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 симметричных алгоритмов шифрования XXTEA и CAST»

Исполнитель

Студент 4 курса группы 7 Голубицкая Ю. В.

(Ф.И.О.)

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

Курсовой проект защищен с оценкой

Минск 2024

Оглавление

[Введение 4](#_Toc182653551)

[1 Постановка задачи 5](#_Toc182653552)

[2 Описание методов 6](#_Toc182653553)

[3 Описание программного средства 12](#_Toc182653554)

[4 Тестирование программного средства 16](#_Toc182653555)

[5 Сравнительный анализ методов 19](#_Toc182653556)

[Заключение 24](#_Toc182653557)

[Список используемых источников 25](#_Toc182653558)

# Введение

С развитием цифровых технологий и увеличением объема передаваемой информации, вопросы безопасности данных становятся все более актуальными. Шифрование информации – один из ключевых методов защиты, позволяющий предотвратить несанкционированный доступ к конфиденциальным данным. Среди множества алгоритмов шифрования особое внимание уделяется симметричным алгоритмам, которые используют один и тот же ключ как для шифрования, так и для расшифрования данных.

В данной работе будет проведен сравнительный анализ производительности и безопасности двух симметричных алгоритмов шифрования – XXTEA и CAST. Выбор этих алгоритмов обусловлен их распространенностью и различиями в подходах к шифрованию. XXTEA, являющийся усовершенствованной версией алгоритма TEA, отличается высокой скоростью работы и простотой реализации. В то же время, алгоритм CAST, разработанный с акцентом на безопасность, обеспечивает надежную защиту данных благодаря использованию различных режимов шифрования и длины ключа.

Результатом данной курсовой работы является проект, содержащий в себе реализацию двух алгоритмов, и позволяющий сравнить их по некоторым признакам.

# Постановка задачи

В рамках данного курсового проекта необходимо провести сравнительный анализ производительности и безопасности симметричных алгоритмов шифрования XXTEA и CAST. Для достижения этой цели следует рассмотреть ряд задач, представленных ниже.

Требование к средствам разработки:

* Язык программирования JavaScript.
* Средство разработки MS Visual Studio Code.

Приложение должно:

* Шифровать и расшифровывать текстовые сообщения.
* Выводить время, затраченное на шифрование и расшифрование, что позволит оценить производительность каждого из алгоритмов.
* Реализовать удобный интерфейс для ввода данных и отображения результатов, включая возможность выбора алгоритма шифрования.

Кроме того, необходимо сравнить результаты тестирования по производительности и безопасности, а также сформулировать выводы о том, какой из алгоритмов лучше подходит для определенных условий и задач.

# Описание методов

Выбранные для сравнения алгоритмы шифрования относятся к классу блочно-симметричных. Блочно-симметричные шифры – это класс криптографических алгоритмов, которые шифруют данные блоками фиксированного размера, например, 128, 192 или 256 бит. Они используют симметричное шифрование, что означает, что для шифрования и дешифрования применяется один и тот же ключ.

Основной принцип работы блочно-симметричных шифров заключается в разбиении данных на блоки одинакового размера. Если размер данных не кратен размеру блока, используется механизм дополнения. На каждом раунде ключ преобразуется с помощью функции ключа, что обеспечивает дополнительную безопасность. Каждый блок проходит через несколько раундов, где применяются операции замены, перестановки и смешивания данных.

XXTEA – криптографический алгоритм, реализующий блочное симметричное шифрование и представляющий собой сеть Фейстеля. Является расширение алгоритма Block TEA. Разработан и опубликован Дэвидом Уилером и Роджером Нидхемом в 1998 году. Выполнен на простых и быстрых операциях: XOR, подстановка, сложение.

XXTEA, как и остальные шифры семейства TEA, обладает рядом отличительных особенностей по сравнению с аналогичными шифрами:

* высокая скорость работы;
* малое потребление памяти;
* простая программная реализация;
* относительно высокая надёжность.

Данный алгоритм работает по следующему принципу:

Исходный текст разбивается на слова по 32 бита каждый, из полученных слов формируется блок. Ключ также разбивают на 4 части, состоящие из слов по 32 бита каждый, и формируют массив ключей. В ходе одного раунда работы алгоритма шифруется одно слово из блока. После того, как были зашифрованы все слова, заканчивается цикл, и начинается новый. Количество циклов (раундов) зависит от количества слов и вычисляется по формуле:

R = 6 + 52/*n,*  (1.1)

где *n* – количество слов.

Шифрование одного слова состоит в следующем:

1. Над левым соседом выполняется операция битового сдвига влево на два, а над правым операция битового сдвига вправо на пять. Над полученными значениями выполняют операцию побитового сложения по модулю 2.
2. Над левым соседом выполняется операция битового сдвига вправо на три, а над правым операция битового сдвига влево на 4. Над полученными значениями выполняют операцию побитового сложения по модулю 2
3. Полученные числа складывают по модулю 232.
4. Константа *δ*, выведенная из Золотого сечения *δ* = (5 - 1) · 231 = 2654435769 = = 9E3779B9*h*, умножается на номер цикла (это было сделано для предотвращения простых атак, основанных на симметрии раундов).
5. Полученное в предыдущем пункте число складывают побитово по модулю 2 с правым соседом.
6. Полученное в 4 пункте число сдвигают побитово направо на 2, складывают побитово по модулю два с номером раунда и находят остаток от деления на 4. С помощью полученного числа выбирают ключ из массива ключей.
7. Выбранный в предыдущем раунде ключ складывают побитово по модулю 2 с левым соседом.
8. Числа, полученные в предыдущем и 4 пунктах, складывают по модулю 232.
9. Числа, полученные в предыдущем и 3 пунктах, складывают побитово по модулю 2, данную сумму складывают с шифруемым словом по модулю 232.

Схема алгоритма представлена на рисунке 1.1.

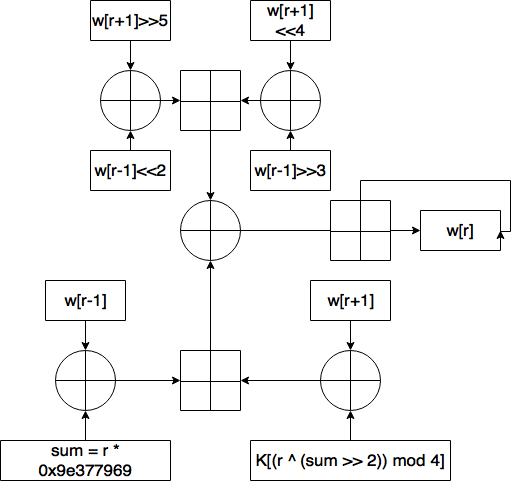


Рисунок 1.1 – Схема алгоритма XXTEA

Для расшифровки сообщений, зашифрованных с помощью XXTEA, необходимо взять зашифрованный текст и ключ, использованный при шифровании. Сначала зашифрованный текст обычно преобразуют из формата, например, Hex или Base64, в бинарный вид. Затем применяют алгоритм XXTEA, используя этот ключ, чтобы выполнить процесс расшифровки. В результате получается оригинальный текст. Важно, чтобы ключ был правильным, иначе расшифровка не будет успешной.

