64-битного блока данных на правую $R_0$ и левую $L_0$ половины длиной по 32 бита выполняются 16 раундов одинаковых действий.

Вначале правая часть блока $R_i$ расширяется до 48 битов с использованием таблицы, которая определяет перестановку плюс расширение на 16 битов. Эта операция приводит размер правой половины в соответствие с размером ключа для выполнения операции XOR. Кроме того, за счет выполнения этой операции быстрее возрастает зависимость всех битов результата от битов исходных данных и ключа (это называется «лавинный эффект»). Общая схема алгоритма DES представлена на рисунке 2.2.

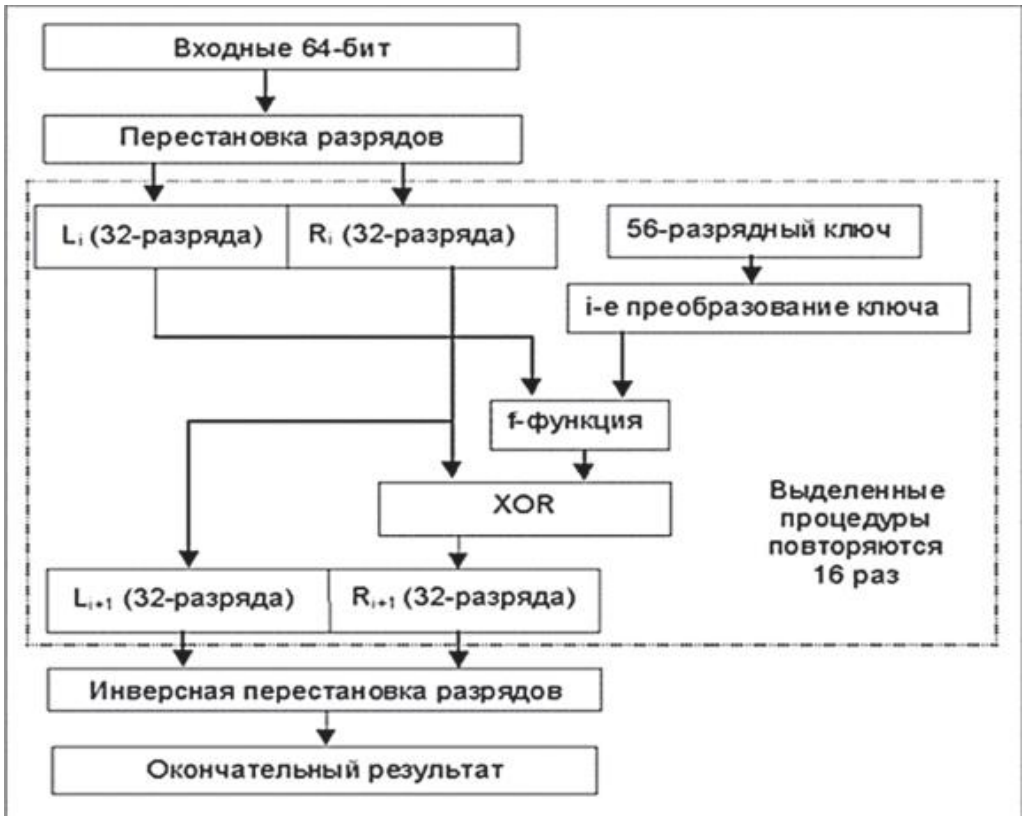


Рисунок 2.2 – Общая схема алгоритма DES

После выполнения перестановки с расширением для полученного 48-битного значения выполняется операция XOR с 48-битным подключом $K_i$. Затем полученное 48-битное значение подается на вход блока подстановки S (от англ. substitution –подстановка), результатом которой является 32-битное значение. Подстановка выполняется в восьми блоках подстановки или восьми *S*-блоках (*S*-boxes). При выполнении этой операции 48 битов данных делятся на восемь 6-битных подблоков, каждый из которых по соответствующей таблице замен замещается четырьмя битами. Подстановка с помощью *S*-блоков является одним из важнейших этапов DES. Таблицы замен для этой операции специально спроектированы так, чтобы обеспечивать максимальную криптостойкость. В

результате выполнения этого этапа получаются восемь 4-битных блоков, которые вновь объединяются в единое 32-битное значение.

Далее полученное 32-битное значение обрабатывается с помощью перестановки *P* (от англ. permutation – перестановка), которая не зависит от используемого ключа. Целью перестановки является такое максимальное переупорядочивание битов, чтобы в следующем раунде шифрования каждый бит с большой вероятностью обрабатывался другим *S*-блоком.

И наконец, результат перестановки объединяется с помощьюоперации XOR с левой половиной первоначального 64-битного блока данных. Затем левая и правая половины меняются местами, и начинается следующий раунд.


Рисунок 2.3 – Схема реализации функции *f*

После выполнения 16-раундового зашифрования 64-битного блока данных осуществляется конечная перестановка ($IP^{-1}$). Она является обратной к перестановке *IP*. И в конечном итоге получаем шифртекст.

При расшифровании на вход алгоритма подается зашифрованный текст. Единственное отличие состоит в обратном порядке использования частичных ключей $K_i$. Ключ $K_{16}$ используется в первом раунде, $K_1$ – в последнем.

После последнего раунда процесса расшифрования две половины выхода меняются местами так, чтобы вход заключительной перестановки был составлен из подблоков $R_{16}$ и $L_{16}$. Выходом этой стадии является расшифрованный текст.

Алгоритм Speck был разработан в 2013 году командой исследователей, включая Карлайл Адамса и Стаффорда Тавареса, и представляет собой блочный симметричный шифр, который использует эффективные методы для обеспечения безопасности данных. Speck отличается своей простотой и высокой производительностью, что делает его особенно привлекательным для использования в устройствах с ограниченными ресурсами, таких как встраиваемые системы.

Speck работает с блоками данных размером 64 бита и поддерживает различные длины ключей, варьирующиеся от 64 до 256 бит. Алгоритм включает в себя несколько раундов шифрования, число которых зависит от длины ключа: от 22 раундов для коротких ключей до 34 раундов для длинных. В процессе шифрования используются операции, такие как XOR, побитовые сдвиги и модулярная арифметика, что позволяет достигать высокого уровня безопасности.

Структура Speck включает в себя комбинацию различных функций, которые применяются в каждом раунде, что обеспечивает гибкость и надежность алгоритма. Его дизайн ориентирован на максимальную производительность, что делает его особенно полезным для мобильных устройств и других систем, где требуется быстрое шифрование данных.

Кроме того, Speck не защищен патентом, что позволяет свободно использовать его в коммерческих и некоммерческих проектах по всему миру. Это делает его доступным для широкого круга разработчиков и пользователей, что способствует его распространению и внедрению в различные приложения.

Таким образом, алгоритм Speck представляет собой современное решение для обеспечения безопасности данных, сочетая в себе высокую производительность и гибкость, что делает его важной частью современного криптографического ландшафта.

Speck представляет собой ARX шифр (использует сложение по модулю, циклический сдвиг и исключающее ИЛИ). Можно выбирать длину блока и ключа в зависимости от задачи. Блок разбит на два слова, длина ключа кратна длине слова. Все операции производятся над словами, сложение и вычитание выполняются по модулю 2 в степени размера слова. Раундовая функция похожа на Threefish и показана на рисунке 2.4.

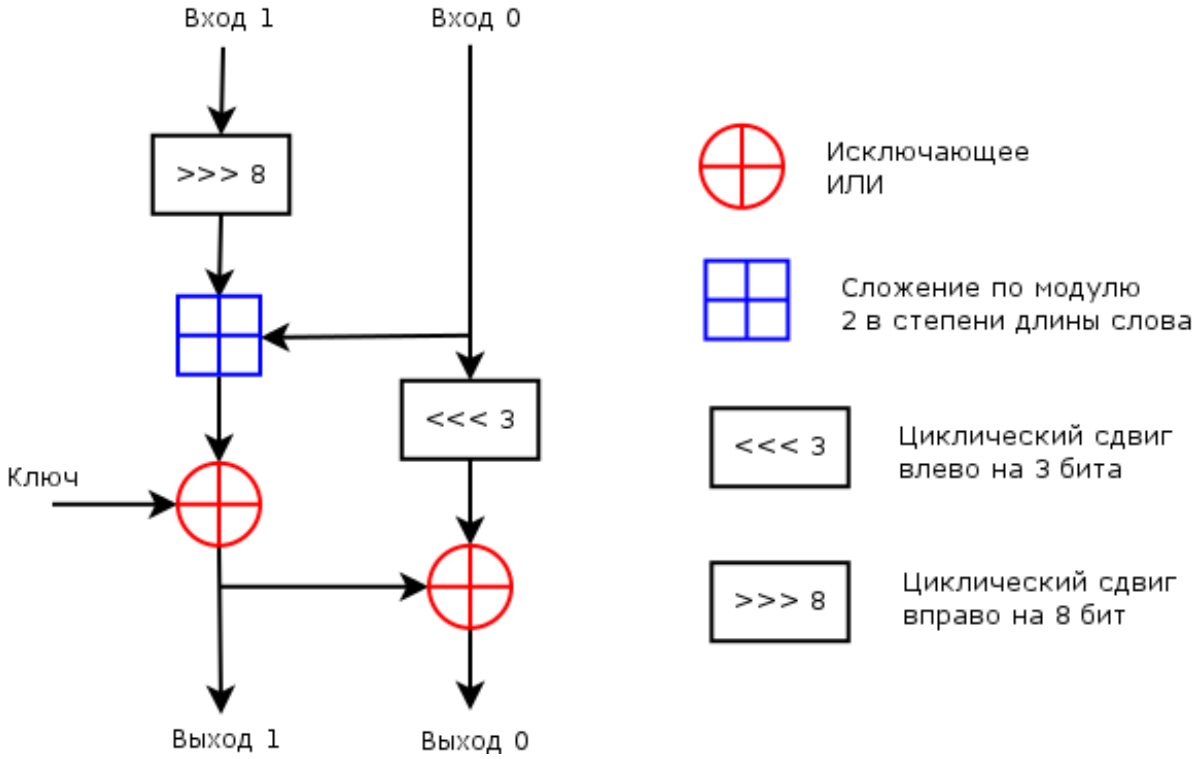


Рисунок 2.4 – Раундовая функция

В каждом раунде $2^n$-битных входа делятся на две $n$-битные половины. Каждый раунд Speck применяет операции конъюнкции, циклического сдвига влево и вправо, а также сложения по модулю 2n. Параметры имеют следующие значения: $\alpha = 7$ и $\beta = 2$, если $n = 16$ (размер блока равен 32) и $\alpha = 8$ и $\beta = 3$ в противном случае.

