Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет среднего профессионального образования
Специальность 10.02.04 Обеспечение информационной безопасности телекоммуникационных систем

ДОПУСТИТЬ К ЗАЩИТЕ
Заместитель декана по УР
_____Н.В. Цыпукова
«____» _____ 2026 г.

ДИПЛОМНАЯ РАБОТА

на тему: Разработка системы защищенного хранения данных

Исполнитель
студент группы И221 _____ А.М. Куприянов
(подпись, дата)

Руководитель _____ Л.В. Никифорова
(подпись, дата)

Нормоконтроль _____ Е.А. Остроушко
(подпись, дата)

Благовещенск 2026

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет среднего профессионального образования
Специальность 10.02.04 - Обеспечение информационной безопасности телекоммуникационных систем

УТВЕРЖДАЮ
Заместитель декана по УР
_____Н.В. Цыпукова
«_____»_____2026 г.

ЗАДАНИЕ
НА ДИПЛОМНУЮ РАБОТУ

Студент: Куприянов Артём Максимович
Руководитель дипломной работы: Никифорова Лариса Владимировна
1. Тема дипломной работы: Разработка системы защищённого хранения данных
утверждена приказом № 777-уч от 03.04.2026 г.
2. Срок сдачи дипломной работы: 08.06.2026 г.
3. Исходные данные к дипломной работе: НПА, учебная литература, научные статьи, стандарты RFC, документация к технологиям
4. Содержание дипломной работы:
В рамках написания дипломной работы осуществить детальную разработку следующих вопросов:
1. Разработка защищенной системы хранения данных с использованием современных криптографических алгоритмов и механизмов управления ключами
2. Проектирование и реализация интерфейсов для взаимодействия с системой, включая обеспечение аутентификации, шифрования и контроля доступа к данным.
5. Перечень материалов приложения: в случае необходимости предоставить соответствующие теме дипломной работы приложения.
6. Дата выдачи задания 12.01.2026 г.

Руководитель дипломной работы _____ Никифорова Л.В.
Задание принял к исполнению _____ Куприянов А.М.

РЕФЕРАТ

Дипломная работа содержит 64 с., 3 таблицы, 2 приложения, 20 источников

КРИПТОГРАФИЯ, ШИФРОВАНИЕ, РАЗРАБОТКА, PYTHON, API

В работе разработана программа, централизующая процесс управления секретными данными, использующая передовые алгоритмы аутентифицированного шифрования.

Целью данной дипломной работы является анализ существующих алгоритмов аутентифицированного шифрования, выбор подходящих в рамках данной работы, анализ существующих на рынке решений, решающих похожие задачи, составление технического задания на разработку, выбор подходящих технологий для разработки, анализ модели угроз, разработка ПО, и его тестирование.

Актуальность данной работы заключается в необходимости управления секретными данными, такими как API-ключи, токены, пароли в окружении современных приложений. На данный момент большинство команд разработки используют подход переменных окружения для управления секретами, однако этот подход в определенных ситуациях представляет риски для утечки данных, а также при большом количестве секретных данных управление ими в файлах «.env» не всегда является удобным.

Для решения этих проблем в рамках данной работы был разработан сервис, реализующий модель конвертного шифрования, централизующий управление секретными данными через интерфейс командной строки, и выдающий данные по веб-API.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По мере роста объёма цифровой информации и количества прикладных сервисов, важность безопасного хранения данных, а также возможности контроля доступа к этой информации, становится чрезвычайно высокой. Эффективные системы безопасности должны учитывать как криптографическую стойкость алгоритмов, так и надлежащее управление жизненным циклом ключей, проверку пользовательского ввода, контроль доступа и повторяемый операционный процесс. Особую озабоченность вызывает вопрос о том, как объединить требования этих разрозненных систем в единое программное решение, которое можно эффективно использовать на практике, а также масштабировать для будущих задач.

В современных средах разработки команды одновременно должны поддерживать несколько сред: локальные, тестовые, эксплуатационные. Во всех них циркулируют чувствительные данные, такие как:

- ключи шифрования;
- токены API внешних сервисов;
- данные пользователей.