На данный момент для XXTEA существует атака на основе адаптивно подобранного открытого текста, опубликованная Элиас Яаррков в 2010 году. Существует два подхода, в которых используется уменьшение количества циклов за счет увеличения количества слов.

Первый подход дифференциального криптоанализа XXTEA заключается в шифровании нескольких слов из открытого текста, начиная с одного слова и добавляя к нему новое значение. Если после шифрования различия сохраняются только в одном слове, процесс продолжается. Элиас Яаррков обнаружил, что вероятность успешного прохождения пяти полных циклов колебалась между 2-109 и 2-110 для большинства ключей. Это означает, что, если пара текстов прошла 5 из 6 циклов, её можно считать верной; при этом возникновение различий в других словах указывает на необходимость повторного подбора. Схема первого подхода показана на рисунке 1.2.

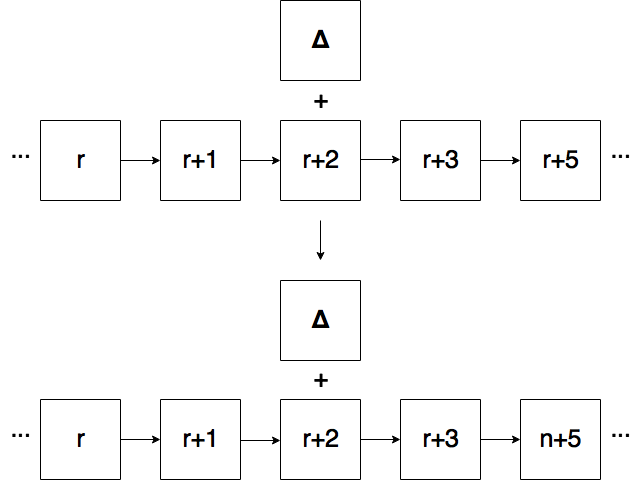


Рисунок 1.2 – Схема первого подхода криптоанализа XXTEA

Второй подход отличается тем, что после первого раунда разница должна перейти в текущее слово из следующего, изменив своё значение. Затем, после следующего раунда, разница возвращается к исходному слову, но со знаком, отличающимся от первоначального. Яаррков выяснил, что для успешного нахождения правильной пары достаточно 259 текстов с разницей в пределах от -32 до 32. Увеличение этого диапазона не улучшает результаты. Эта атака на XXTEA была успешной даже при уменьшении количества циклов до четырёх, где правильная пара была получена с помощью 235 пар текстов, что соответствует предыдущей оценке в 234,7 пар. Схему второго подхода можно увидеть на рисунке 1.3.

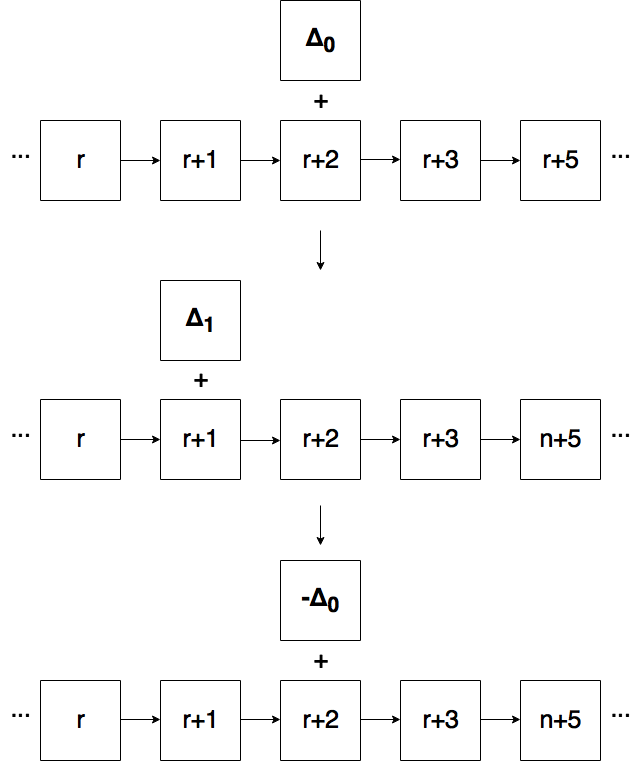


Рисунок 1.3 – Схема второго подхода криптоанализа XXTEA

CAST-128 (или CAST5) в криптографии – блочный алгоритм симметричного шифрования на основе сети Фейстеля, который используется в целом ряде продуктов криптографической защиты, в частности некоторых версиях PGP и GPG, и, кроме того, одобрен для использования Канадским правительством.

Алгоритм был создан в 1996 году Карлайлом Адамсом и Стаффордом Таваресом используя метод построения шифров CAST, который используется также и другим их алгоритмом CAST-256.

CAST-128 состоит из 12 или 16 раундов сети Фейстеля с размером блока 64 бита и длиной ключа от 40 до 128 бит (но только с инкрементацией по 8 бит). 16 раундов используются, когда размеры ключа превышают 80 бит. В алгоритме используются 8 на 16 S- блоки, основанные на бент-функции, операции XOR и модулярной арифметике. Есть три различных типа функций раундов, но они похожи по структуре и различаются только в выборе выполняемой операции в различных местах.

Хотя CAST-128 защищен патентом Entrust, его можно использовать во всём мире для коммерческих или некоммерческих целей бесплатно.

CAST-128 основан на сети Фейстеля. Полный алгоритм шифрования изложен в следующих четырех шагах:

ВХОД: текст *m1* ... *m64*, ключ *K* = *k1* ... *k128*.

ВЫХОД: зашифрованный текст *c1* ... *c64*.

1. Развертка ключа cоставляет 16 пар подключей {*Kmi, Kri*} полученных из *K*. CAST-128 использует пару подключей за раунд: 32-битные величины *Km* используется в качестве «маскировки» ключа и *Kr* используют как «перестановки» ключа, из которых используются только начальные 5-бит.
2. Разделяет текст на левую *L0* и правую *R0* 32-битные половины *L0* = *m1* ... *m32* и *R0* = *m33* ... *m64*).
3. 16 раундов вычисляется *Li* и *Ri* следующим образом: *Li* = *Ri-1*; *Ri* =   
   = *Li-1*F (*Ri-1*,*Kmi*,*Kri*), где *F* имеет тип 1, тип 2, тип 3 или, в зависимости от *i*. Три различных типов функции *F* используются в CAST-128. Типы выглядит следующим образом (где «*D*» является входными данными в функцию *F* и «*Ia*» – «*Id*» является наиболее значимый байт – наименее значимый байт *I*, соответственно). Нужно обратить внимание, что «+» и «-» сложение и вычитание по модулю 232, «^» является побитовое XOR и «<<<» является циклическим сдвигом влево. Таблица, которая показывает неидентичные раунды продемонстрирована под пунктом 1.