В шифре Speck, вводя на каждом раунде новые переменные, получим следующие количества уравнений и неизвестных:

$$e = \begin{cases} (7n-3)(T-1) + (8n-3)(T-1) + 2n \\ 2(8n-3)(T-1) + 2n \end{cases}$$

$$u = \begin{cases} n(5T-4) \\ n(6T-5) \end{cases}$$

В шифре Speck основным методом сохранения степени является введение новых переменных для выходных битов нелинейных операций. В этом случае степень не будет превышать 2. На каждом раунде вводится 28n новых переменных. В системе уравнений, описывающей сложение по модулю $2n$ , имеется всего $5(7n - 8)$ мономов. На практике оказалось, что различных мономов в системе уравнений сложения по модулю $2n$ не больше $25n - 18$. Таким, образом, количество различных мономов на каждом раунде шифра Speck не больше $28n - 18$. Итоговая оценка числа различных мономов, исключая такие, которые образуются при генерации ключей (все уравнения линейны), имеет вид:

$$M \leq (28n - 18)T$$

На рисунке 2.5 можно увидеть три раунда Speck, где длина ключа равна длине блока.

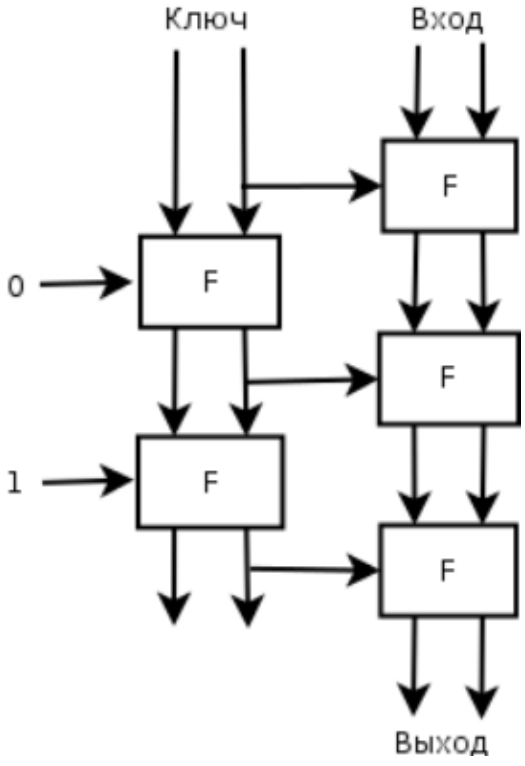


Рисунок 2.5 – Три раунда

Такой подход позволяет повторно использовать код раундовой функции и даёт дополнительную гибкость — если нужно оптимизировать скорость выполнения, то можно посчитать раундовые ключи заранее, а если хочется экономить память, то их можно считать на ходу. Второй вариант также удобнее, если надо шифровать один блок на новом ключе.

Если ключ длиннее блока, слова ключа используются по кругу. На рисунке 2.6 четыре раунда, дальше всё так же.

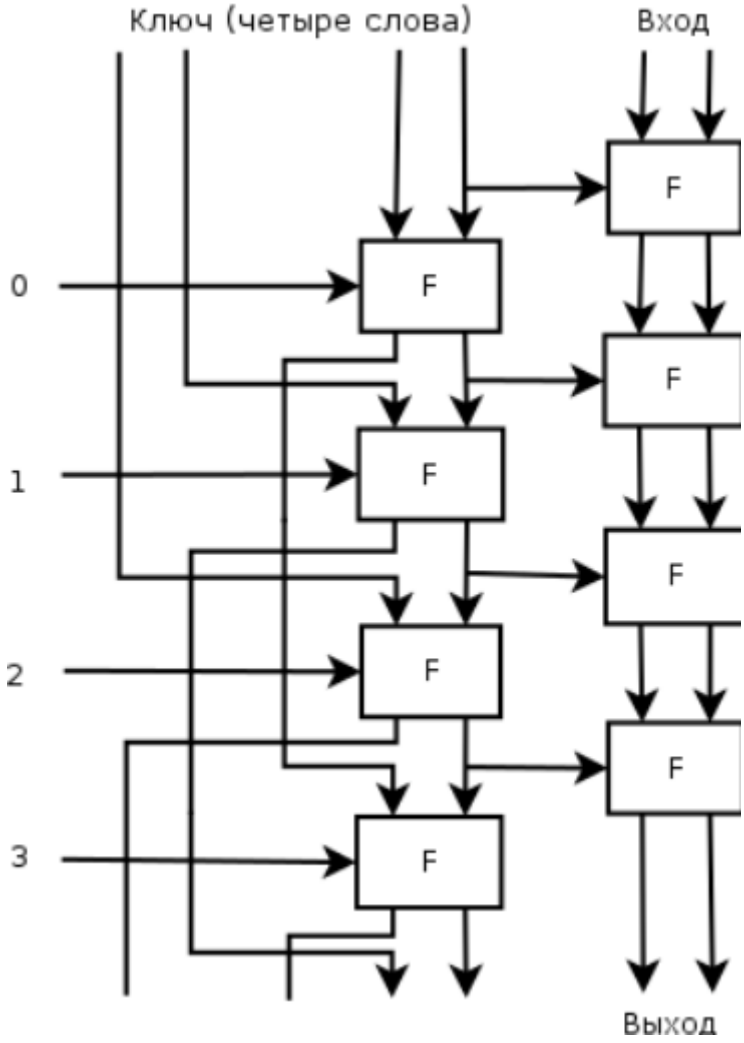


Рисунок 2.6 – Четыре раунда

Отличительными особенностями Speck является простота имплементации и малое количество используемой памяти. Нет ни магических констант, ни чёрных ящиков подстановки-перестановки. Нет даже отбеливания ключа, это делает первый раунд.

## 2.1 Вывод

В данном исследовании были рассмотрены два алгоритма симметричного шифрования: классический DES и современный Speck. DES, как один из первых широко распространенных стандартов шифрования, заложил основу для многих последующих разработок. Его структура, основанная на сети Фейстеля, обеспечивала высокий уровень безопасности для своего времени. Однако, с развитием вычислительной техники, длина ключа DES в 56 бит стала недостаточной для защиты от современных атак.

Speck, напротив, представляет собой более современный подход к шифрованию, сочетающий простоту реализации и высокую производительность. Его структура, основанная на небольшом наборе простых операций, позволяет эффективно использовать вычислительные ресурсы.

Сравнение этих двух алгоритмов демонстрирует эволюцию криптографии: от сложных и ресурсоемких конструкций, таких как DES, к более простым и эффективным, как Speck. Выбор между ними зависит от конкретных требований к безопасности, производительности и ресурсным ограничениям.

## 3 Описание программного средства

В данной курсовой работе было реализовано веб-приложение, реализующее блочно-симметричное шифрование и дешифрование с использованием алгоритмов DES и Speck.

Используются следующие языки и технологии:

– JavaScript:

JavaScript – это высокоуровневый язык программирования, который используется для создания интерактивных элементов на веб-страницах. Он поддерживается всеми современными браузерами и позволяет работать с DOM, обеспечивая динамическое изменение содержимого и структуры страниц. JavaScript также поддерживает асинхронное программирование, что позволяет выполнять код без блокировки выполнения.

– Node.js:

Node.js – это платформа, которая позволяет выполнять JavaScript на стороне сервера. Она основана на движке V8 от Google Chrome и предоставляет разработчикам возможность использовать JavaScript для создания серверного кода. Node.js использует асинхронную модель ввода-вывода, что позволяет обрабатывать множество запросов одновременно, эффективно используя ресурсы сервера.

– Express.js:

Express.js – это минималистичный и гибкий веб-фреймворк для Node.js, который упрощает разработку серверных приложений. Он предоставляет набор инструментов и функций для создания веб-приложений и API, позволяя разработчикам легко обрабатывать маршруты, запросы и ответы.

Помимо этого, были использованы и различные библиотеки:

– crypto_cours:

crypto_cours – это основная библиотека проекта (версия 1.0.0), которая предоставляет функции для работы с криптографией, включая шифрование и создание хешей. Она служит базой для других модулей и компонентов, обеспечивая безопасность данных в приложении.

– node-forge:

node-forge – это библиотека для Node.js, которая предоставляет инструменты для работы с криптографией и сетевыми протоколами. Она поддерживает различные функции, такие как шифрование, создание цифровых подписей, генерация ключей и работа с сертификатами.

– crypto-js:

crypto-js – это библиотека для работы с криптографией (версия 4.2.0), которая предоставляет инструменты для шифрования, создания хешей, а также для работы с различными алгоритмами, такими как AES и SHA. Она полезна для обеспечения конфиденциальности и целостности данных в приложениях.

– body-parser:

body-parser – это middleware для обработки тела запроса в Express. Он позволяет извлекать данные из тела запросов, таких как JSON и URL-кодированные данные, что упрощает работу с данными, отправляемыми клиентами.

– nvm:

nvm (Node Version Manager) – это инструмент для управления версиями Node.js. Он позволяет разработчикам устанавливать, обновлять и переключаться между различными версиями Node.js на одной машине. Это особенно полезно, когда разные проекты требуют разных версий Node.js.

– express-fileupload:

express-fileupload – это middleware для Node.js и Express, который упрощает процесс загрузки файлов через HTTP-запросы. Он позволяет обрабатывать файлы, отправленные в запросах, и предоставляет удобный интерфейс для работы с загруженными файлами, включая поддержку различных параметров, таких как переименование и контроль размера загружаемых файлов.

Структура разработанного приложения представлена на рисунке 3.1.

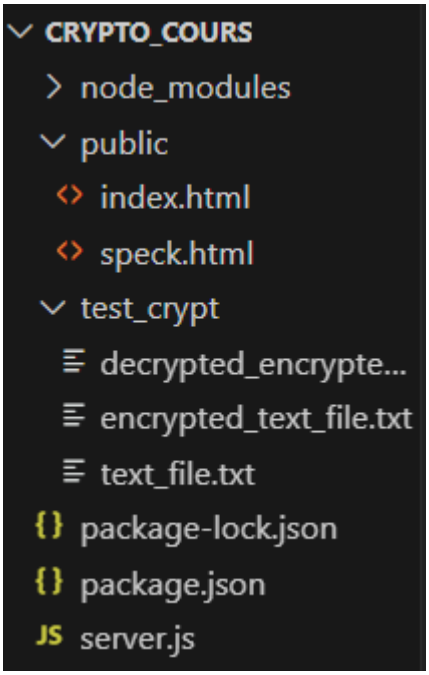


Рисунок 3.1 – Структура приложения

Ниже представлено описание структуры:

– server.js: Основной файл, содержащий настройки сервера и маршруты и функции для шифрования/расшифрования.

– public: папка с HTML-шаблонами.