При отсутствии механизма управления секретнными данными возникают следующие проблемы:

- секреты хранятся в файлах «.env», конфигурационных файлах, скриптах, что повышает риск утечки;
- ключи копируются вручную, из-за чего появляются проблемы с контролем версией ключей;
- процессы шифрования и дешифрования выполняются неконсистентно, что приводит к ошибкам интеграций;
- сложно отслеживать кто, когда, и при каких обстоятельствах получил доступ к данным.

При разработке это приводит к снижению скорости внедрения изменений и росту количества инцидентов при взаимодействии команд разработки и DevOps. Для сред эксплуатации это создает риски компрометации данных, простоя сервисов и нарушений требований регуляторов.

В связи с этим возникает необходимость в решении, которое:

- стандартизирует операции с данными;
- позволяет унифицировать работу с данными в разных средах;

- обеспечивает удобный API для разработчиков.

Цель данной работы – разработка программного решения для централизованного управления ключами, обеспечивающего безопасное хранение, повторное шифрование и предоставление зашифрованных данных.

Для достижения указанной цели решаются следующие задачи:

- изучить современные алгоритмы шифрования и выбрать подходящие для данного проекта;
- проанализировать существующие решения на рынке;
- составить функциональные и нефункциональные требования к системе;
- спроектировать модель данных приложения;
- выбрать подходящий технологический стек для приложения;
- проанализировать модель угроз приложения;
- разработать программу;
- разработать модули тестирования программы.

Разработанный в рамках этой дипломной работы проект посвящен разработке программного приложения «Vault», которое обеспечивает безопасное управление ключами и обработку данных. Приложение будет иметь интерфейс командной строки для администрирования, веб-API для предоставления данных после соответствующей авторизации, модуль криптографических операций и уровень хранения, использующий реляционную базу данных. Приложение использует современные алгоритмы AEAD («AES-128-GCM», «AES-256-GCM» и «ChaCha20-Poly1305») для обеспечения конфиденциальности и целостности данных.

Одним из ключевых принципов проектирования является разделение ролей ключей. Например, главный ключ (KEK) никогда не хранится в базе данных, а предоставляется среде, в которой происходит операция, в виде зашифрованного (упакованного) значения.

При этом для шифрования данных используется другой ключ (ключ шифрования данных, DEK).

Такая архитектура разделения ролей ключей ограничивает потенциальную уязвимость для компрометации как главного ключа, так и зашифрованных значений, хранящихся в базе данных, и соответствует основным принципам безопасных систем управления ключами.

1 АНАЛИТИЧЕСКИЙ РАЗДЕЛ

1.1 Теоретические основы криптографии и модели безопасности

1.1.1 Базовые свойства защиты (Триада CIA)

Фундамент теоретических представлений об информационной безопасности образует так называемая «триада CIA» – конфиденциальность, целостность и доступность (Confidentiality, Integrity, Availability). Данные принципы имеют междисциплинарный характер и составляют основу как законодательной, так и технической областей [19].

- **confidentiality (конфиденциальность)** означает, что информация не может быть прочитана или использована лицами, не имеющими соответствующих полномочий. Обеспечение конфиденциальности предполагает решение двуединой задачи: предотвращение несанкционированного доступа к данным и недопущение их перехвата при передаче. Практическая реализация этого принципа осуществляется посредством шифрования, разграничения доступа, физической защиты носителей [19];
- **integrity (целостность)** заключается в ее свойстве существовать в неискаженном виде и не подвергаться случайным или преднамеренным изменениям. Нарушение целостности может быть следствием как злонамеренных действий (внедрение вредоносного ПО, модификация данных), так и технических сбоев. Защита целостности требует применения методов контроля (контрольные суммы, хэш-функции) и предотвращения (резервирование, управление доступом) [19];
- **availability (доступность)** предполагает, что авторизованные пользователи получают доступ к информации и связанным с ней активам всегда, когда это необходимо. Угрозы доступности (например, DDoS-атаки) способны парализовать деятельность организаций, критически зависимых от информационных систем. Обеспечение доступности достигается резервированием компонентов, созданием отказоустойчивых конфигураций, планированием восстановления после сбоев [19].

Теоретический анализ информационной безопасности неразрывно связан с классификацией угроз. Под угрозой информационной безопасности понимается совокупность условий и факторов, создающих потенциальную или реальную опасность нарушения безопасности информации [19].

