

**Криптографическая защита данных в парадигме Zero-Dependency:
генерация браузерного шифратора файлов на базе Web Crypto API
методами промпт-инжиниринга**

Пайдершев Алексей Константинович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В работе рассматривается задача создания браузерного инструмента для шифрования файлов без использования сторонних библиотек. С помощью платформы Qwen Studio и метода детализированных текстовых запросов разработан веб-шифратор, работающий на основе нативного интерфейса Web Crypto API. Приложение реализует современные криптографические стандарты – AES-256-GCM для шифрования и PBKDF2 для получения ключа из пароля. Особенностью исследования стало использование бинарного критерия качества: сгенерированный код считается успешным только в том случае, если зашифрованный файл корректно расшифровывается. Эксперимент показал, что языковые модели хорошо справляются с задачами, опирающимися на строгие математические стандарты, и способны с первой попытки создавать рабочий криптографический код без необходимости ручных доработок.

Ключевые слова: Qwen Studio, нейросети, промпт-инжиниринг, вайб-кодинг, Zero-Dependency.

**Cryptographic data protection in the Zero-Dependency paradigm: generating
a browser-based file cipher based on the Web Crypto API using prompt
engineering methods**

Paysershev Alexey Konstantinovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The paper examines the task of creating a browser-based tool for encrypting files without using third-party libraries. Using the Qwen Studio platform and the method of detailed text queries, a web-based encryptor was developed, operating based on the native Web Crypto API interface. The application implements modern cryptographic standards – AES-256-GCM for encryption and PBKDF2 for deriving a key from a password. A distinctive feature of the study is the use of a binary quality criterion: the generated code is considered successful only if the encrypted file is correctly decrypted. The experiment showed that language models are well-suited to tasks based on strict mathematical standards and are capable of generating

working cryptographic code on the first attempt, without the need for manual adjustments.

Keywords: Qwen Studio, neural networks, prompt engineering, vibe coding, Zero-Dependency.

1 Введение

1.1 Актуальность

Сегодня пользователи всё чаще сталкиваются с необходимостью защищать личные документы, финансовые отчёты и рабочие материалы при передаче через интернет. Существующие решения – десктопные шифраторы, архиваторы с паролем, корпоративные системы защиты – требуют установки программ, настройки серверов или покупки лицензий. В этих условиях браузерные приложения, работающие локально и не отправляющие данные на внешние серверы, становятся привлекательной альтернативой.

Однако разработка криптографических программ считается одной из самых сложных задач в программировании. Ошибка в алгоритме шифрования или генерации ключа может сделать систему уязвимой, даже если математическая основа верна. Здесь возникает вопрос: может ли большая языковая модель самостоятельно написать криптографически стойкий код, опираясь только на встроенные средства браузера?

Кроме того, криптография даёт чёткий критерий качества: файл либо расшифровывается, либо нет. Это позволяет объективно оценить результат без субъективных суждений.

1.2 Обзор исследования

Д. С. Пашенко подтверждает, что внедрение ИИ-инструментов становится ключевым фактором конкурентоспособности ИТ-компаний, позволяя существенно ускорять процессы разработки [1]. Фундаментальную возможность создания сложных интерактивных приложений исключительно на базе нативных веб-технологий без внешних зависимостей обосновывают С. В. Рогатин и В. А. Антошкин [2]. Теоретическую базу для защиты данных в таких автономных решениях формирует Е. О. Кочедыкова, детально классифицирующая современные криптографические методы [3]. Однако автоматическая генерация кода требует строгого контроля, что И. В. Елманов доказывает, предлагая методологию Spec Driven Development для надежной валидации программ в парадигме вайб-кодинга [4]. Практическую способность больших языковых моделей эффективно работать со сложной структурой веб-приложений демонстрирует А. М. Титеев, оптимизируя с их помощью создание автотестов и анализ HTML-разметки [5].

1.3 Цель исследования

Цель работы – проверить, способна ли большая языковая модель (на примере Qwen Studio) самостоятельно спроектировать криптографическое веб-приложение, работающее с бинарными данными и реализующее современные стандарты шифрования без привлечения внешних библиотек.

2 Материалы и методы

Исследование построено на сочетании детализированных текстовых запросов к нейросети и строгого архитектурного ограничения: всё приложение размещается в одном HTML-файле без использования сторонних библиотек. Криптографические операции выполняются через встроенный в браузер Web Crypto API, который обеспечивает аппаратно ускоренные вычисления.

При составлении запроса особое внимание уделялось криптографической корректности. В промпте зафиксированы конкретные параметры: алгоритм AES-256-GCM, функция деривации PBKDF2 с SHA-256, размер соли – 16 байт, размер вектора инициализации – 12 байт, количество итераций – не менее 100 000. Также задан формат зашифрованного файла: соль и вектор инициализации добавляются в начало, чтобы не хранить их отдельно.