– index.html: HTML файл, который представляет собой страницу с шифрованием/расшифрованием при помощи алгоритма DES.

– speck.html: HTML файл, который представляет собой страницу с шифрованием/расшифрованием при помощи алгоритма Speck.

Интерфейс включает в себя две html страницы, где пользователи могут ввести оригинальное сообщение или подгрузить файл и ключ для шифрования. Результаты отображаются под кнопками действий в виде двух строк. Первая строка отвечает за вывод результата операции, а вторая ведёт подсчёт времени выполнения в миллисекундах Это делает процесс анализа интуитивно понятным и

доступным для пользователей с различным уровнем подготовки. Представлено на рисунке 3.2.

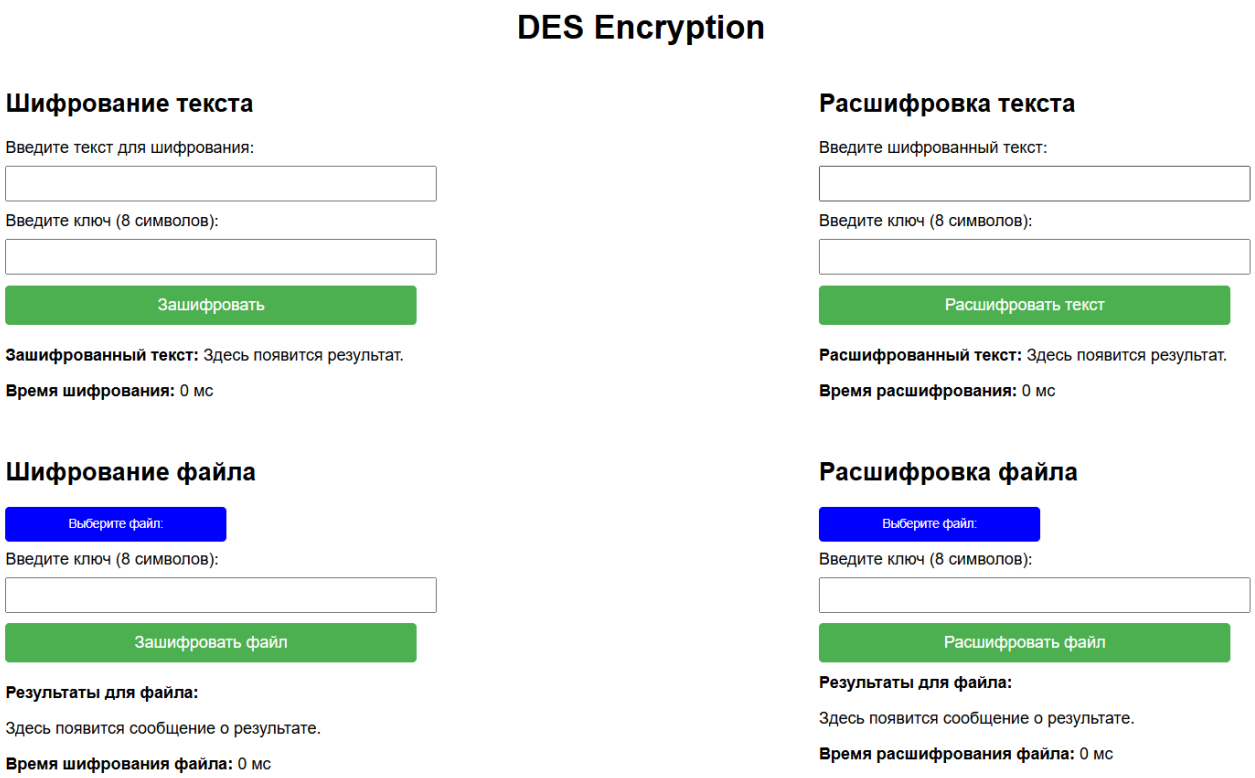


Рисунок 3.2 – Интерфейс приложения

Листинг функций расшифрования и шифрования алгоритмом DES представлен под пунктом 3.1 и 3.2 соответственно.

```
function decryptDES(encryptedText, key) {
  if (key.length !== 8) {
    // Генерируем случайную "белиберду", если ключ некорректный
    return generateRandomString(encryptedText.length);
  }

  try {
    const decipher = crypto.createDecipheriv('des-ecb',
Buffer.from(key, 'utf8'), null);
    let decrypted = decipher.update(encryptedText, 'hex', 'utf8');
    decrypted += decipher.final('utf8');
    return decrypted;
  } catch (err) {
    // Генерируем случайную строку в случае ошибки расшифровки
    return generateRandomString(encryptedText.length);
  }
}
```

Листинг 3.1 – Функция расшифрования алгоритмом DES

```
function encryptDES(text, key) {
  if (key.length !== 8) {
    throw new Error('Ключ должен быть длиной ровно 8 символов.');
  }

  const cipher = crypto.createCipheriv('des-ecb', Buffer.from(key,
'utf8'), null);
  let encrypted = cipher.update(text, 'utf8', 'hex');
  encrypted += cipher.final('hex');
  return encrypted;
}
```

Листинг 3.2 – Функция шифрования алгоритмом DES

Листинг функций шифрования и расшифрования алгоритмом Speck представлен под пунктом 3.3.

```
function rotl(x, n, bits) {
    let result = (x << n) | (x >>> (bits - n));
    return result & ((1 << bits) - 1);  // Убедитесь, что результат
в пределах битовой ширины
}

function encryptSpeck(plaintext, keys) {
    let [x, y] = plaintext;
    for (let round = 0; round < keys.length; round++) {
        x = rotr(x, 7, 16);
        x = (x + y) & 0xFFFF;
        x ^= keys[round];

        y = rotl(y, 2, 16);
        y ^= x;
    }

    return [x, y];
}
function decryptSpeck(ciphertext, keys) {
    let [x, y] = ciphertext;

    for (let round = keys.length - 1; round >= 0; round--) {
        y ^= x;
        y = rotl(y, 16 - 2, 16);

        x ^= keys[round];
        x = (x - y) & 0xFFFF;
        x = rotr(x, 16 - 7, 16);
    }

    return [x, y];
}
```

Листинг 3.3 – Функция шифрования и расшифрования алгоритмом Speck

На второй странице представлен алгоритм Speck. Пользователю предоставлен выбор способа шифрования. Либо ввод текста/шифртекста вручную или же загрузкой через файл. Представлено на рисунке 3.3.

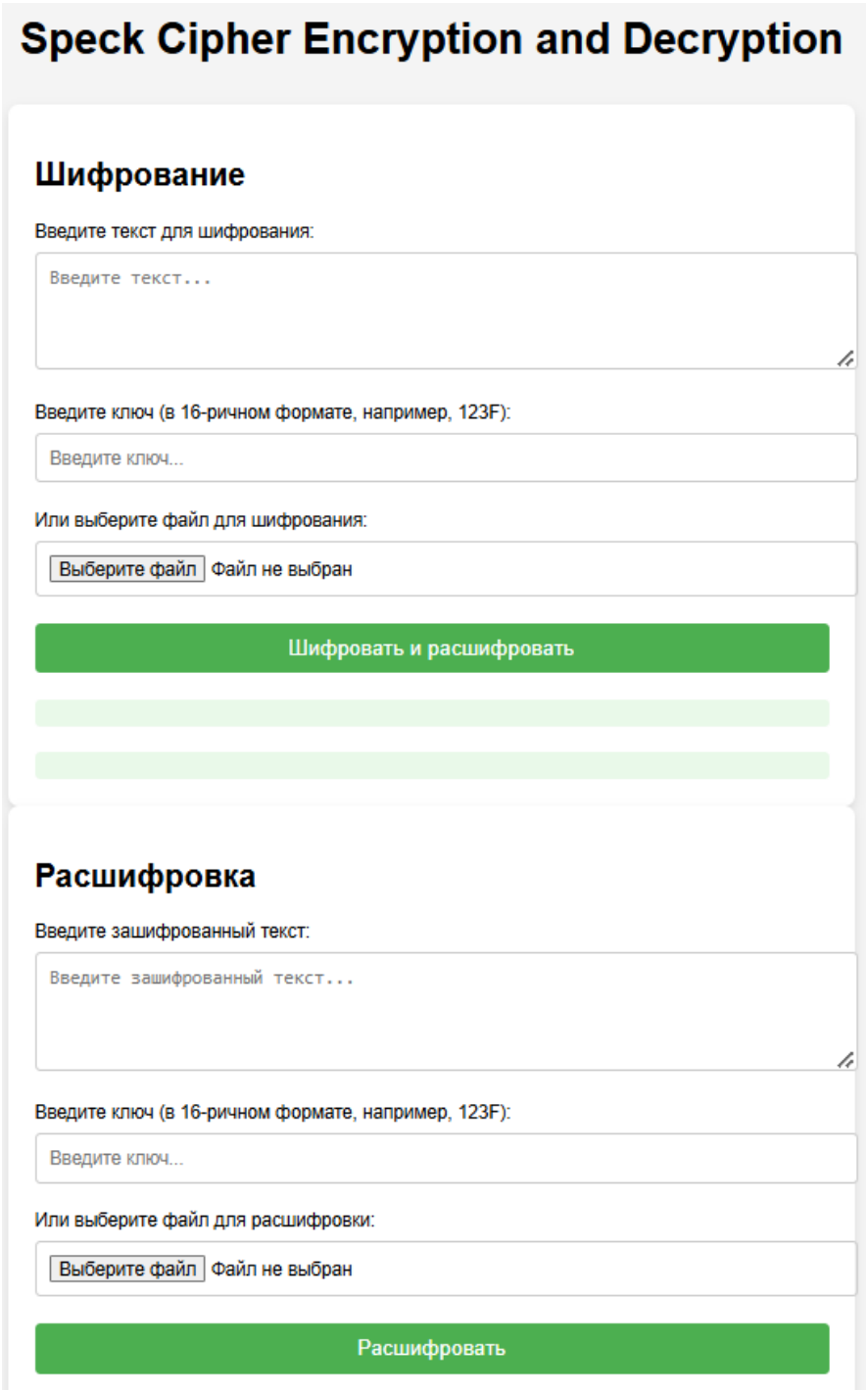


Рисунок 3.3 – Интерфейс приложения

В итоге, разработанное веб-приложение позволяет выполнять следующие действия:

– шифровать и расшифровывать текст с использованием двух различных алгоритмов;

– отображать результаты в удобном формате на веб-странице, где пользователи могут видеть: зашифрованный текст, расшифрованный текст и время, затраченное на каждую операцию.

### 3.1 Вывод

В рамках данной курсовой работы было разработано веб-приложение, которое реализует блочно-симметричное шифрование и дешифрование с применением алгоритмов DES и Speck. Это приложение построено на современных технологиях, таких как JavaScript, что делает его не только функциональным, но и удобным в использовании для конечного пользователя.