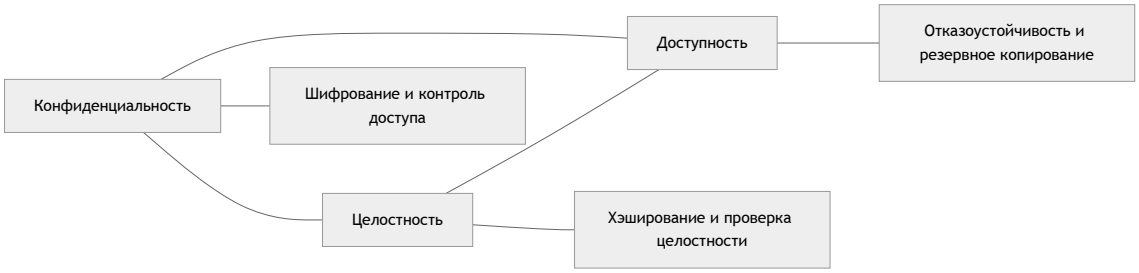

Рисунок 1 – Взаимодействие компонентов триады CIA

1.1.2 Используемые криптографические термины

Алгоритмы аутентифицированного шифрования (Authenticated Encryption – AE) представляют собой класс алгоритмов шифрования, которые также позволяют аутентифицировать зашифрованные данные, т. е. обеспечивают возможность проверки их целостности. Алгоритмы аутентифицированного шифрования с присоединёнными данными (Authenticated Encryption with Associated Data – AEAD) можно считать частным случаем AE-алгоритмов, поскольку они также обеспечивают как конфиденциальность, так и целостность зашифрованных данных. Помимо этого, AEAD-алгоритмы позволяют обеспечить целостность и аутентичность некоторых дополнительных данных, для которых не требуется конфиденциальность. Дополнительные данные при этом не шифруются. Необходимость в появлении такого класса алгоритмов обусловлена тем, что во многих приложениях требуется, помимо передачи некоторых данных в зашифрованном виде, передавать также определённую открытую информацию с контролем её целостности/аутентичности. Данная возможность важна, когда взаимодействие с устройствами с небольшими ресурсами ограничено незначительным промежутком времени.

Разработанный проект использует алгоритмы симметричного шифрования AEAD «AES-128-GCM», «AES-256-GCM» и «ChaCha20-Poly1305» для проверки целостности данных.

Ключ шифрования ключа (Key Encryption Key, KEK) – ключ, используемый для шифрования другого ключа. (обычно ключей шифрования трафика – Traffic Encryption Key – TEK) для передачи или хранения [2].

Ключ шифрования данных (Data Encryption Key, DEK) – Ключ, используемый для шифрования и расшифрования данных, не являющихся

другими ключами [4].

Однократно используемое число (nonce) – Значение, используемое в криптографических протоколах, которое никогда не повторяется с одним и тем же ключом [1].

Имитовставка (tag) – Битовая строка, добавляемая к сообщению с целью обеспечения возможности обнаружения подмены и/или имитации сообщения и являющаяся результатом применения к нему криптографической хеш-функции, зависящей от ключа [1].

Оборачивание ключа (key wrapping) – это криптографическая операция шифрования ключевого материала с использованием ключа шифрования ключей (KEK), направленная на обеспечение конфиденциальности и целостности защищаемого ключа при его хранении или передаче.

Извлечение ключа (key unwrapping) – это операция, обратная оборачиванию, при которой исходный ключ восстанавливается из защищенного (обернутого) представления с использованием ключа шифрования ключей (KEK).

1.1.3 Общие термины, используемые в работе

Dev (development) – среда, в которой выполняется разработка, тестирование и отладка программного обеспечения.

Prod (production) – промышленная среда эксплуатации, в которой система используется для обработки реальных данных и предъявляет повышенные требования к надёжности и безопасности.

Среда **staging (предпродуктивная среда)** представляет собой изолированную среду, максимально приближенную к «production», предназначенную для проверки корректности работы системы перед развертыванием в промышленной эксплуатации. В данной среде проводится интеграционное тестирование, проверка конфигурации и сценариев развертывания.