Приложение состоит из трёх модулей: модуль получения ключа из пароля, модуль шифрования и дешифрования, модуль интерфейса с drag-and-drop загрузкой файлов и индикатором надёжности пароля.

Главная особенность методологии – бинарная верификация. Успешность генерации проверяется не визуальным осмотром кода, а практическим тестом: зашифрованный файл должен корректно расшифровываться тем же приложением с тем же паролем.

3 Результаты

Первый этап исследования - формирование комплексного промпта, описывающего архитектуру шифратора.

Промт для генерации: «Действуй как Senior Frontend разработчик и специалист по криптографии. Твоя задача - написать полный, готовый к запуску код веб-приложения «Шифратор файлов». Приложение должно быть реализовано в одном файле и работать локально в браузере. Никаких сторонних библиотек - только нативные HTML, CSS и JavaScript с Web Crypto API.

Используй только нативный Web Crypto API для всех криптографических операций. Реализация должна быть криптографически стойкой и следовать современным лучшим практикам.

Архитектурные требования:

- Используй AES-GCM (256-бит) для шифрования файлов.
- Используй PBKDF2 с SHA-256 для получения ключа из пароля пользователя (минимум 100 000 итераций).
- Генерируй криптографически случайные salt (16 байт) и IV (12 байт) для каждой операции шифрования.
- Формат зашифрованного файла: [salt (16 байт)] + [iv (12 байт)] + [ciphertext].
- Обработывай файлы любого размера (в пределах доступной RAM).

Требования к интерфейсу:

- Современный, минималистичный дизайн. Тёмная тема с мягкими тенями и скруглёнными углами.

– Расположение: левая боковая панель с настройками, основная область с зоной загрузки и кнопками действий.

Элементы управления боковой панели:

- Поле ввода пароля (с переключателем показа/скрытия)
- Индикатор надёжности пароля (слабый/средний/сильный по длине и разнообразию символов)
- Слайдер итераций PBKDF2 (100 000 - 500 000, по умолчанию 200 000)
- Информационный блок об алгоритме (AES-256-GCM + PBKDF2)

Основная область:

- Большая зона drag-and-drop для загрузки файла
- Отображение информации о выбранном файле (имя, размер, тип)
- Две основные кнопки: «Зашифровать» и «Дешифровать»
- Прогресс-бар во время шифрования/дешифрования
- Область результата с кнопкой скачивания

Автоопределение режима: если файл имеет расширение .enc, предлагать дешифрование; иначе - шифрование

Показывать понятные сообщения об ошибках при неверном пароле или повреждённых файлах

Требования к логике шифрования:

- Читать файл как ArrayBuffer через FileReader
- Получать 256-битный ключ из пароля + salt через PBKDF2
- Шифровать данные через AES-GCM с полученным ключом и случайным IV
- Добавлять salt и IV в начало ciphertext
- Сохранять как Blob с расширением .enc, добавленным к исходному имени файла

Требования к логике дешифрования:

- Читать .enc файл как ArrayBuffer
- Извлекать salt (первые 16 байт) и IV (следующие 12 байт)
- Получать ключ из пароля + извлечённого salt через PBKDF2
- Дешифровать оставшиеся данные через AES-GCM
- Если дешифрование не удалось (неверный пароль или повреждённый файл), показывать понятное сообщение об ошибке
- Сохранять дешифрованный файл с исходным именем (убрать расширение .enc)

Код должен быть чистым, хорошо структурированным и прокомментирован на русском языке

Добавь визуальную обратную связь во время длительных операций

Выведи полный код в одном блоке, чтобы я мог сохранить его как index.html и сразу запустить».

Результат генерации: https://disk.yandex.ru/d/_2zutYikn4ezRg

В результате первичной генерации получен полностью функциональный прототип шифратора файлов (рис. 1). Нейросеть строго соблюла архитектурные ограничения, реализовав приложение в едином файле. Интерфейс успешно построен средствами нативного HTML и CSS с использованием CSS-переменных для темизации.