Благодаря использованию Node.js и Express.js, приложение способно эффективно обрабатывать серверные запросы, что значительно повышает его производительность и отзывчивость. В результате, данное приложение не только демонстрирует возможности современных технологий шифрования, но и предоставляет пользователю удобный инструмент для анализа производительности различных алгоритмов. Это делает его полезным как для изучения основ криптографии, так и для практического применения в реальных сценариях.

## 4 Тестирование программного средства

Чтобы протестировать разработанное программное средство, заполним нужные поля информацией. Текст может быть неограниченных размеров, а вот ключ не более 64 бит или же 8 символов в системе ASCII.

Как можно увидеть на рисунке 4.1 сообщение успешно шифруется и расшифровывается, а также выводится время затраченное на операцию.


Рисунок 4.1 – Результат работы алгоритма DES

Если же при расшифровании использовать неверный ключ, то сообщение будет расшифровано некорректно. Представлено на рисунке 4.2.

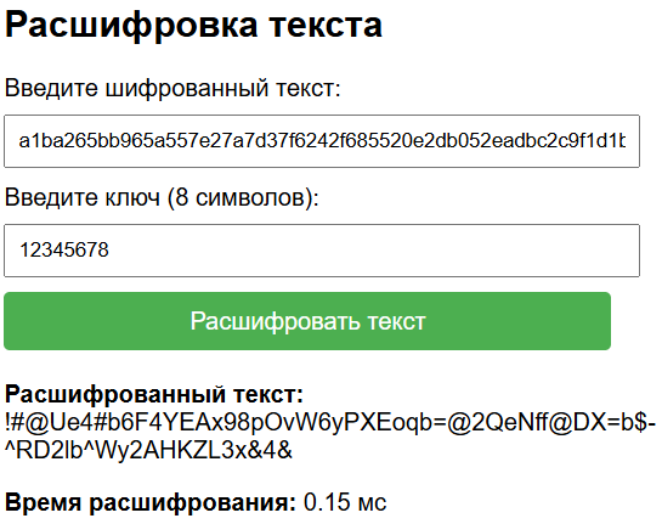


Рисунок 4.2 – Результат с неверным ключем

Точно также работает расшифровка с использованием файлов. Представлено на рисунке 4.3.

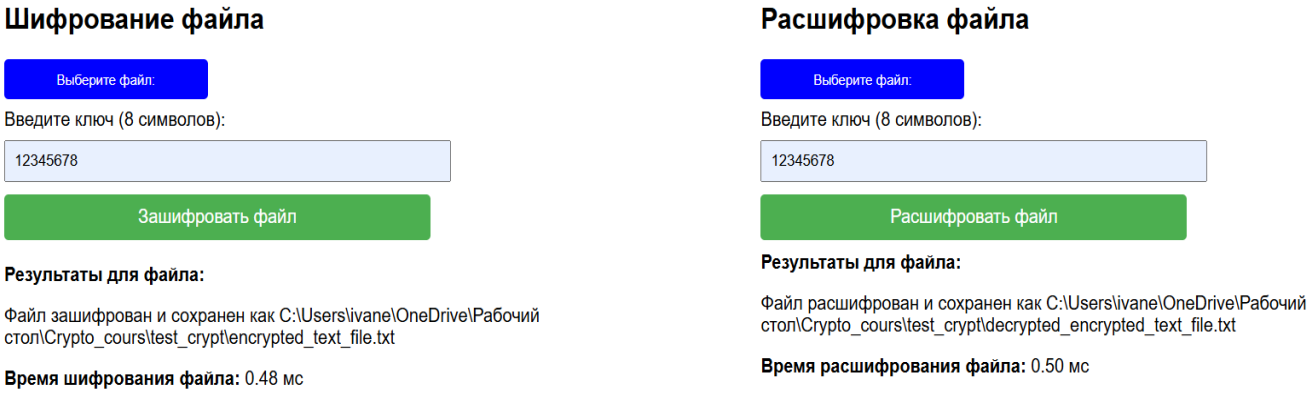


Рисунок 4.3 – Результат работы алгоритма DES с файлами

В случае если формат ключа указан неверно, то происходит обратно ошибки и у пользователя появляется всплывающее окно с необходимой информацией об ошибке. Представлено на рисунке 4.4.

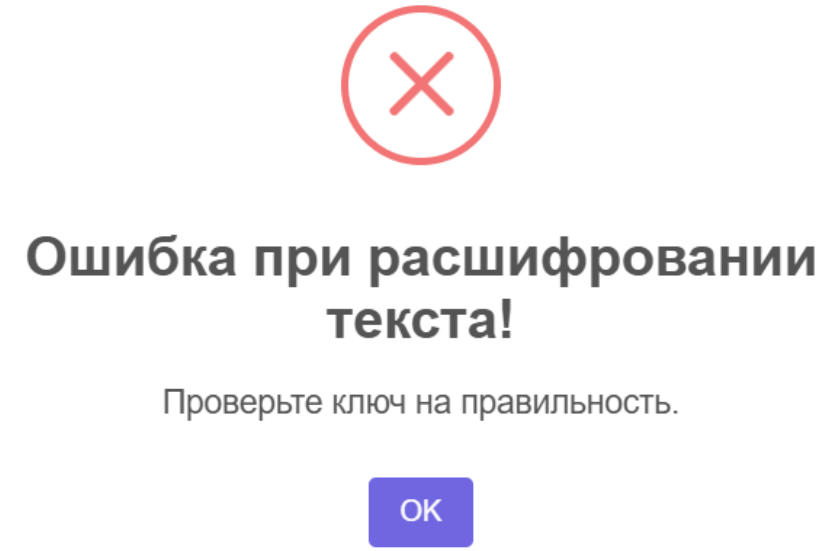


Рисунок 4.4 – Всплывающее окно с ошибкой

Алгоритм Speck выглядит точно также, только вместо четырех форм реализовано 2. И пользователю изначально дан выбор, вписывать данные самостоятельно или через файл. Представлено на рисунке 4.5.

Расшифрование Speck работает точно таким же способом. Через файл или же написанный текст. Обратно ошибок работает также как это реализовано в алгоритме DES, описанным выше.

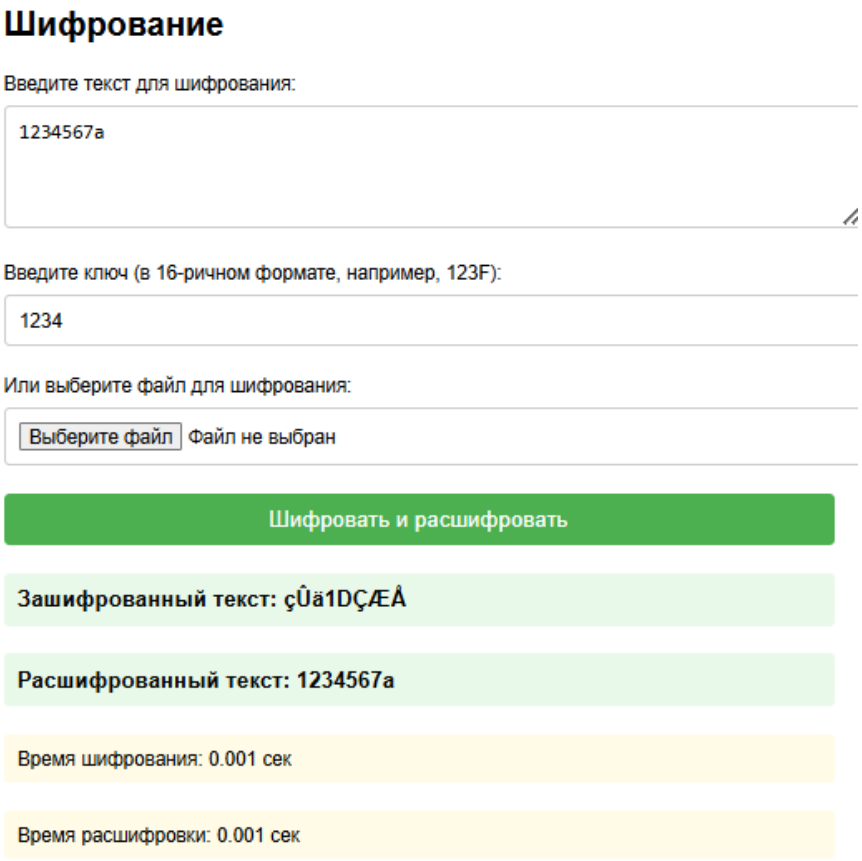


Рисунок 4.5 – Шифрование Speck

Также в Speck представлено время шифрования и расшифрования.

### 4.1 Вывод

В процессе тестирования разработанного приложения были успешно проверены все его функции. Каждое из действий шифрования и дешифрования выполнялось корректно, что обеспечивало целостность и безопасность обрабатываемых данных. Результаты проведенных испытаний подтверждают высокую универсальность приложения, а также его способность эффективно работать с текстовыми данными в различных сценариях. Это свидетельствует о надежности и стабильности приложения, что делает его ценным инструментом для пользователей, заинтересованных в надежном шифровании информации.

## 5 Сравнительный анализ методов

В ходе курсового проекта было проведено сравнение двух алгоритмов по нескольким критериям: скорости шифрования, скорости расшифрования, криптостойкости, ресурсоемкости и возможности распараллеливания.

Как уже упоминалось, DES (Data Encryption Standard) представляет собой классический алгоритм блочного шифрования, который работает с 64-битными блоками данных и использует ключи длиной 56 бит. Он основан на принципе итеративного блочного шифрования и включает в себя 16 раундов, в ходе которых входной блок данных проходит через последовательность сложных преобразований. Эти преобразования включают в себя операции перестановки, замены и побитового XOR с раундовыми ключами, которые генерируются из основного ключа. Несмотря на свою историческую значимость, DES в настоящее время считается устаревшим из-за своей уязвимости к атакам на основе полного перебора ключей.

Speck, с другой стороны, является современным блочным шифром, разработанным для эффективной работы как на аппаратных, так и на программных платформах. Speck поддерживает различные размеры блоков данных, начиная от 64 бит и заканчивая 128 бит, и может работать с ключами длиной от 64 до 256 бит. Алгоритм использует простую и быструю структуру, основанную на принципе итеративного шифрования с использованием раундовых ключей, которые также генерируются из основного ключа. Speck подходит для различных приложений, включая системы с ограниченными ресурсами, благодаря своей низкой ресурсоемкости и высокой скорости обработки данных..