1.1.4 Алгоритм PBKDF2

В разработанной программе для вывода ключей используется криптографическая функция PBKDF2.

Алгоритм вывода ключей PBKDF2 (Password-Based Key Derivation Function 2) необходим для вывода стойкого криптографически ключа из строки пользователя. Алгоритм определен в стандарте RFC 8018 [5] и основан на применении PRF (псевдослучайной функции).

Суть алгоритма заключается в увеличении сложности перебора (Brute Force) паролей за счет применения соли и большого числа итераций. Схема работы алгоритма PBKDF2 представлена на рисунке.

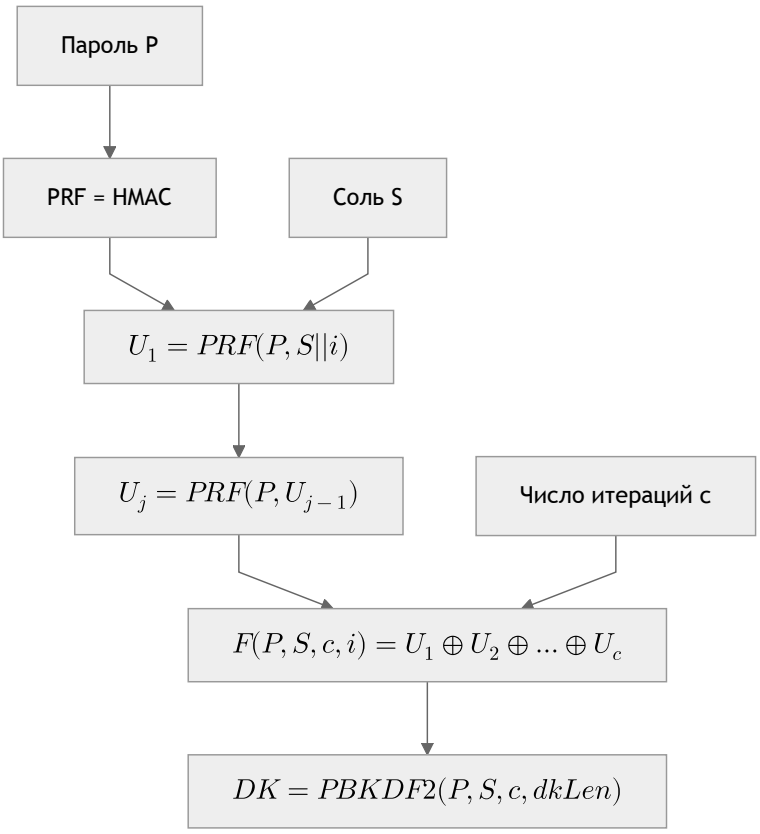

Рисунок 2 – Схема работы алгоритма PBKDF2

Алгоритм принимает на вход следующие параметры:

- пароль P;
- соль S;
- число итераций c;
- длину ключа $dkLen$.

В основе PBKDF2 лежит псевдослучайная функция, обычно выбирается HMAC.

1) Начальное вычисление

Сначала выбирается значение:

$$U_1 = PRF(P, S||i) \tag{1}$$

где

- i – индекс блока выходного ключа;
- || – операция сложения строк (конкатенация).

2) Итеративное преобразование

Далее выполняется последовательное вычисление значений:

$$U_j = PRF(P, U_{j-1}), j = 2, 3, ..., c \quad (2)$$

Таким образом, результат предыдущей итерации используется как вход для следующей, что увеличивает вычислительную сложность алгоритма.

3) Агрегация результатов

Итоговое значение функции F вычисляется как побитовое сложение по модулю 2 (XOR) всех промежуточных значений:

$$F(P, S, c, i) = U_1 \oplus U_2 \oplus \cdots \oplus U_c \quad (3)$$

4) Формирование итогового ключа

Выходной ключ формируется путём конкатенации результатов функции F для различных значений индекса i:

$$DK = PBKDF2(P, S, c, dkLen) \quad (4)$$

1.1.5 Алгоритм AES-GCM

В разработанной системе для шифрования данных используются алгоритмы семейства AES-GCM.