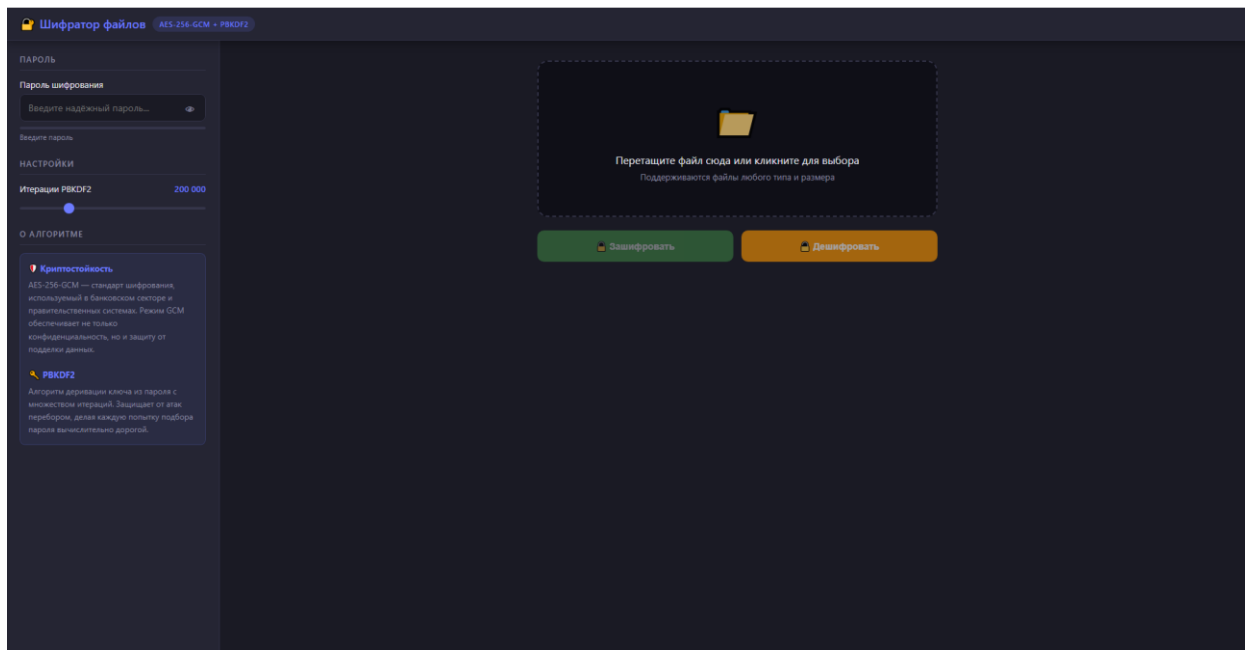


Рисунок 1. Интерфейс приложения

Важно отметить, что весь код заработал с первой попытки. Шифратор успешно зашифровал и дешифровал тестовый файл без ручных правок [6]. Это говорит о том, что стандартизированные криптографические протоколы с чёткой спецификацией оказываются более понятными для языковых моделей, чем задачи проектирования сложных пользовательских сценариев.

Сгенерированный код реализует все компоненты современной криптосистемы:

Получение ключа (PBKDF2) – функция `deriveKey()` принимает пароль пользователя и превращает его в 256-битный ключ AES-GCM. Используется случайная соль и настраиваемое число итераций (по умолчанию 200 000). Это делает перебор пароля медленным: каждая попытка требует заметных вычислительных ресурсов.

Шифрование (AES-256-GCM) – аутентифицированный режим, который обеспечивает и конфиденциальность, и целостность данных. Если кто-то попытается изменить зашифрованный файл, дешифрование завершится ошибкой, потому что GCM автоматически проверяет контрольную метку.

Генерация случайных чисел – соль (16 байт) и вектор инициализации (12 байт) создаются через `crypto.getRandomValues()`. Эта функция использует криптографически стойкий генератор случайных чисел операционной системы.

Формат файла – зашифрованный файл содержит все нужные метаданные в начале, поэтому для дешифрования не нужно отдельно передавать соль и вектор инициализации. Также при анализе файла через hex-редактор видно, что один и тот же файл при шифровании разными паролями имеет разные метаданные (рис. 2).