Длина ключа в алгоритме DES составляет фиксированные 56 бит, что ограничивает количество возможных комбинаций и делает его уязвимым для атак. В то время как Speck позволяет выбирать длину ключа в диапазоне от 64 до 256 бит, что значительно увеличивает уровень безопасности и количество возможных ключей. Например, количество возможных ключей для DES составляет $2^{56}$, что значительно меньше, чем для Speck с длиной ключа 256 бит, что равняется $2^{256}$. Это делает Speck более подходящим для современных требований к безопасности в различных приложениях.

Используя разработанное приложение, сравним алгоритмы по скоростям шифрования и расшифрования учитывая разный размер текста.

Результаты времени шифрования и расшифрования алгоритмом DES показаны в таблице 1.

Таблица 1 – Скорость шифрования и расшифрования алгоритмом DES

| Размер текста (символы) | Время зашифрования (с) | Время расшифрования (с) |
|---|---|---|
| 10 | 0,09 | 0,12 |
| 100 | 0,08 | 0,13 |
| 500 | 0,10 | 0,17 |
| 1500 | 0,17 | 0,21 |
| 5000 | 0,44 | 0,49 |

Для более наглядной демонстрации, можем посмотреть на график, который представлен на рисунке 5.1.

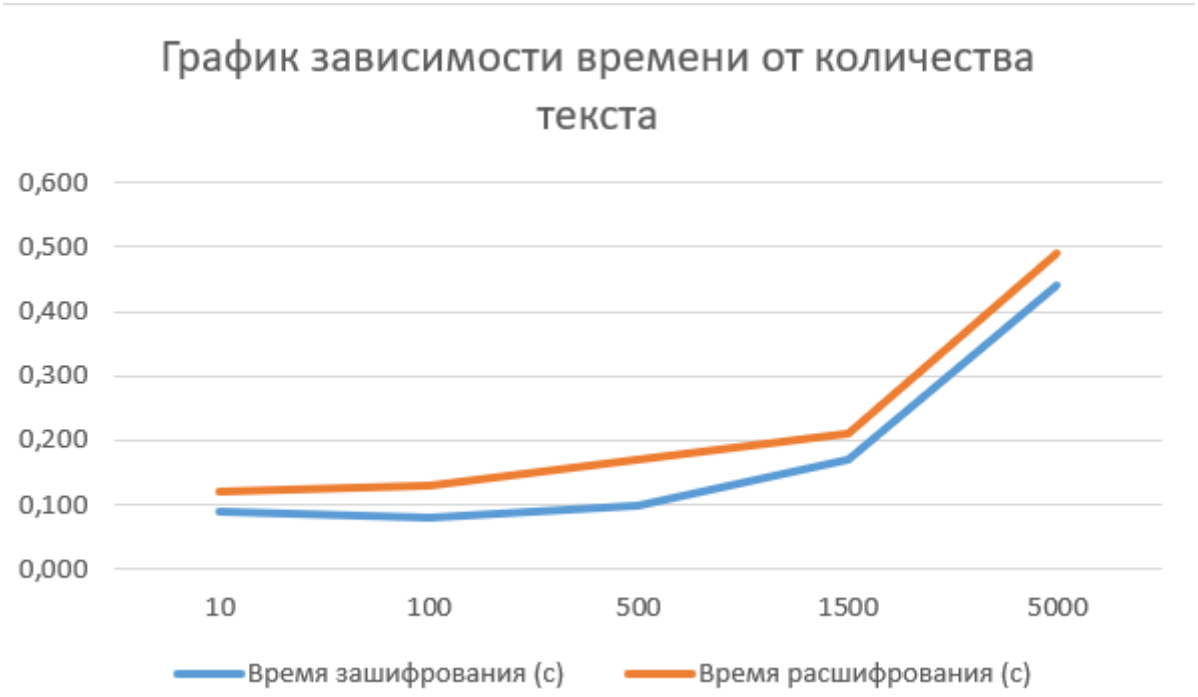


Рисунок 5.1 – График зависимости времени от объема текста для DES

Время шифрования увеличивается с ростом количества текста. Для 10 символов время шифрования составляет менее 0,100 с, а для текста размером 5000 символов около – 0,440 с.

Время расшифрования также растет с увеличением количества текста. Так, для 10 символов время расшифрования равно 0,120 с, а для 5000 Кб – 0,490 с.

Можно сделать вывод, что алгоритм XXTEA демонстрирует линейную зависимость времени шифрования и расшифрования от размера обрабатываемого текста. Чем больше размер текста, тем больше времени требуется на выполнение криптографических операций.

В целом, представленные данные показывают, что алгоритм DES может быть эффективным для шифрования небольших и средних объемов данных, но его производительность снижается при работе с большими объемами информации.

Также были проведены измерения времени для алгоритма Speck, результаты можно увидеть в таблице 2.

Таблица 2 – Скорость шифрования и расшифрования алгоритмом CAST

| Размер текста (символы) | Время зашифрования (с) | Время расшифрования (с) |
|---|---|---|
| 10 | 0,012 | 0,014 |
| 100 | 0,020 | 0,015 |
| 500 | 0,080 | 0,089 |
| 1500 | 0,102 | 0,115 |
| 2000 | 0,21 | 0,20 |

График зависимости времени шифрования и расшифрования от объема текста для XXTEA представлен на рисунке 5.2.

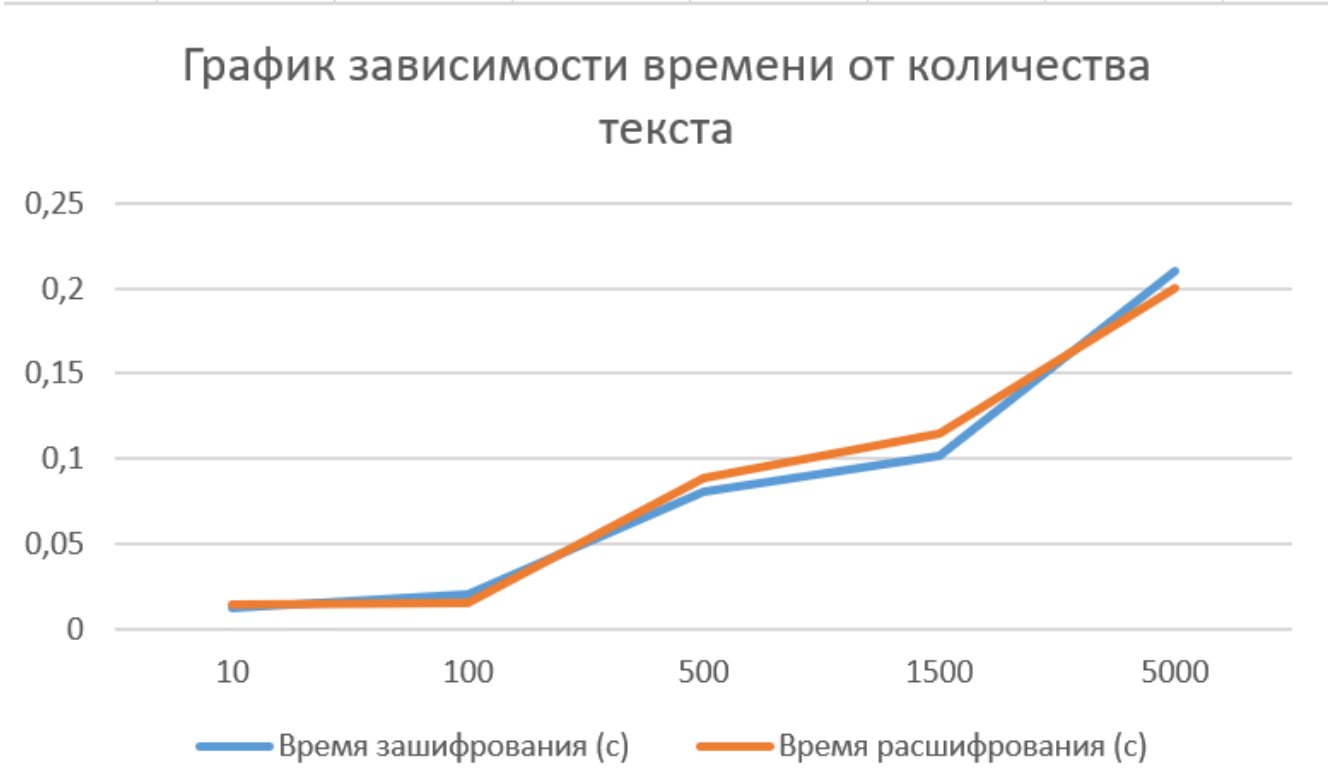


Рисунок 5.2 – График зависимости времени от объема текста для Speck

Алгоритм Speck был разработан с учётом оптимизации для работы на устройствах с ограниченными вычислительными ресурсами, например, на мобильных устройствах, встроенных системах или микроконтроллерах. Он представляет собой довольно быстрый блочный шифр с небольшой длиной ключа и эффективными операциями.

Несмотря на то, что с увеличением объёма данных можно ожидать замедление работы шифрования и расшифровки, некоторые особенности алгоритма и его реализации могут объяснить, почему мы не наблюдаем значительного замедления при увеличении данных, как это видно в DES.

Сравнивая результаты для DES и Speck, можно заметить, что алгоритм DES требует более длительного времени как на шифрование, так и на расшифрование по сравнению с Speck для одинаковых объемов данных. Это связано с тем, что DES является более сложным и с более длинным ключем, Speck использует простые операции по типу битовых сдвигов, сложений и XOR.

Теперь сравним размеры шифртекстов. Результаты представлены в таблице 3.

Таблица 3 – Анализ длины шифртекстов

| Размер текста (символов) | Шифртекст DES (символов) | Шифртекст Speck (символов) |
|---|---|---|
| 10 | 32 | 16 |
| 100 | 208 | 123 |

| | | |
|---|---|---|
| 500 | 1008 | 597 |
| 1500 | 3008 | 1553 |
| 5000 | 10016 | 5925 |

Как можно заметить, размер шифртекста DES увеличился в два раза. А шифр текст в алгоритме Speck практически остался таким же. Таким образом, видно, что в алгоритме Speck длина шифртекста пропорциональна размеру исходного текста, а в алгоритме DES практически двухкратное увеличение шифртекста. График с зависимости представлен на рисунке 5.3.


Рисунок 5.3 – График анализа длинны шифртекста

Воспользуемся программным средством и оценим лавинный эффект. Результаты продемонстрированы в таблице 4.

Таблица 4 – Анализ лавинного эффекта

| Текст | Шифр DES | Шифр Speck |
|---|---|---|
| abcd | b1e1429ad11b22fb | 21d0 6a29 |
| abcx | f0a822900b7f7541 | 111b 10d6 |
| abxd | 9213e71416fad437 | 18cf c484 |
| axcd | 3440d7733c1e6954 | 9717 912a |
| xbcd | e1e3ba44948f3814 | 0aa2 4b40 |

Взяв одинаковые ключи и проведя процесс шифрования, невооруженным глазом видно, что при смене даже одного символа (8 бит) результат значительно меняется. Это говорит нам о том, что оба алгоритма обладают высокой степенью лавинного эффекта, что делает их устойчивыми к различным атакам, связанным с изменением входных данных.