AES-GCM (Advanced Encryption Standard в режиме Galois/Counter Mode) представляет собой симметричный алгоритм аутентифицированного шифрования (AEAD), обеспечивающий одновременно конфиденциальность и контроль целостности данных. Алгоритм стандартизирован в NIST SP 800-38D [3].

Суть алгоритма заключается в комбинировании режима потокового шифрования (CTR) и вычисления аутентификационного тега с использованием функции GHASH. Схема работы алгоритма AES-GCM представлена на рисунке 3.

Алгоритм принимает на вход следующие параметры:

- секретный ключ K;
- nonce (инициализационный вектор) N;

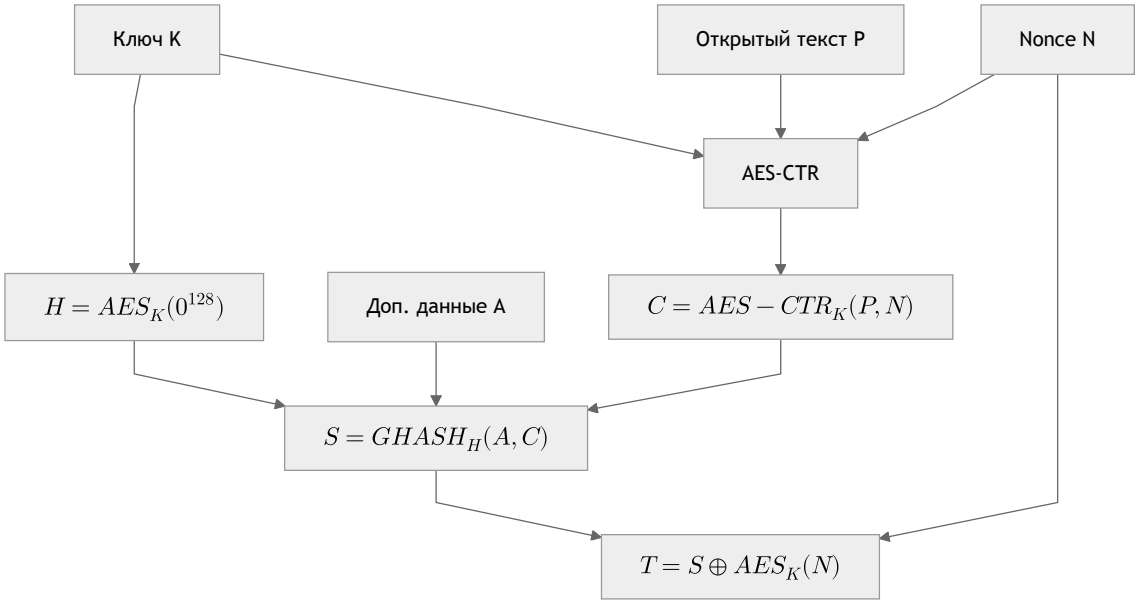

Рисунок 3 – Схема работы алгоритма AES-GCM

- открытый текст P;
- дополнительные аутентифицируемые данные A.

В результате работы алгоритма формируются:

- зашифрованный текст C;
- тег аутентификации T.

1) Формирование вспомогательного ключа

Сначала вычисляется вспомогательный ключ для функции GHASH:

$$H = AES_K(0^{128}) \tag{5}$$

где H используется как ключ хеширования в поле $GF(2^{128})$.

2) Шифрование данных

Шифрование выполняется в режиме счетчика (CTR):

$$C = AES\text{-}CTR_K(P, N) \tag{6}$$

В данном режиме генерируется поток ключа, который складывается по модулю 2 с открытым текстом.

3) Вычисление тега аутентификации

Для обеспечения целостности данных вычисляется значение GHASH:

$$S = GHASH_H(A, C) \tag{7}$$

где учитываются как зашифрованные данные, так и дополнительные аутентифицируемые данные.

4) Формирование итогового тега

Итоговый тег аутентификации определяется как:

$$T = S \oplus AES_K(N) \tag{8}$$

5) Результат работы алгоритма

Итогом работы AES-GCM является пара значений:

$$(C, T) = AES\text{-}GCM_K(P, N, A) \tag{9}$$

1.1.6 Алгоритм ChaCha20-Poly1305

В разработанной системе наряду с AES-GCM используется алгоритм ChaCha20-Poly1305.