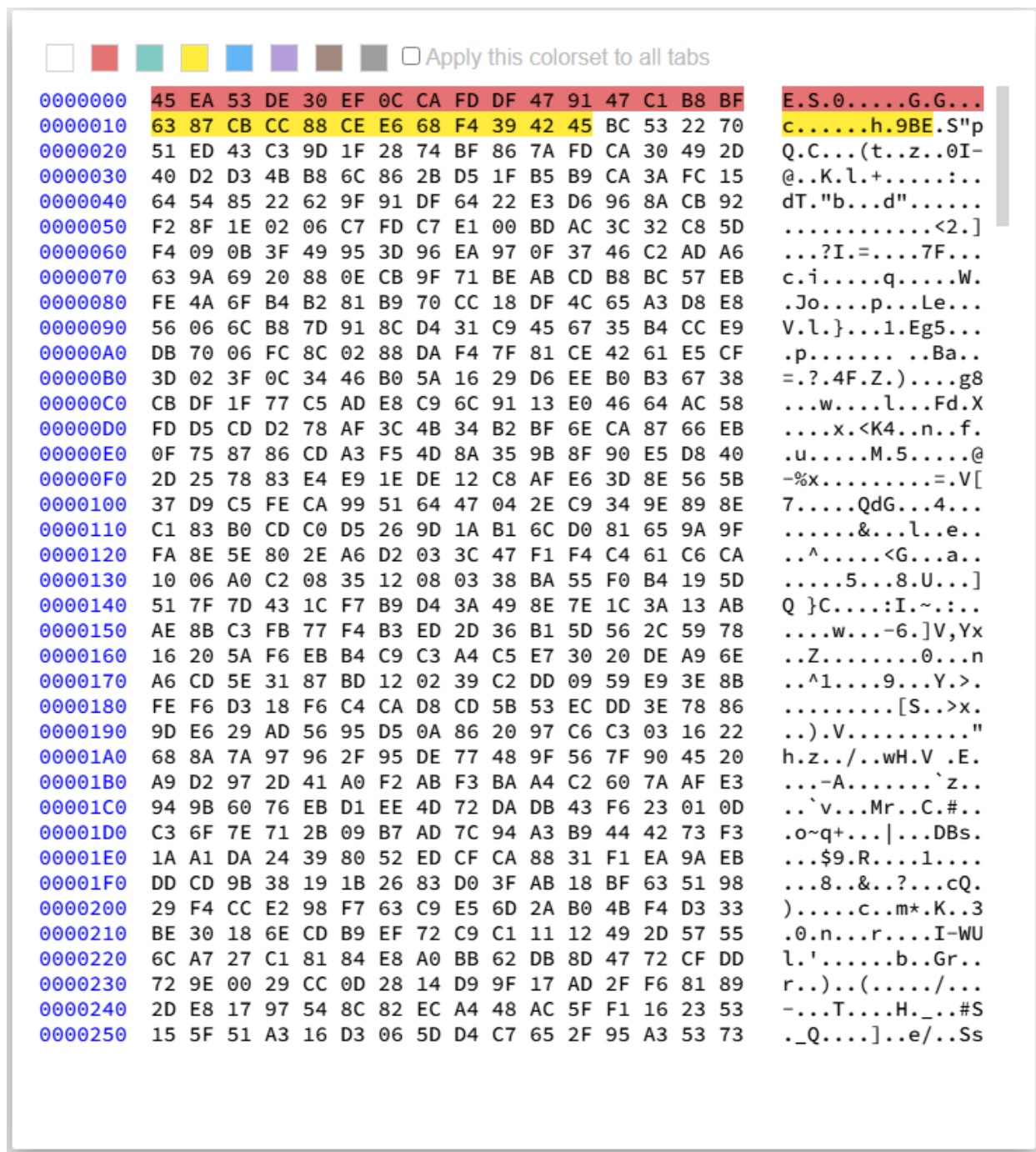


Рисунок 3. Структура зашифрованного файла

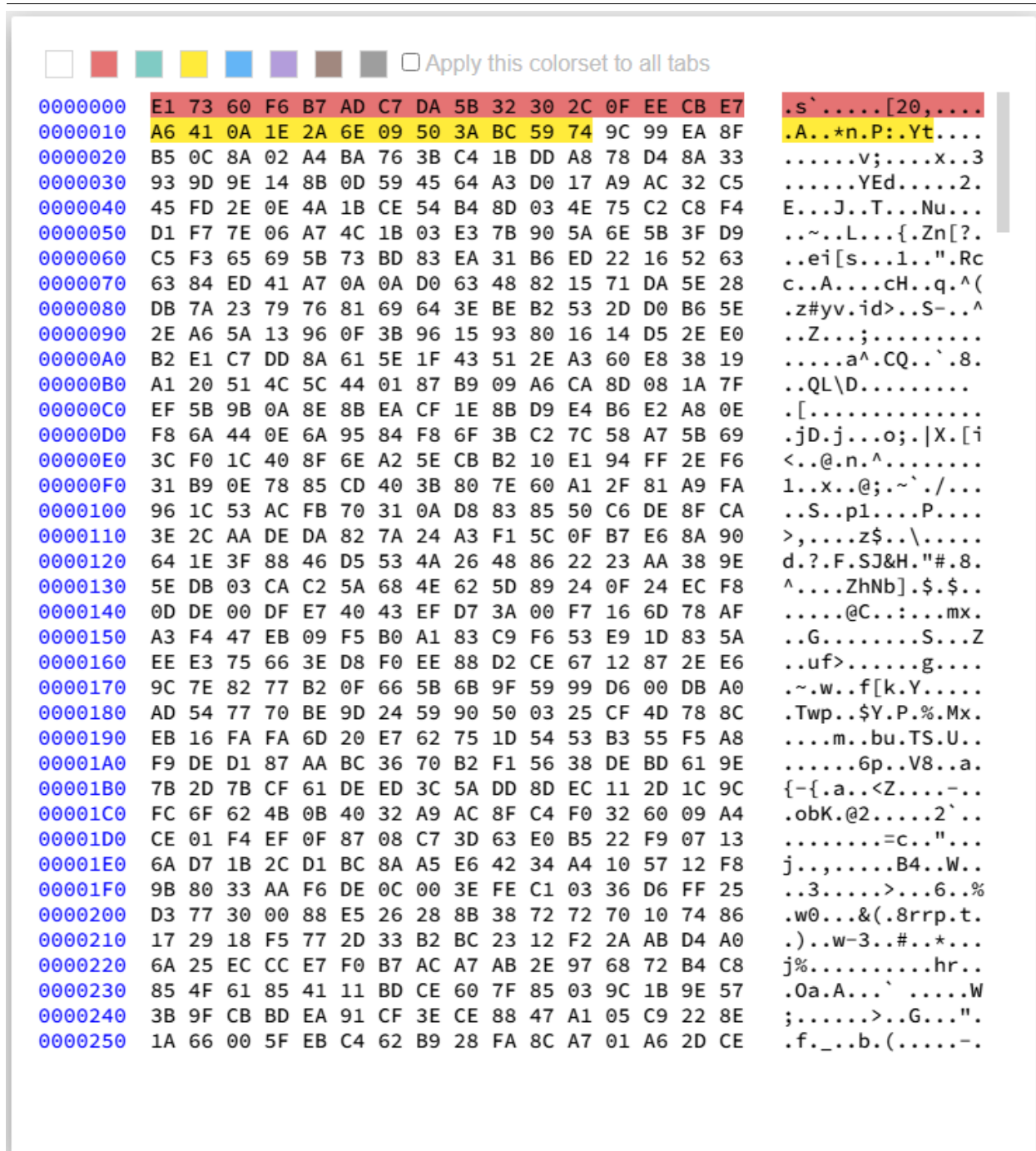


Рисунок 4. Структура зашифрованного файла

Нейросеть создала понятный интерфейс, который не требует обучения. В боковой панели расположены поле ввода пароля с кнопкой показа, индикатор надёжности пароля, слайдер