Оценим криптостойкость обоих алгоритмов. Результаты продемонстрированы в таблице 5.

Таблица 5 – Сравнение криптостойкости DES и Speck

| Характеристика | DES | Speck |
|---|---|---|
| Длина ключа | 56 бит | 64, 128, 256 бит |
| Уязвимость к перебору | Уязвим к атаке полного перебора | Устойчив к атаке полного перебора при длинных ключах |
| Дифференциальный криптоанализ | Уязвим (существуют дифференциальные пары) | Меньше уязвим, но возможны атаки при коротких ключах |
| Линейный криптоанализ | Уязвим (существуют линейные зависимости) | Меньше уязвим, сложнее атаки |
| Производительность | Низкая (по современным меркам) | Высокая, особенно на аппаратных решениях |
| Рекомендации | Не рекомендуется для использования в современных системах | Безопасен с длинными ключами и большим числом раундов |

Таким образом, по криптостойкости DES устарел и не рекомендуется для использования в современных системах. Его короткая длина ключа и уязвимости к атакам криптоанализа делают его неподобающим для защиты данных. В то время как Speck является более безопасным и эффективным алгоритмом, особенно при использовании длинных ключей и большого числа раундов. Он обеспечивает высокую производительность и криптографическую стойкость, что делает его хорошим выбором для устройств с ограниченными ресурсами.

Возможности распараллеливания шифровальных алгоритмов зависят от их внутренней структуры и принципов работы. Рассмотрим, как распараллеливание возможно в алгоритмах DES и Speck, а также их особенности в контексте многозадачности.

DES (Data Encryption Standard) является блочным шифром с фиксированным размером блока данных в 64 бита и длиной ключа 56 бит. Он использует структуру сети Фейстеля, где шифрование каждого блока выполняется за несколько раундов, и результат каждого раунда зависит от предыдущего. В частности, в DES используется 16 раундов, и каждый раунд шифрования зависит от результатов предыдущего, что делает алгоритм трудно распараллеливаемым на уровне обработки блоков данных.

Speck, в отличие от DES, был разработан для повышения производительности, особенно в контексте аппаратных реализаций на таких платформах, как FPGA или ASIC. Это блочный шифр, использующий сеть Фейстеля, но с упрощенной структурой и меньшей зависимостью между раундами, что делает его более подходящим для распараллеливания. В Speck каждый раунд шифрования зависит не от предыдущего раунда, а от значений, которые используются в текущем раунде и ключах, что облегчает параллельную обработку на уровне раундов.

### 5.1 Вывод

В данном разделе проведено сравнение алгоритмов шифрования DES и Speck по критериям скорости шифрования, скорости расшифрования, криптостойкости, ресурсоемкости и возможности распараллеливания. DES, показал линейную зависимость времени шифрования и расшифрования от размера данных, что делает

его эффективным для обработки малых объемов информации. Алгоритм Speck, занимал ещё меньше времени чем его конкурент DES, особенно, это заметно при больших объёмах данных. Результаты показали, что алгоритм DES увеличивает размер шифртекста в два раза, а Speck увеличивает лишь на малую часть, а также продемонстрировали их устойчивость к изменениям входных данных, что свидетельствует о наличии лавинного эффекта. Важно отметить, что Speck может быть предпочтительным для приложений с ограниченными ресурсами, в то время как DES подходит для сценариев, требующих более высокой безопасности, т.к длинна ключа больше. Выбор между этими алгоритмами зависит от конкретных требований к скорости и уровню защиты данных.

## Заключение

В рамках данного курсового проекта был проведен сравнительный анализ производительности и безопасности блочно-симметричных алгоритмов шифрования DES и Speck. Speck — это симметричный блочный шифр, известный своей высокой эффективностью и простотой реализации. DES — это также симметричный блочный шифр, который был разработан для обеспечения безопасности данных.

– Для этой задачи было разработано приложение, использующее различные технологии, такие как: JavaScript, Node.js и Express.js. Кроме того, применяются и библиотеки: node-forge, crypto, nvm, crypto-js: express-fileupload.

В итоге приложение выполняет следующие функции:

– шифрование и расшифрование текстовых сообщений;

– вывод времени, затраченного на шифрование и расшифрование, что позволяет оценить производительность каждого из алгоритмов;

– предоставляет удобный интерфейс для ввода данных и отображения результатов.

В ходе анализа алгоритмов, были проверены различные аспекты, такие как:

– зависимость времени шифрования и расшифрования от объемов шифруемого текста;

– лавинный эффект;

– криптостойкость шифров.

– зависимость размера шифртекста от исходного текста

Анализ выявил, что Speck более эффективен для обработки небольших объемов данных, обеспечивая быструю скорость шифрования и расшифрования, в то время как DES демонстрирует более высокую безопасность за счёт ключа и большую популярность, но требует больше времени для выполнения операций. Оба алгоритма имеют высокий уровень лавинного эффекта, что повышает их устойчивость к атакам. Несмотря на ограничения в распараллеливании, алгоритм Speck может быть адаптирован для многопоточной работы, что улучшает его производительность при шифровании больших объемов информации и делает его более предпочтительным, чем DES.

В конечном итоге, выбор между DES и Speck зависит от конкретных требований к безопасности и вычислительным ресурсам, что делает их подходящими для разных сценариев использования. Выбор алгоритма должен основываться на тщательном анализе нужд приложения и условий эксплуатации.

## Список используемых источников

1 Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие. – Минск: БГТУ, 2016. – 220 с.

2 Шифрование Speck [Электронный ресурс]. Режим доступа: https://www.mathnet.ru/links/a0aa7fe92c5126eeb1ca36851f62d1b7/pdma538.pdf. Дата доступа: 12.10.2024.

3 Урбанович, П. П. Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 1. Кодирование информации: учеб.-метод. пособие для студентов учреждений высшего образования / П. П. Урбанович, Д. В. Шиман, Н. П. Шутько. – Минск: БГТУ, 2019. – 116 с

4 Шифрование DES [электронный ресурс]. Режим доступа: https://www.opennet.ru/docs/RUS/inet_book/6/des_641.html. Дата доступа: 13.11.2024.

5 Шифрование Speck. [электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/sravnitelnyi-analiz-legkovesnyh-blochnyh-algoritmov-shifrovaniya-nash-i-speck-ispolzuemyh-v-ustroi-stvah-s-ogranichennymi. Дата доступа: 17.11.2024.

# ПРИЛОЖЕНИЕ А

## Листинг файла server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const crypto = require('crypto');
const path = require('path');
const fileUpload = require('express-fileupload');
const fs = require('fs');

const app = express();
const port = 3000;



// Middleware для обработки JSON и формы
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(fileUpload());

// Указываем статическую папку, где лежат HTML-файлы
app.use(express.static(path.join(__dirname, 'public')));

// Новый маршрут для отображения страницы Speck Cipher
app.get('/speck-cipher', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'speck.html'));
});



function encryptDES(text, key) {
  if (key.length !== 8) {
    throw new Error('Ключ должен быть длиной ровно 8 символов.');
  }

  const cipher = crypto.createCipheriv('des-ecb', Buffer.from(key, 'utf8'), null);
  let encrypted = cipher.update(text, 'utf8', 'hex');
  encrypted += cipher.final('hex');
  return encrypted;
}

// Генератор случайной строки (для "белиберды")
function generateRandomString(length) {
  const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+=-';
  let result = '';
  for (let i = 0; i < length; i++) {
    result += characters.charAt(Math.floor(Math.random() * characters.length));
  }
```

```
    return result;
  }

  function decryptDES(encryptedText, key) {
    if (key.length !== 8) {
      // Генерируем случайную "белиберду", если ключ некорректный
      return generateRandomString(encryptedText.length);
    }

    try {
      const decipher = crypto.createDecipheriv('des-ecb', Buffer.from(key, 'utf8'), null);
      let decrypted = decipher.update(encryptedText, 'hex', 'utf8');
      decrypted += decipher.final('utf8');
      return decrypted;
    } catch (err) {
      // Генерируем случайную строку в случае ошибки расшифровки
      return generateRandomString(encryptedText.length);
    }
  }

  // Маршрут для шифрования текста
  app.post('/encrypt', (req, res) => {
    const { text, key } = req.body;
    console.log(req.body);

    if (!text || !key || key.length !== 8) {
      return res.status(400).json({ error: 'Текст и ключ (ровно 8 символов) обязательны.' });
    }

    try {
      const startEncrypt = process.hrtime();
      const encrypted = encryptDES(text, key);
      const encryptTime = process.hrtime(startEncrypt);

      const startDecrypt = process.hrtime();
      const decrypted = decryptDES(encrypted, key); // В случае неверного ключа вернется случайная строка
      const decryptTime = process.hrtime(startDecrypt);

      res.json({
        encrypted,
        encryptTimeMs: (encryptTime[0] * 1e3 + encryptTime[1] / 1e6).toFixed(2),
        decryptTimeMs: (decryptTime[0] * 1e3 + decryptTime[1] / 1e6).toFixed(2),
        decrypted,
      });
    } catch (err) {
      console.error(err);
      res.status(500).json({ error: 'Ошибка при обработке данных. Проверьте входные данные.' });
```

```
      }
    });

    // Новый маршрут для расшифровки текста
    app.post('/decrypt', (req, res) => {
      const { encryptedText, key } = req.body;
      console.log(req.body);

      if (!encryptedText || !key || key.length !== 8) {
        return res.status(400).json({ error: 'Шифрованный текст и
ключ (ровно 8 символов) обязательны.' });
      }

      const startDecrypt = process.hrtime();
      const decrypted = decryptDES(encryptedText, key); // В случае
неверного ключа вернется случайная строка
      const decryptTime = process.hrtime(startDecrypt);

      res.json({
        decrypted,
        decryptTimeMs: (decryptTime[0] * 1e3 + decryptTime[1] /
1e6).toFixed(2),
      });
    });

    // Маршрут для шифрования файла
    app.post('/encrypt-file', (req, res) => {
      const { key } = req.body;
      const file = req.files.file;

      if (!file || key.length !== 8) {
        return res.status(400).json({ error: 'Файл и ключ (ровно 8
символов) обязательны.' });
      }

      try {
        const content = file.data.toString('utf8');
        const startEncrypt = process.hrtime();
        const encrypted = encryptDES(content, key);
        const encryptTime = process.hrtime(startEncrypt);

        const saveDir = path.join('C:', 'Users', 'ivane',
'OneDrive', 'Рабочий стол', 'Crypto_cours', 'test_crypt');

        if (!fs.existsSync(saveDir)) {
          fs.mkdirSync(saveDir, { recursive: true });
        }

        const filePath = path.join(saveDir, 'encrypted_' +
file.name);
        fs.writeFileSync(filePath, encrypted);

        res.json({
          message: `Файл зашифрован и сохранен как ${filePath}`,
```

```
          encryptTimeMs: (encryptTime[0] * 1e3 + encryptTime[1] /
1e6).toFixed(2),
        });
      } catch (err) {
        console.error(err);
        res.status(500).json({  error:  'Ошибка  при  шифровании
файла.' });
      }
    });