Таблица 1 – Неидентичные раунды

|  |  |
| --- | --- |
| Раунды 1,4,7,10,13,16 | *I = ((Kmi + Ri-1) <<< Kri)*  *F = ((S1[Ia] ^ S2[Ib]) – (S3[Ic]) )+ S4[Id]* |
| Раунды 2,5,8,11,14 | *I = ((Kmi ^ Ri-1) <<< Kri)*  *F = ((S1[Ia] - S2[Ib]) + (S3[Ic])) ^ S4[Id]* |
| Раунды 3,6,9,12,15 | *I = ((Kmi - Ri-1) <<< Kri)*  *F = ((S1[Ia] + S2[Ib]) ^ (S3[Ic]))- S4[Id]* |

CAST-128 использует восемь полей замены: поля *S1, S2, S3* и *S4* раундовые функции полей замены, *S5, S6, S7* и *S8* являются ключами развертки полей замены. Несмотря на то, что 8 полей замены требуют в общей сложности 8 Кбайт для хранения, только 4 Кбайта требуются во время фактического шифрования или дешифрование, так как генерация подключа обычно делается до любого ввода данных.

1. Меняем окончательные блоки местами *L16*, *R16* и объединяем, чтобы сформировать зашифрованный текст.

Расшифрование совпадает с алгоритмом шифрования, приведенным выше, кроме того, что раунды (и, следовательно, пары подключей), используются в обратном порядке, чтобы вычислить (*L0, R0*) из (*R16, L16*). Схема алгоритма представлена на рисунке 1.4.

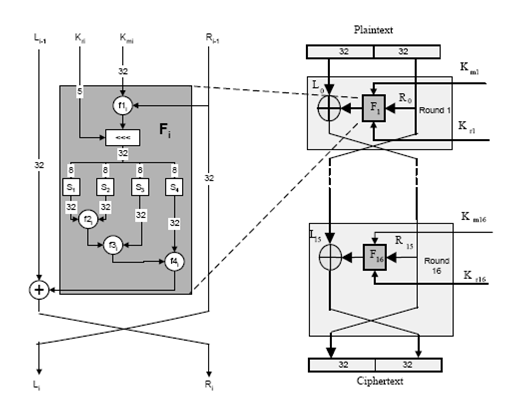


Рисунок 1.4 – Схема алгоритма CAST-128

Существует несколько подходов к криптоанализу CAST-128:

* Дифференциальный криптоанализ: Этот метод основан на анализе различий между парами зашифрованных текстов, чтобы выявить закономерности. В случае CAST-128 были проведены исследования, показывающие, что при определенных условиях можно уменьшить количество возможных ключей.
* Линейный криптоанализ: Этот метод использует линейные связи между открытым текстом, зашифрованным текстом и ключом. Исследования показывают, что с использованием большого количества пар текстов можно упростить задачу нахождения ключа.
* Атаки на основе статистического анализа: Криптоаналитики могут использовать частотный анализ и другие статистические методы, чтобы выявить слабые места в шифре.

Несмотря на некоторые уязвимости, CAST-128 считается достаточно безопасным для большинства приложений, особенно при использовании длинных ключей.

# Описание программного средства

В данной курсовой работе было реализовано веб-приложение, реализующее симметричное шифрование и дешифрование с использованием алгоритмов XXTEA и CAST.

Используются следующие языки и технологии:

* JavaScript/TypeScript:

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.

TypeScript является надмножеством JavaScript, добавляющим статическую типизацию. Это позволяет разработчикам задавать типы переменных, что помогает избежать ошибок в коде. TypeScript компилируется в JavaScript, что делает его совместимым с любыми браузерами. Он часто используется в крупных проектах, где важна поддержка и масштабируемость кода.

* 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.

* EJS (Embedded JavaScript):

EJS (Embedded JavaScript) – это шаблонизатор для JavaScript, который позволяет встраивать JavaScript-код непосредственно в HTML-разметку. Он используется в веб-приложениях для динамической генерации контента на стороне сервера. EJS позволяет разработчикам создавать шаблоны, в которых можно использовать переменные, циклы и условные операторы, что делает разметку более интерактивной и адаптивной к данным, получаемым от сервера.

Помимо этого, были использованы и различные библиотеки:

* xxtea-node:

xxtea-node – это библиотека для Node.js, реализующая алгоритм шифрования XXTEA (Corrected Block TEA). Этот алгоритм предназначен для безопасного шифрования данных и является улучшенной версией оригинального TEA.

* 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:

Crypto – это встроенный модуль в Node.js, который предоставляет набор криптографических функций для обеспечения безопасности приложений. Он позволяет выполнять различные операции, такие как хеширование данных, шифрование и дешифрование, создание цифровых подписей и генерацию случайных чисел.

* 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.

* chart.js:

Популярная библиотека для визуализации данных на веб-страницах, позволяющая создавать интерактивные графики с использованием HTML5 Canvas. Она поддерживает различные типы графиков, включая линии, столбцы, круговые диаграммы и многие другие, что делает ее универсальным инструментом для разработчиков.

Структура разработанного приложения представлена на рисунке 3.1.

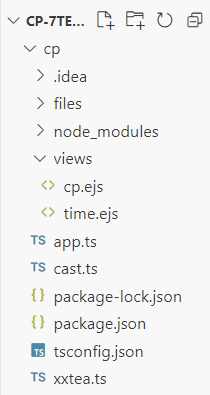


Рисунок 3.1 – Структура приложения

Ниже представлено описание структуры:

* app.ts: Основной файл, содержащий настройки сервера и маршруты.
* cast.ts: файл для шифрования и дешифрования с использованием алгоритма CAST.
* xxtea.ts: файл для шифрования и дешифрования с использованием алгоритма XXTEA.
* views: папка с HTML-шаблонами.
* cp.ejs: HTML-шаблон, который предоставляет интерфейс для взаимодействия с пользователями.
* time.ejs: HTML-шаблон, который предоставляет интерфейс для демонстрации зависимости времени шифрования и расшифрования от объемов текстов.