количества итераций PBKDF2 и блок с информацией об алгоритмах (рис. 4).

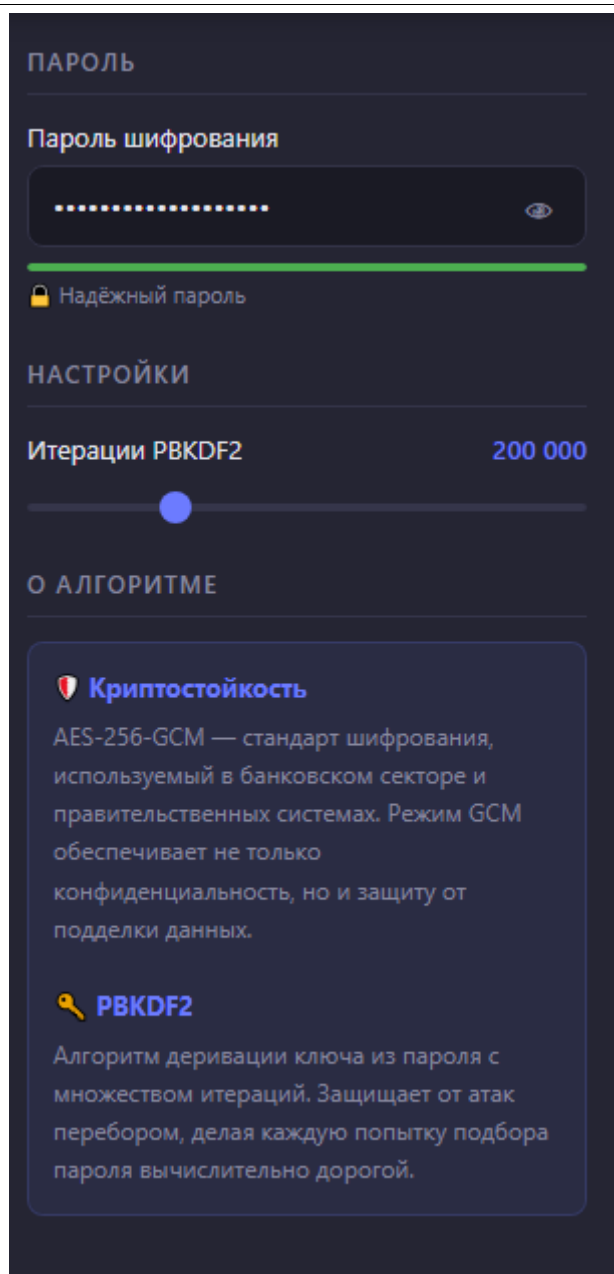


Рисунок 5. Боковая панель

Индикатор надёжности анализирует длину пароля и разнообразие символов (заглавные и строчные буквы, цифры, спецсимволы) и показывает, насколько пароль слабый, средний или надёжный (рис. 5-7).