    // Маршрут для расшифровки файла
    app.post('/decrypt-file', (req, res) => {
      const { key } = req.body;
      const file = req.files.file;

      if (!file || key.length !== 8) {
        return res.status(400).json({ error: 'Файл и ключ (ровно 8
символов) обязательны.' });
      }

      try {
        const encryptedContent = file.data.toString('utf8');
        const startDecrypt = process.hrtime();
        const decrypted = decryptDES(encryptedContent, key); // В
случае неверного ключа вернется случайная строка
        const decryptTime = process.hrtime(startDecrypt);

        const  saveDir  =  path.join('C:',  'Users',  'ivane',
'OneDrive', 'Рабочий стол', 'Crypto_cours', 'test_crypt');

        if (!fs.existsSync(saveDir)) {
          fs.mkdirSync(saveDir, { recursive: true });
        }

        const  filePath  =  path.join(saveDir,  'decrypted_'  +
file.name);
        fs.writeFileSync(filePath, decrypted);

        res.json({
          message: `Файл расшифрован и сохранен как ${filePath}`,
          decryptTimeMs: (decryptTime[0] * 1e3 + decryptTime[1] /
1e6).toFixed(2),
        });
      } catch (err) {
        console.error(err);
        res.status(400).json({  error:  'Ошибка  при  расшифровке
файла! Проверьте ключ на правильность!' });
      }
    });

    // Запуск сервера
    app.listen(port, () => {
      console.log(`Сервер запущен на http://localhost:${port}`);
});
```

# ПРИЛОЖЕНИЕ Б

## Листинг файла index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DES Encryption</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      padding: 0;
    }
    .container {
      display: flex;
      flex-wrap: wrap;
      justify-content: flex-end; /* Прижать блоки к правой стороне */
    }
    .form-container {
      width: 48%; /* Две колонки */
      box-sizing: border-box;
      margin-bottom: 20px;
      padding-left: 140px; /* Добавить отступ слева для визуального разделения */
    }
    form {
      margin: 20px 0;
    }
    label {
      display: block;
      margin-bottom: 8px;
    }
    input, button {
      padding: 8px;
      margin-bottom: 10px;
      display: block;
      width: 100%;
      max-width: 400px;
    }
    #result {
      margin-top: 20px;
      padding: 10px;
      border: 1px solid #ccc;
      background-color: #f9f9f9;
    }
    #result p {
      margin: 5px 0;
    }
```

```
        #head {
          text-align: center;
        }
        button {
                width: 100%;
                padding: 10px;
                background-color: #4CAF50;
                color: #fff;
                border: none;
                border-radius: 4px;
                font-size: 16px;
                cursor: pointer;
            }
            /* Стили для label, чтобы он выглядел как кнопка */
    label[for="encryptFile"] {
        display: inline-block;
        width: 30%;
        padding: 10px;
        background-color: blue;
        color: #fff;
        border: none;
        border-radius: 4px;
        font-size: 12px;
        cursor: pointer;
        text-align: center;
    }
    label[for="decryptFile"] {
        display: inline-block;
        width: 30%;
        padding: 10px;
        background-color: blue;
        color: #fff;
        border: none;
        border-radius: 4px;
        font-size: 12px;
        cursor: pointer;
        text-align: center;
    }

    /* Скрытие реального input[type="file"] */
    input[type="file"] {
        display: none;
    }
      </style>
    </head>
    <body>
      <h1 id="head">DES Encryption</h1>

      <div class="container">
        <!-- Форма для шифрования текста -->
        <div class="form-container">
          <form id="encryptionForm">
            <h2>Шифрование текста</h2>
            <label for="text">Введите текст для шифрования:</label>
```

```
            <input type="text" id="text" name="text" required>

            <label for="key">Введите ключ (8 символов):</label>
            <input  type="text"  id="key"  name="key"  maxlength="8"
required>

            <button type="submit">Зашифровать</button>
          </form>

          <div id="textResult">
            <p><strong>Зашифрованный     текст:</strong>     <span
id="encryptedText">Здесь появится результат.</span></p>
            <p><strong>Время      шифрования:</strong>      <span
id="encryptTime">0</span> мс</p>
          </div>
        </div>

        <!-- Форма для расшифровки текста -->
        <div class="form-container">
          <form id="decryptionForm">
            <h2>Расшифровка текста</h2>
            <label   for="encryptedTextInput">Введите   шифрованный
текст:</label>
            <input       type="text"       id="encryptedTextInput"
name="encryptedText" required>

            <label      for="decryptKey">Введите      ключ      (8
символов):</label>
            <input    type="text"    id="decryptKey"    name="key"
maxlength="8" required>

            <button type="submit">Расшифровать текст</button>
          </form>

          <div id="decryptionResult">
            <p><strong>Расшифрованный    текст:</strong>    <span
id="decryptedTextOutput">Здесь появится результат.</span></p>
            <p><strong>Время     расшифрования:</strong>     <span
id="decryptTimeOutput">0</span> мс</p>
          </div>
        </div>
      </div>

      <div class="container">
        <!-- Форма для шифрования файла -->
        <div class="form-container">
          <form    id="encryptFileForm"    enctype="multipart/form-
data">
            <h2>Шифрование файла</h2>
            <label for="encryptFile">Выберите файл:</label>
            <input   type="file"   id="encryptFile"   name="file"
required>
```

```
            <label for="encryptKey">Введите ключ (8
символов):</label>
            <input type="text" id="encryptKey" name="key"
maxlength="8" required>

            <button type="submit">Зашифровать файл</button>
          </form>

          <div id="fileResult">
            <strong>Результаты для файла:</strong>
            <p id="fileMessage">Здесь появится сообщение о
результате.</p>
            <p><strong>Время шифрования файла:</strong> <span
id="fileEncryptTime">0</span> мс</p>
          </div>
        </div>

        <!-- Форма для расшифровки файла -->
        <div class="form-container">
          <form id="decryptFileForm" enctype="multipart/form-
data">
            <h2>Расшифровка файла</h2>
            <label for="decryptFile">Выберите файл:</label>
            <input type="file" id="decryptFile" name="file"
required>

            <label for="decryptKey2">Введите ключ (8
символов):</label>
            <input type="text" id="decryptKey2" name="key"
maxlength="8" required>

            <button type="submit">Расшифровать файл</button>
            <strong>Результаты для файла:</strong>
            <p id="fileMessage2">Здесь появится сообщение о
результате.</p>
            <p><strong>Время расшифрования файла:</strong> <span
id="fileDecryptTime">0</span> мс</p>
          </form>
        </div>
      </div>
      <script
src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
      <script>
        // Обработка формы для шифрования текста
        const textForm = document.getElementById('encryptionForm');
        textForm.addEventListener('submit', async (e) => {
          e.preventDefault();

          const text = document.getElementById('text').value;
          const key = document.getElementById('key').value;

          if (key.length !== 8) {
            return Swal.fire({
            icon: 'error',
```

```
            title: 'Ошибка при шифровании текста!',
            text: 'Проверьте ключ на правильность.',
            });;
          }

          try {
            const response = await fetch('/encrypt', {
              method: 'POST',
              headers: { 'Content-Type': 'application/json' },
              body: JSON.stringify({ text, key })
            });
            const data = await response.json();

            if (data.error) {
              alert(data.error);
            } else {
              document.getElementById('encryptedText').textContent
= data.encrypted;
              document.getElementById('encryptTime').textContent =
data.encryptTimeMs;
              document.getElementById('decryptedText').textContent
= data.decrypted;
              document.getElementById('decryptTime').textContent =
data.decryptTimeMs;
            }
          } catch (err) {
            console.log(`!!!${err}`);
          }
        });

        // Обработка формы для расшифровки текста
        const                    decryptForm                    =
document.getElementById('decryptionForm');
        decryptForm.addEventListener('submit', async (e) => {
          e.preventDefault();

          const                    encryptedText                  =
document.getElementById('encryptedTextInput').value;
          const key = document.getElementById('decryptKey').value;

          if (key.length !== 8) {
            return Swal.fire({
            icon: 'error',
            title: 'Ошибка при шифровании текста!',
            text: 'Проверьте ключ на правильность.',
            });;
            ;
          }

          try {
            const response = await fetch('/decrypt', {
              method: 'POST',
              headers: { 'Content-Type': 'application/json' },
              body: JSON.stringify({ encryptedText, key })
```

```
            });
            const data = await response.json();

            if (data.error) {
              alert(data.error);
            } else {

document.getElementById('decryptedTextOutput').textContent =
data.decrypted;

document.getElementById('decryptTimeOutput').textContent =
data.decryptTimeMs;
            }
          } catch (err) {
            console.log(err);
          }
        });

        // Обработка формы для шифрования файла
        const encryptFileForm =
document.getElementById('encryptFileForm');
        encryptFileForm.addEventListener('submit', async (e) => {
          const key = document.getElementById('encryptKey').value;
          e.preventDefault();
          if(key.length !== 8){
            console.log('tut');
            return Swal.fire({
            icon: 'error',
            title: 'Ошибка при шифровании текста!',
            text: 'Проверьте ключ на правильность.',
            });
          }
          const formData = new FormData(encryptFileForm);
          try {
            const response = await fetch('/encrypt-file', {
              method: 'POST',
              body: formData,
            });
            const data = await response.json();

            if (data.error) {
              alert(data.error);
            } else {
              document.getElementById('fileMessage').textContent =
data.message;

document.getElementById('fileEncryptTime').textContent =
data.encryptTimeMs;
            }
          } catch (err) {
            console.log(err);
          }
        });
```

```
        // Обработка формы для расшифровки файла
    const decryptFileForm =
document.getElementById('decryptFileForm');
    decryptFileForm.addEventListener('submit', async (e) => {
      e.preventDefault();

      // Получаем значение ключа и убираем лишние пробелы
      const key =
document.getElementById('decryptKey2').value.trim();

      // Проверяем длину ключа
      if (key.length !== 8) {
        console.log('tut');
        return Swal.fire({
          icon: 'error',
          title: 'Ошибка при расшифровании текста!',
          text: 'Проверьте ключ на правильность.',
        });
      }

      // Создаем FormData для отправки файла
      const formData = new FormData(decryptFileForm);
      try {
        const response = await fetch('/decrypt-file', {
          method: 'POST',
          body: formData,
        });
        const data = await response.json();

        if (data.error) {
          alert(data.error);
        } else {
          document.getElementById('fileMessage2').textContent =
data.message;
          document.getElementById('fileDecryptTime').textContent =
data.decryptTimeMs;
        }
      } catch (err) {
        console.log('Ошибка при отправке запроса:', err);
      }
    });

      </script>
    </body>
</html>
```