Интерфейс включает в себя форму, где пользователи могут ввести оригинальное сообщение и ключ для шифрования. Результаты отображаются в виде таблицы с двумя колонками для каждого алгоритма, что позволяет легко сравнивать их производительность и результаты. Представлено на рисунке 3.2.

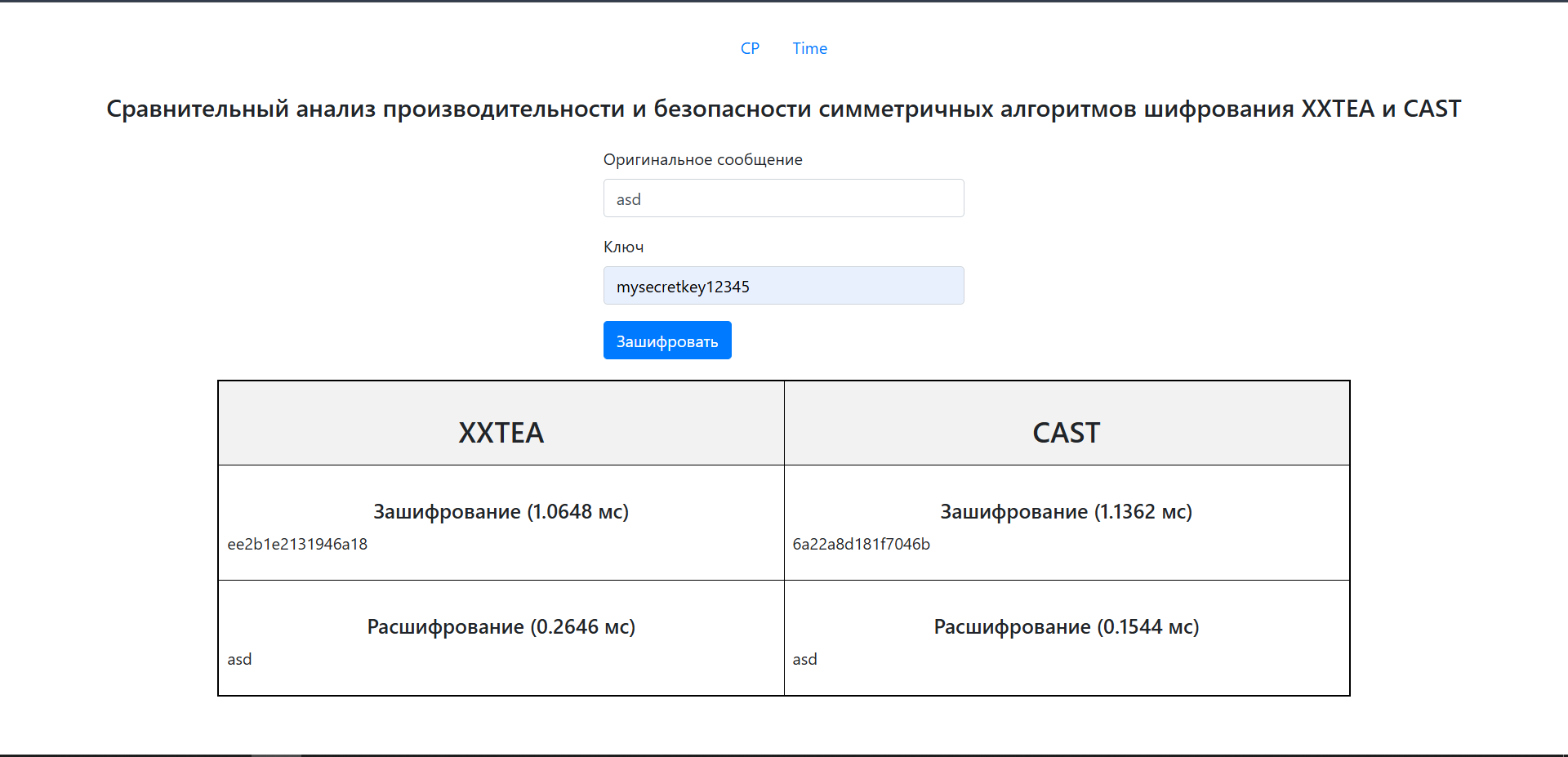


Рисунок 3.2 – Интерфейс приложения

Также, для более наглядного анализа приложение шифрует тексты различных объемов. Это позволяет проанализировать время шифрования и расшифрования алгоритмами, можно увидеть на рисунке 3.3.