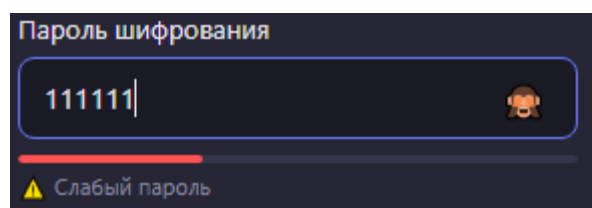


Рисунок 5. Слабый пароль

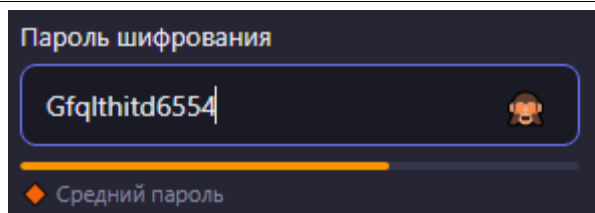


Рисунок 6. Средний пароль

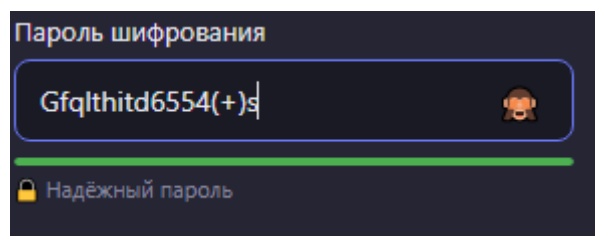


Рисунок 7. Надёжный пароль

В основной области находятся зона перетаскивания файлов, информация о выбранном файле и две кнопки – «Зашифровать» и «Дешифровать». Приложение само определяет режим: если файл имеет расширение .eps, предлагается дешифрование, иначе – шифрование (рис. 8-9).