# ПРИЛОЖЕНИЕ В

## Листинг файла speck.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Speck Cipher</title>
    <style>
        /* Стили для блока */
        body {
            font-family: Arial, sans-serif;
            padding: 20px;
            background-color: #f4f4f4;
        }
        h1 {
            text-align: center;
            margin-bottom: 30px;
        }
        .container {
            max-width: 600px;
            margin: 0 auto;
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        }
        label {
            font-size: 14px;
            margin-bottom: 8px;
            display: block;
        }
        input, textarea {
            width: 100%;
            padding: 10px;
            margin-bottom: 20px;
            border: 1px solid #ccc;
            border-radius: 4px;
            font-size: 14px;
        }
        button {
            width: 100%;
            padding: 10px;
            background-color: #4CAF50;
            color: #fff;
            border: none;
            border-radius: 4px;
            font-size: 16px;
            cursor: pointer;
        }
```

```
            button:hover {
                background-color: #45a049;
            }
            .result {
                margin-top: 20px;
                padding: 10px;
                background-color: #e9f9e9;
                border-radius: 4px;
                font-weight: bold;
            }
            .time-result {
                margin-top: 20px;
                padding: 10px;
                background-color: #fffbe6;
                border-radius: 4px;
                font-size: 14px;
            }
        </style>
    </head>
    <body>
        <h1>Speck Cipher Encryption and Decryption</h1>

        <!-- Блок для шифрования -->
        <div class="container">
            <h2>Шифрование</h2>
            <label      for="plaintext">Введите      текст      для
шифрования:</label>
            <textarea id="plaintext" rows="4" placeholder="Введите
текст..."></textarea>

            <label  for="key">Введите  ключ  (в  16-ричном  формате,
например, 123F):</label>
            <input   type="text"   id="key"   placeholder="Введите
ключ...">

            <label   for="fileToEncrypt">Или   выберите   файл   для
шифрования:</label>
            <input type="file" id="fileToEncrypt">

            <button   onclick="processSpeckCipher()">Шифровать   и
расшифровать</button>

            <div class="result" id="encryptedResult"></div>
            <div class="result" id="encryptedResultHex"></div>
            <div class="result" id="decryptedResult"></div>
            <div                                  class="time-result"
id="encryptionTimeResult"></div>
            <div                                  class="time-result"
id="decryptionTimeResult"></div>
        </div>

        <!-- Блок для расшифровки -->
        <div class="container">
            <h2>Расшифровка</h2>
```

```
            <label for="ciphertext">Введите зашифрованный
текст:</label>
            <textarea id="ciphertext" rows="4" placeholder="Введите
зашифрованный текст..."></textarea>

            <label for="decryptKey">Введите ключ (в 16-ричном
формате, например, 123F):</label>
            <input type="text" id="decryptKey" placeholder="Введите
ключ...">

            <label for="fileToDecrypt">Или выберите файл для
расшифровки:</label>
            <input type="file" id="fileToDecrypt">

            <button
onclick="decryptSpeckText()">Расшифровать</button>

            <div class="result" id="decryptedCiphertext"></div>
            <div                                    class="time-result"
id="decryptionTimeResult2"></div>
        </div>

        <script>
            // Функции для шифрования и расшифрования
            function rotr(x, n, bits) {
                let result = (x >>> n) | (x << (bits - n));
                return result & ((1 << bits) - 1);  // Убедитесь,
что результат в пределах битовой ширины
            }

            function rotl(x, n, bits) {
                let result = (x << n) | (x >>> (bits - n));
                return result & ((1 << bits) - 1);  // Убедитесь,
что результат в пределах битовой ширины
            }

            function encryptSpeck(plaintext, keys) {
                let [x, y] = plaintext;

                for (let round = 0; round < keys.length; round++) {
                    x = rotr(x, 7, 16);
                    x = (x + y) & 0xFFFF;
                    x ^= keys[round];

                    y = rotl(y, 2, 16);
                    y ^= x;
                }

                return [x, y];
            }

            function decryptSpeck(ciphertext, keys) {
                let [x, y] = ciphertext;
```

```
                for (let round = keys.length - 1; round >= 0; round-
-) {
                    y ^= x;
                    y = rotl(y, 16 - 2, 16);

                    x ^= keys[round];
                    x = (x - y) & 0xFFFF;
                    x = rotr(x, 16 - 7, 16);
                }

                return [x, y];
            }

            function stringToBlocks(str) {
                if (str.length % 2 !== 0) {
                    str += '\0';
                }

                let blocks = [];
                for (let i = 0; i < str.length; i += 2) {
                    let block = 0;
                    block |= str.charCodeAt(i) << 8;
                    block |= str.charCodeAt(i + 1);
                    blocks.push(block);
                }
                return blocks;
            }

            function blocksToString(blocks) {
                let str = '';
                for (let i = 0; i < blocks.length; i++) {
                    str += String.fromCharCode((blocks[i] >> 8) &
0xFF);
                    str += String.fromCharCode(blocks[i] & 0xFF);
                }
                return str.replace(/\0/g, '');
            }

            function        generateKeysFromInitialKey(initialKey,
numberOfRounds) {
                let keys = [];
                let key = initialKey;
                for (let i = 0; i < numberOfRounds; i++) {
                    keys.push(key);
                    key = (key + 0x1) & 0xFFFF;
                }
                return keys;
            }

            // Чтение файла
            function readFile(fileInputId) {
                return new Promise((resolve, reject) => {
                    let                fileInput                =
document.getElementById(fileInputId);
```

```
                    let file = fileInput.files[0];
                    if (!file) {
                        reject('Нет файла');
                    }
                    let reader = new FileReader();
                    reader.onload = function(event) {
                        resolve(event.target.result);
                    };
                    reader.onerror = function(error) {
                        reject(error);
                    };
                    reader.readAsText(file);
                });
            }

            // Функция для сохранения зашифрованного текста в файл
            function saveToFile(content, filename) {
                const blob = new Blob([content], { type:
'text/plain' });
                const url = URL.createObjectURL(blob);
                const a = document.createElement('a');
                a.href = url;
                a.download = filename;
                a.click();
                URL.revokeObjectURL(url);
            }

            // Шифрование и расшифровка с интерфейсом
            async function processSpeckCipher() {
                let plaintext =
document.getElementById('plaintext').value;
                let keyString =
document.getElementById('key').value;
                let initialKey = parseInt(keyString, 16);

                let fileEncrypted = await
readFile('fileToEncrypt').catch(() => null);
                if (fileEncrypted) {
                    plaintext = fileEncrypted;
                }

                let blocks = stringToBlocks(plaintext);
                let keys = generateKeysFromInitialKey(initialKey,
24);

                // Засекаем время шифрования
                let startEncryptionTime = Date.now();

                let ciphertext = [];
                for (let i = 0; i < blocks.length; i += 2) {
                    let block = [blocks[i], blocks[i + 1] || 0];
                    let encryptedBlock = encryptSpeck(block, keys);
                    ciphertext.push(...encryptedBlock);
                }
```

```
                let encryptedText = blocksToString(ciphertext);

document.getElementById('encryptedResult').textContent =
'Зашифрованный текст: ' + encryptedText;

                // Зашифрованный текст в шестнадцатеричной
кодировке
                let encryptedTextHex = ciphertext.map(block =>
block.toString(16).padStart(4, '0')).join(' ');

document.getElementById('encryptedResultHex').textContent =
'Зашифрованный текст (Hex): ' + encryptedTextHex;

                // Сохранение зашифрованного текста в файл
                saveToFile(encryptedText, 'encrypted_text.txt');

                // Завершаем время шифрования
                let endEncryptionTime = Date.now();
                let encryptionTime = (endEncryptionTime -
startEncryptionTime) / 1000;  // Время в секундах

document.getElementById('encryptionTimeResult').textContent =
`Время шифрования: ${encryptionTime.toFixed(3)} сек`;

                // Засекаем время расшифровки
                let startDecryptionTime = Date.now();

                let decrypted = [];
                for (let i = 0; i < ciphertext.length; i += 2) {
                    let block = [ciphertext[i], ciphertext[i + 1]
|| 0];
                    let decryptedBlock = decryptSpeck(block, keys);
                    decrypted.push(...decryptedBlock);
                }


document.getElementById('decryptedResult').textContent =
'Расшифрованный текст: ' + blocksToString(decrypted);

                // Сохраняем расшифрованный текст в файл
                saveToFile(blocksToString(decrypted),
'decrypted_text.txt');

                // Завершаем время расшифровки
                let endDecryptionTime = Date.now();
                let decryptionTime = (endDecryptionTime -
startDecryptionTime) / 1000;  // Время в секундах

document.getElementById('decryptionTimeResult').textContent =
`Время расшифровки: ${decryptionTime.toFixed(3)} сек`;
            }

            async function decryptSpeckText() {
```

```
                let                encryptedText                =
document.getElementById('ciphertext').value;
                let                  keyString                  =
document.getElementById('decryptKey').value;
                let initialKey = parseInt(keyString, 16);

                let         fileDecrypted         =         await
readFile('fileToDecrypt').catch(() => null);
                if (fileDecrypted) {
                    encryptedText = fileDecrypted;
                }

                let blocks = stringToBlocks(encryptedText);
                let  keys  =  generateKeysFromInitialKey(initialKey,
24);

                // Засекаем время расшифровки
                let startDecryptionTime = Date.now();

                let decrypted = [];
                for (let i = 0; i < blocks.length; i += 2) {
                    let block = [blocks[i], blocks[i + 1] || 0];
                    let decryptedBlock = decryptSpeck(block, keys);
                    decrypted.push(...decryptedBlock);
                }

                let decryptedText = blocksToString(decrypted);

document.getElementById('decryptedCiphertext').textContent        =
'Расшифрованный текст: ' + decryptedText;

                // Сохраняем расшифрованный текст в файл
                saveToFile(decryptedText, 'decrypted_text.txt');

                // Завершаем время расшифровки
                let endDecryptionTime = Date.now();
                let   decryptionTime   =   (endDecryptionTime   -
startDecryptionTime) / 1000;  // Время в секундах

document.getElementById('decryptionTimeResult2').textContent      =
`Время расшифровки: ${decryptionTime.toFixed(3)} сек`;
            }
        </script>
    </body>
</html>
```