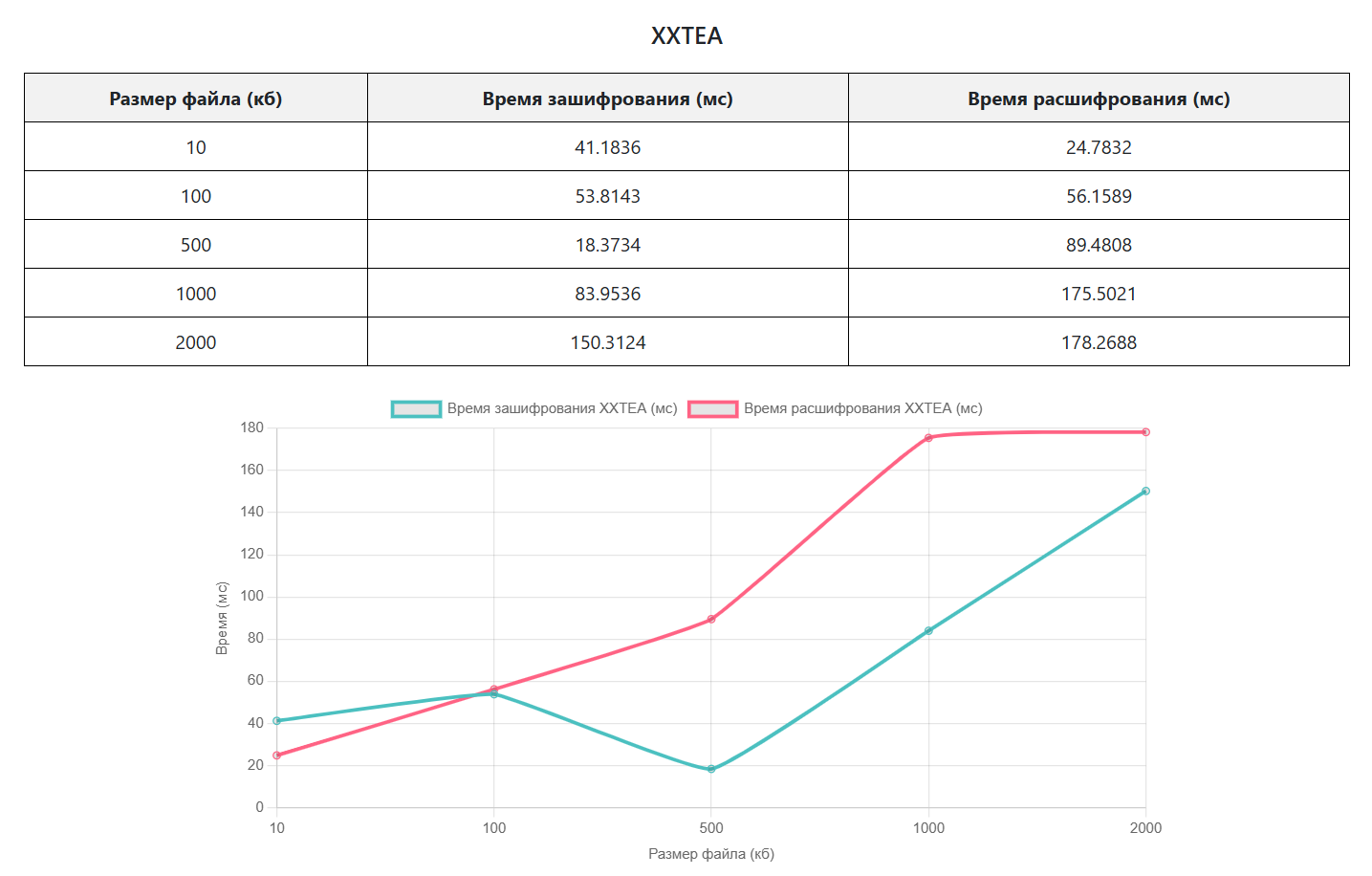


Рисунок 3.3 – Шифрование текстов разных объемов

Таким образом, разработанное веб-приложение позволяет выполнять следующие действия:

* Шифровать и расшифровывать текст с использованием двух различных алгоритмов:
* Проводить сравнительный анализ производительности между двумя алгоритмами. Для каждого алгоритма приложение измеряет время, затраченное на шифрование и расшифрование.
* Отображать результаты в удобном формате на веб-странице, где пользователи могут видеть: зашифрованный текст, расшифрованный текст и время, затраченное на каждую операцию.

# Тестирование программного средства

Чтобы протестировать разработанное программное средство, было выбрано несколько кодировок для исходного сообщения.

Первая из тестируемых кодировок – UTF-8. Результат работы приложения продемонстрирован на рисунке 4.1.

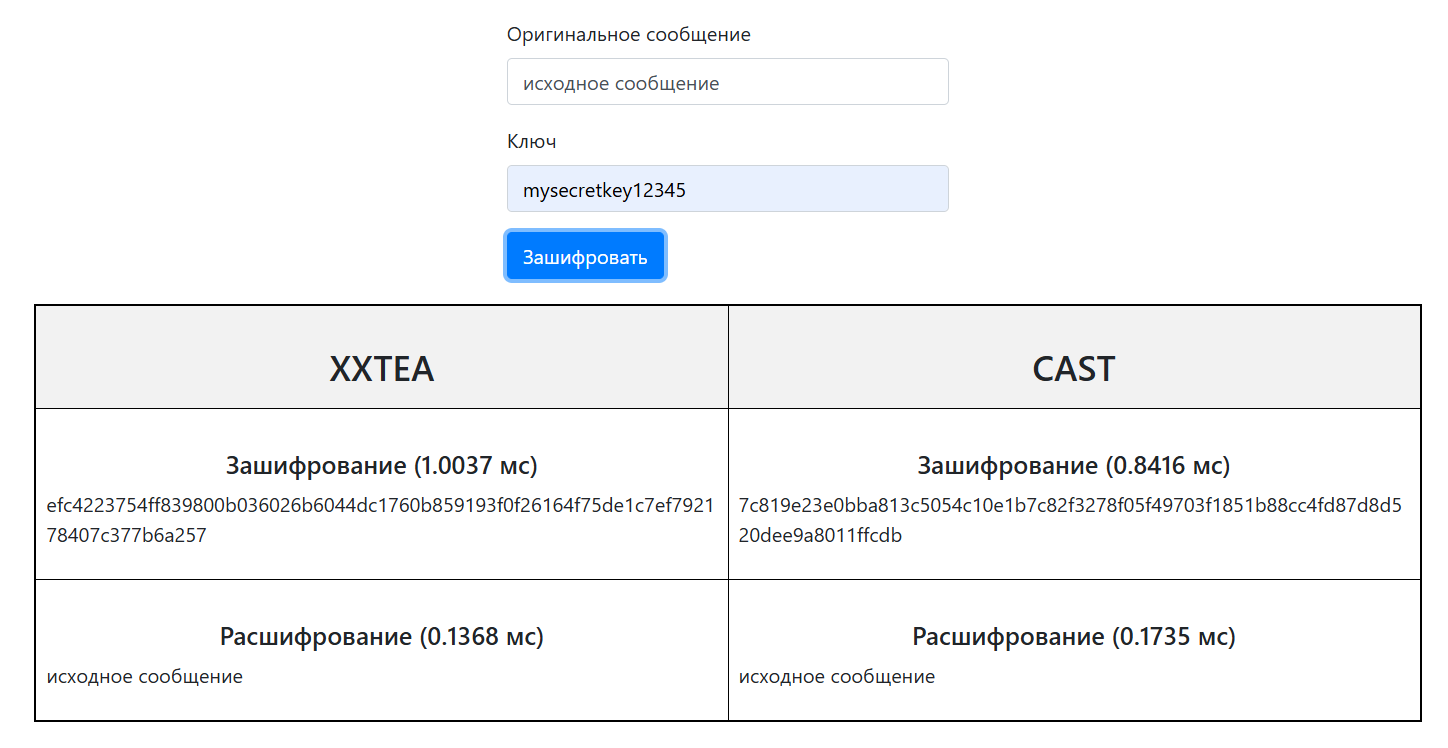


Рисунок 4.1 – Результат работы с кодировкой UTF-8

Разработанный программный продукт показал положительные результаты при работе с кодировкой UTF-8. Все функции шифрования и дешифрования работают корректно, обеспечивая целостность и безопасность исходных данных.

Следующая из выбранных кодировок – ASCII. ASCII – это стандартный код для представления текстовых символов, использующий 7 бит для кодирования 128 уникальных символов, включая английские буквы, цифры и специальные знаки. Результат работы приложения можно увидеть на рисунке 4.2.