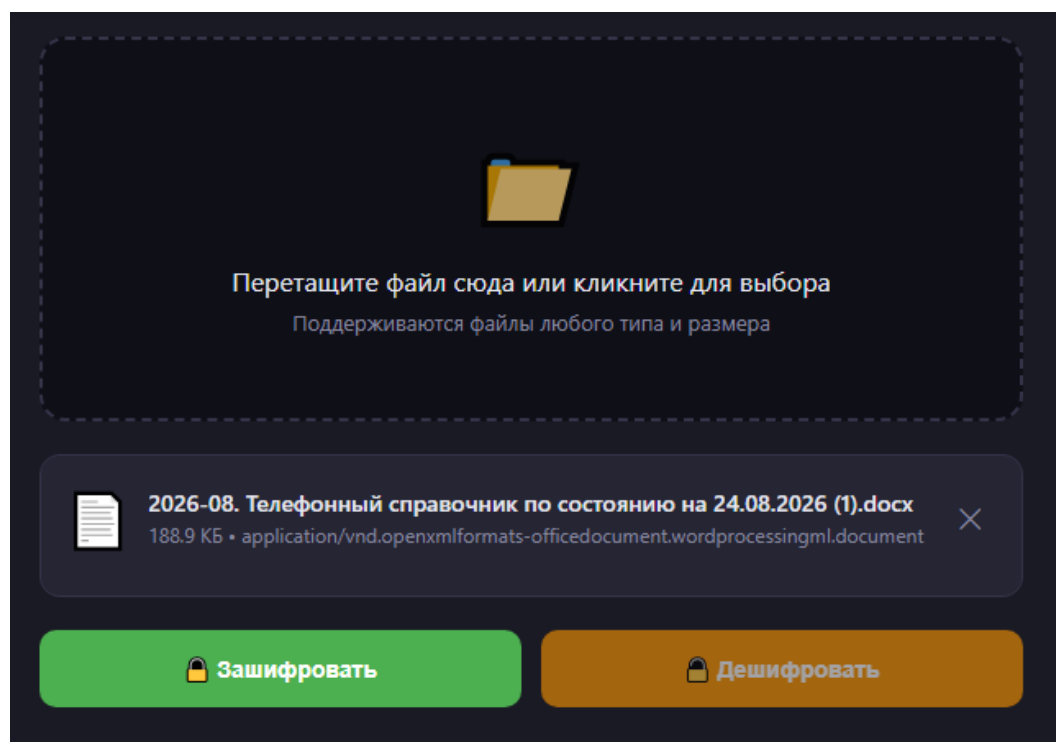


Рисунок 8. Автоопределение режима

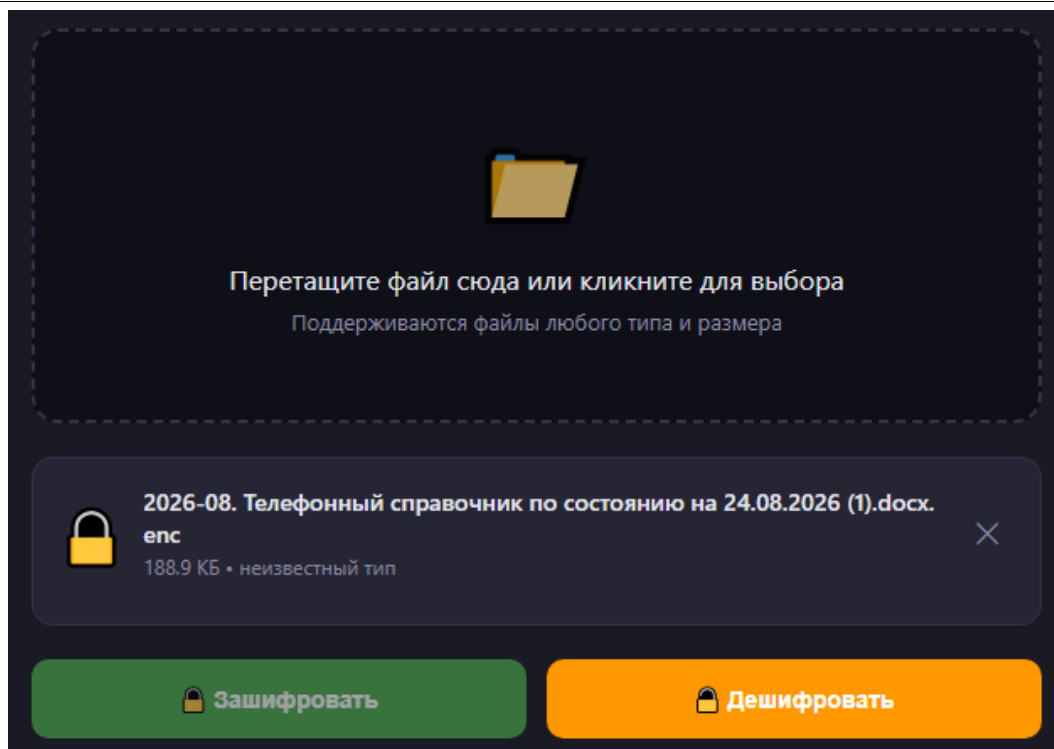


Рисунок 9. Автоопределение режима

Проверка проводилась по бинарному критерию: файл шифровался, а затем дешифровался тем же приложением и тем же паролем. Тестировались файлы разных типов и размеров (таб. 1).

Таблица 1. Результаты проверки зашифрованных файлов

Тип файла	Исходный размер (байт)	Зашифрованный размер (байт)	Результат
Текстовый документ	12 288	12 332	Успешно
Изображение	2 516 582	2 516 626	Успешно
PDF-документ	876 544	876 588	Успешно
Архив	16 462 643	16 462 687	Успешно
Видео	50 646 221	50 646 265	Успешно

Размер зашифрованного файла увеличивается ровно на 44 байта (16 байт соль + 12 байт вектор + 16 байт контрольная метка GCM). Это подтверждает правильность реализации формата.

Проверка с неверным паролем дала ожидаемый результат – ошибку «Неверный пароль или повреждённый файл». Это показывает, что механизм аутентификации AES-GCM работает корректно (рис. 10).