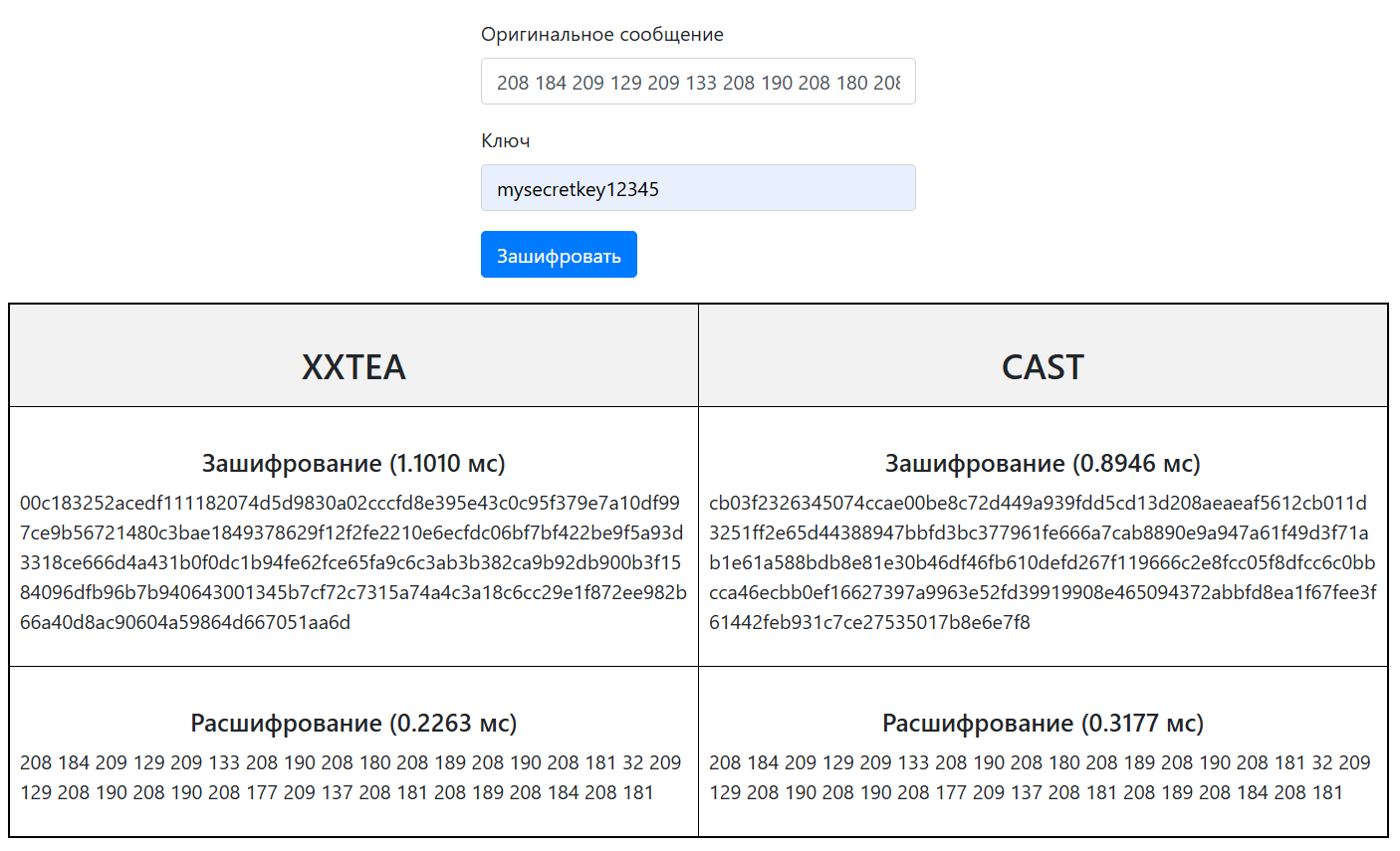


Рисунок 4.2 – Результат работы с кодировкой ASCII

Тестирование работы приложения с кодировкой ASCII дало положительные результаты. Все функции шифрования и дешифрования успешно обработали исходные сообщения, что подтверждает корректность реализации. Это свидетельствует о надежности алгоритмов при работе с базовым набором символов.

И последняя из тестируемых кодировок CP-1251. CP-1251 – это кодировка символов, используемая для представления кириллицы в операционных системах Windows. Она включает в себя 256 символов, охватывающих буквы русского и других славянских языков, а также специальные знаки. Результат работы приложения можно увидеть на рисунке 4.3.