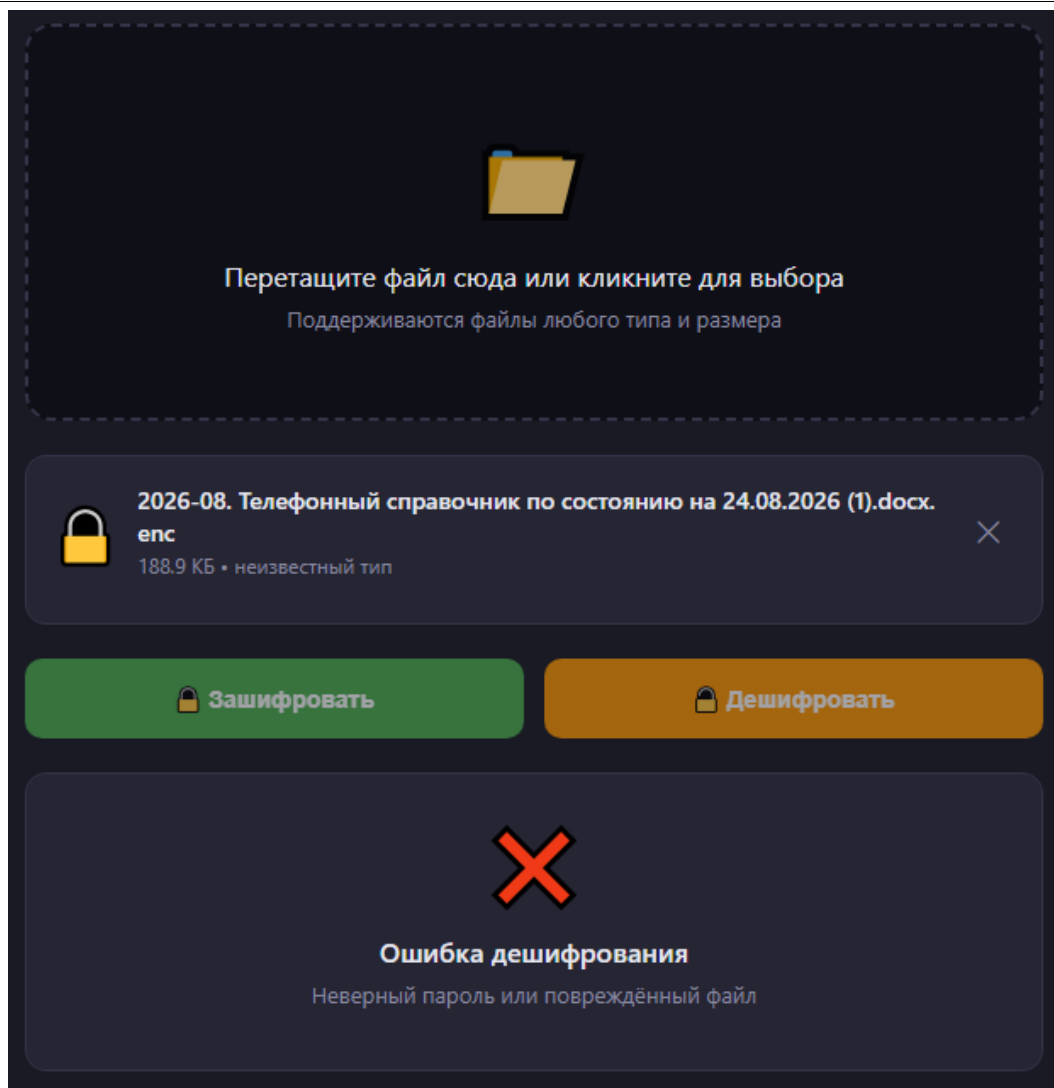


Рисунок 10. Ошибка при неверном пароле

Приложение реализует двухканальную модель безопасности: зашифрованный файл можно отправить по обычной почте или через облако, а пароль передать отдельно – по телефону или при личной встрече. Такой подход соответствует современным стандартам и позволяет обмениваться конфиденциальными данными без настройки специальной инфраструктуры.

Для дешифрования получателю нужны три вещи: сам зашифрованный файл с расширением .enc, копия HTML-файла шифратора и пароль. Приложение работает полностью локально – данные не отправляются на сервер, что исключает утечку через серверные уязвимости.

4 Выводы

Исследование подтверждает, что большие языковые модели хорошо справляются с генерацией криптографических веб-приложений в парадигме без внешних зависимостей. Эксперимент по созданию браузерного шифратора на базе Web Crypto API показал, что нейросеть способна самостоятельно выстроить полный криптографический конвейер: от получения ключей через PBKDF2 до аутентифицированного шифрования по стандарту AES-256-GCM,

с правильной генерацией случайных примитивов и сборкой самодостаточного формата зашифрованного файла.

Главный результат – успешная генерация с первой попытки. Это позволяет сформулировать важное правило: генеративный подход наиболее эффективен в задачах, опирающихся на строгие математические стандарты и однозначные спецификации (криптография, кодирование, протоколы передачи данных).

Бинарный критерий верификации – успешное дешифрование файла – обеспечивает объективную оценку качества кода, без субъективности, которая возникает при визуальной оценке интерфейсов. Тестирование на файлах разных типов и размеров (от 12 КБ до 48 МБ) подтвердило универсальность решения: приложение работает с бинарными данными и не зависит от типа контента.

Разработанное приложение реализует двухканальную модель безопасности и позволяет организовать защищённый обмен данными без серверной инфраструктуры. В перспективе описанный подход можно применять для быстрого создания корпоративных инструментов защиты данных, образовательных платформ по информационной безопасности и автономных криптографических сервисов, работающих полностью на стороне пользователя.

Библиографический список

1. Пащенко Д. С. Использование ИИ-инструментов в конкурентном развитии ИТ-компаний: прикладное исследование 2026 //Информатика. Экономика. Управление-Informatics. Economics. Management. 2026. Т. 5. №. 3. С. 2012-2024.
2. Рогатин С. В., Антошкин В. А. Разработка компьютерных игр по технологии html5 //Информатика и прикладная математика. 2015. №. 21. С.12-16. URL: <https://elibrary.ru/item.asp?id=25030492>
3. Кочедыкова Е. О., Криптографические методы защиты данных //В мире научных открытий: Материалы III Международной студенческой научной конференции. 2019. С. 59-62. URL: <https://lib.ugsha.ru/bitstream/123456789/21996/1/2019-07-59-62.pdf>
4. Елманов И. В. Инженерный подход к ии: использование spec driven development для валидации vibe coding //Вестник науки. 2025. Т. 4. №. 12 (93). С. 716-723. URL: <https://cyberleninka.ru/article/n/inzhenernyy-podhod-kii-ispolzovanie-spec-driven-development-dlya-validatsii-vibe-coding>
5. Титеев А. М., Оптимизация процесса создания автотестов веб-приложений с использованием LLM и структурного анализа HTML //Моделирование и анализ данных. 2025. Т. 15. №4. С. 87–103. URL: https://psyjournals.ru/journals/mda/archive/2025_n4/Titeev