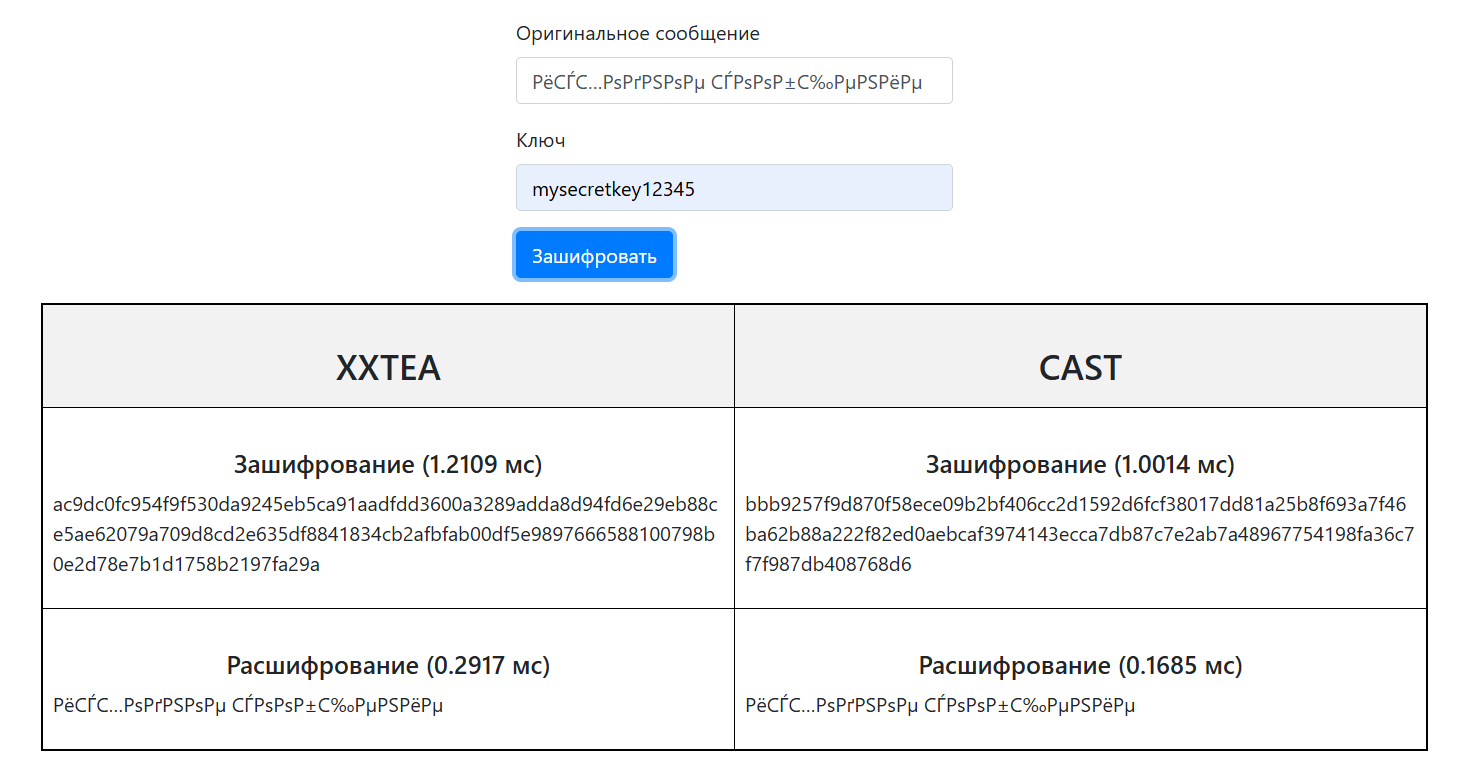


Рисунок 4.3 – Результат работы с кодировкой CP-1251

Все протестированные сообщения, закодированные в CP-1251, успешно зашифровались и расшифровались без потери данных.

Можно сделать вывод, что в ходе тестирования приложения были проверены различные кодировки, включая UTF-8, CP-1251 и ASCII. Все тестовые сообщения успешно проходили этапы шифрования и расшифровки, что подтверждает универсальность разработанного решения. Каждая кодировка обрабатывалась корректно, без каких-либо ошибок или искажений. Результаты тестирования свидетельствуют о высоком уровне надежности приложения. Это позволяет использовать его в различных сценариях работы с текстовыми данными.

# Сравнительный анализ методов

В ходе курсового проекта было проведено сравнение двух алгоритмов по нескольким критериям: скорость, безопасность, ресурсоемкость, возможность распараллеливания и т.д.

Как уже упоминалось, XXTEA работает на основе принципа итеративного блочного шифрования с использованием 32-битных целых чисел. Для зашифрования блока данных алгоритм использует несколько итераций с раундовыми ключами, которые генерируются из основного ключа. Основной принцип работы XXTEA заключается в делении блока данных на несколько слов и последовательном применении операций шифрования, включающих операции сложения, побитового XOR и сдвигов. Количество раундов зависит от размера блока данных и ключа.

CAST представляет собой блочный шифр, который использует 128-битные блоки данных и может работать с ключами длиной от 40 до 128 бит. Алгоритм основан на принципе итеративного блочного шифрования и использует 12 раундов для обработки данных. Сначала входной блок разбивается на две части, и затем к ним последовательно применяются раундовые функции, которые включают сложение, побитовые операции XOR и сдвиги. Генерация раундовых ключей происходит из основного ключа, что добавляет уровень сложности шифрования.

Длина ключа при использовании алгоритма шифрования XXTEA составляет 128 битов. Длина ключа при использовании алгоритма шифрования CAST составляет от 40 до 128 бит, что позволяет выбирать уровень безопасности в зависимости от требований приложения. Количество возможных ключей для CAST варьируется в зависимости от длины ключа. Количество возможных ключей: XXTEA – 2128 = 3.4 · 1038, CAST при длине ключа 128 имеет такое же количество возможных ключей.

Используя разработанное приложение, сравним алгоритмы по скоростям шифрования и расшифрования учитывая разный размер текста и ключ размером 128 бит.

Результаты времени шифрования и расшифрования алгоритмом XXTEA показаны в таблице 2.

Таблица 2 – Скорость шифрования и расшифрования алгоритмом XXTEA

|  |  |  |
| --- | --- | --- |
| Размер текста (Кб) | Время зашифрования (мс) | Время расшифрования (мс) |
| 10 | 0,6695 | 0,2633 |
| 100 | 13,8607 | 4,7095 |
| 500 | 10,0235 | 19,8756 |
| 1000 | 19,5595 | 57,4429 |
| 2000 | 36,9955 | 66,7451 |

Для более наглядной демонстрации, разработанное приложение выводит графики зависимости времени шифрования и расшифрования от объема текста. Такой график для XXTEA представлен на рисунке 5.1.



Рисунок 5.1 – График зависимости времени от объема текста для XXTEA

Время шифрования увеличивается с ростом размера текста. Для текста размером 10 Кб время шифрования составляет 0,6695 мс, а для текста размером 2000 Кб – 36,9955 мс.

Время расшифрования также растет с увеличением размера текста. Так, для 10 Кб текста время расшифрования равно 0,2633 мс, а для 2000 Кб – 66,7451 мс.

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 XXTEA может быть эффективным для шифрования небольших объемов данных, но его производительность снижается при работе с большими объемами информации.

Также были проведены измерения времени для алгоритма CAST, результаты можно увидеть в таблице 3.

Таблица 3 – Скорость шифрования и расшифрования алгоритмом CAST

|  |  |  |
| --- | --- | --- |
| Размер текста (Кб) | Время зашифрования (мс) | Время расшифрования (мс) |
| 10 | 0,8466 | 0,2345 |
| 100 | 4,243 | 1,6594 |
| 500 | 8,9299 | 8,642 |
| 1000 | 42,2927 | 26,4564 |
| 2000 | 35,5703 | 43,6435 |

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

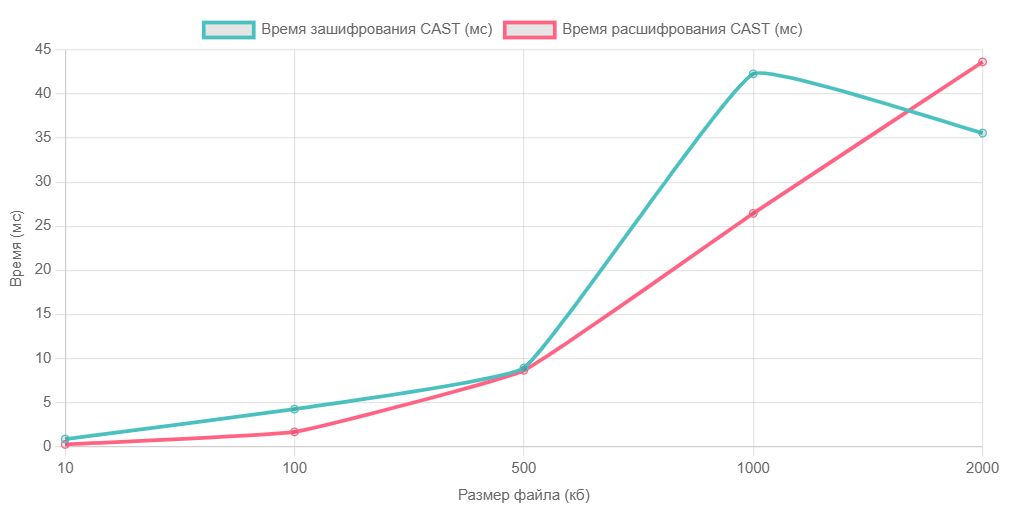


Рисунок 5.2 – График зависимости времени от объема текста для CAST

Можно сделать вывод, что скорость шифрования и расшифрования алгоритмом CAST-128 зависит от размера обрабатываемого текста. Чем больше размер текста, тем больше времени требуется для его шифрования и расшифрования.

Сравнивая результаты для XXTEA и CAST, можно заметить, что алгоритм CAST требует более длительного времени как на шифрование, так и на расшифрование по сравнению с XXTEA для одинаковых объемов данных. Это связано с тем, что CAST является более сложным и безопасным алгоритмом, использующим дополнительные раунды преобразований, которые увеличивают вычислительную нагрузку.

Теперь сравним размеры шифртекстов. Результаты представлены в таблице 4.

Таблица 4 – Анализ длины шифртекстов

|  |  |  |
| --- | --- | --- |
| Размер текста (Кб) | Шифртекст XXTEA (Кб) | Шифртекст CAST (Кб) |
| 10 | 20 | 20 |
| 100 | 200 | 200 |
| 500 | 1000 | 1000 |
| 1000 | 2000 | 2000 |
| 2000 | 4000 | 4000 |

Как можно заметить, размер шифртекста XXTEA и CAST увеличился в два раза. Сам шифр текст имеет вид в шестнадцатеричной системе. Для текста размером 10 Кб длина шифртекста составляет 20 Кб как для алгоритма XXTEA, так и для CAST. Таким образом, видно, что длина шифртекста пропорциональна размеру исходного текста, и оба алгоритма показывают идентичные результаты в этом отношении.

Алгоритм XXTEA может быть предпочтительнее в ситуациях, где требуется быстрая обработка небольших объемов данных, например, в мобильных приложениях. В то же время, CAST будет более подходящим для приложений, где безопасность имеет более высокий приоритет, например, при шифровании больших объемов конфиденциальных данных.

Воспользуемся программным средством и оценим лавинный эффект. Результаты продемонстрированы в таблице 5.

Таблица 5 – Анализ лавинного эффекта

|  |  |  |
| --- | --- | --- |
| Текст | Шифр XXTEA | Шифр CAST |
| abcd | db50f87e59e7747c | c3cf3b17a3eb8efe |
| abcx | c5490562d1213419 | c9d64e588f7f4856 |
| abxd | 17302fa4329cde62 | 63b7b21b78f34c99 |
| axcd | da9176509e5ba1c1 | 739e53613a81ca55 |
| xbcd | 8cb6b7d9ab74530a | 83c4fb985a99ed74 |

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.

Важным аспектом в современных вычислительных системах является распараллеливание операций при шифровании данных. Блочные шифры разбивают текст на блоки фиксированного размера и обрабатывают каждый блок отдельно. Это позволяет применять методы распараллеливания, особенно в режимах шифрования, которые поддерживают независимую обработку блоков. Оценим возможность распараллеливания, пользуясь информацией о методе построения блочного шифра.

Алгоритм XXTEA использует сеть Фейстеля. Это значит, что в каждом раунде необходимо использовать результат предыдущего раунда, что ограничивает степень распараллеливания. Может быть распараллелен только на уровне раундов шифрования.

Алгоритм CAST, в свою очередь, представляет собой блочный шифр с использованием структуры, аналогичной сети Фейстеля, но с некоторыми отличиями. Он использует 16 раундов шифрования и может быть оптимизирован для распараллеливания на уровне раундов. Однако, как и в случае с XXTEA, степень распараллеливания ограничена зависимостями данных между раундами. Тем не менее, CAST может быть адаптирован для работы в многопоточном окружении, что предоставляет определенные возможности для улучшения производительности при шифровании больших объемов данных. Проанализируем уязвимости алгоритмов, показано в таблице 6.

Таблица 6 – Сравнение криптостойкости шифров

|  |  |  |
| --- | --- | --- |
| Алгоритм | Атака | Вычислительная мощность |
| XXTEA | На основе адаптивно подобранного открытого текста | 235 пар текстов |
| С восстановлением ключа | 264 операций |
| CAST-128 | Дифференциальный криптоанализ | 261 операций |
| Линейный криптоанализ | 250 известных текстов |

На основе представленных в таблице 6 данных, XXTEA может быть предпочтителен в условиях ограниченных вычислительных ресурсов, особенно когда риск восстановления ключа не является критичным. Он требует меньше пар текстов для атаки на основе адаптивного подобранного открытого текста. В то же время, CAST-128 предлагает более высокую стойкость к дифференциальному и линейному криптоанализу, что делает его более подходящим для сценариев, требующих большей безопасности и где доступны достаточные вычислительные мощности. Выбор между этими алгоритмами зависит от конкретных требований к безопасности и ресурсам.

# Заключение

В рамках данного курсового проекта был проведен сравнительный анализ производительности и безопасности симметричных алгоритмов шифрования XXTEA и CAST. XXTEA — это симметричный блочный шифр, известный своей высокой эффективностью и простотой реализации, который работает с блоками данных фиксированного размера. CAST — это также симметричный блочный шифр, который был разработан для обеспечения безопасности данных и поддерживает различные размеры ключей, обеспечивая гибкость в использовании.

Для этой задачи было разработано приложение, использующее различные технологии, такие как: JavaScript/TypeScript, Node.js, Express.js и EJS. Кроме того, применяются и библиотеки: xxtea-node, node-forge, crypto, nvm и chart.js.

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, включая возможность выбора алгоритма шифрования.

Результаты тестирования показали, что разработанное приложение корректно работает с различными кодировками.

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

* Зависимость времени шифрования и расшифрования от объемов шифруемого текста.
* Лавинный эффект.
* Криптостойкость шифров.

Анализ выявил, что XXTEA более эффективен для обработки небольших объемов данных, обеспечивая быструю скорость шифрования и расшифрования, в то время как CAST демонстрирует более высокую безопасность, 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 CAST может быть адаптирован для многопоточной работы, что улучшает его производительность при шифровании больших объемов информации.

В конечном итоге, выбор между XXTEA и CAST 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 TEA, XTEA, XXTEA [Электронный ресурс]. Режим доступа: https://temofeev.ru/info/articles/shifrovanie-tea-xteaxxtea/. Дата доступа: 10.10.2024.
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 CAST-128 [электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/CAST-128. Дата доступа: 12.11.2024.
5. Шифрование XXTEA. [электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/XXTEA. Дата доступа: 14.11.2024.