ChaCha20-Poly1305 представляет собой алгоритм аутентифицированного шифрования (AEAD), сочетающий потоковый шифр ChaCha20 и механизм аутентификации сообщений Poly1305. Алгоритм стандартизирован в RFC 8439 [6].

Основная идея алгоритма заключается в использовании ChaCha20 для шифрования данных и Poly1305 для вычисления тега аутентификации. Схема работы алгоритма представлена на рисунке 4.

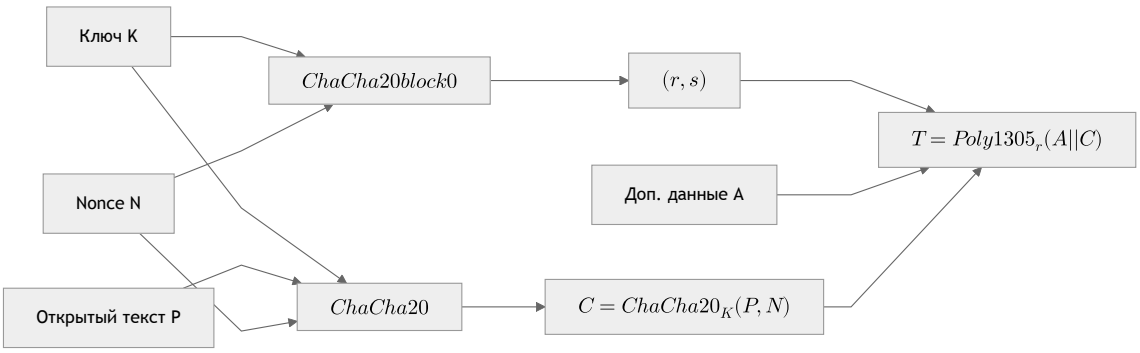

Рисунок 4 – Схема работы алгоритма ChaCha20-Poly1305

Алгоритм принимает на вход следующие параметры:

- секретный ключ K;
- nonce N;
- открытый текст P;
- дополнительные аутентифицируемые данные A.

В результате работы формируются:

- зашифрованный текст C;
- тег аутентификации T.

1) Генерация ключа Poly1305

На первом этапе с использованием ChaCha20 генерируется одноразовый ключ для алгоритма Poly1305:

$$(r, s) = ChaCha20_K(N, 0) \tag{10}$$

где первый блок выходного потока используется как ключ аутентификации.

2) Шифрование данных

Шифрование выполняется с использованием потокового шифра ChaCha20:

$$C = ChaCha20_K(P, N) \tag{11}$$

Открытый текст складывается по модулю 2 с псевдослучайным потоком ключа.

3) Формирование данных для аутентификации

Для вычисления тега формируется последовательность:

$$M = A \parallel C \tag{12}$$

где || обозначает операцию конкатенации.

4) Вычисление тега аутентификации

Тег вычисляется с использованием алгоритма Poly1305:

$$T = Poly1305_r(M) \tag{13}$$

5) Результат работы алгоритма

Итогом работы является пара значений:

$$(C, T) = ChaCha20\text{-}Poly1305_K(P, N, A) \quad (14)$$

1.2 Анализ предметной области

1.2.1 Особенности задачи для сред разработки и production

Основная проблема при работе с чувствительными данными заключается в том, что требования безопасности часто внедряются во вред удобству.

Для среды разработки важны:

- простота локального запуска и тестирования;
- понятный CLI интерфейс для типовых операций;
- быстрая диагностика ошибок в формате, удобном для CI/CD.

Для production-среды важны:

- недоступность мастер-ключа в базе данных и репозитории;
- шифрование с контролем целостности (AEAD);
- защищенный сетевой доступ (HTTPS + авторизация);
- контролируемая схема хранения с миграционной совместимостью.

Данный проект решает этот конфликт через разделение ответственности компонентов: CLI используется для администрирования и загрузки данных, API – для строго ограниченной выдачи данных. При этом криптографическая логика и валидация вынесены в отдельные модули и не дублируются между сценариями.

1.2.2 Анализ существующих решений (конкурентов)

На данный момент на рынке уже существуют зрелые системы управления чувствительными данными. Их использование оправдано в крупных инфраструктурах, однако для учебных, исследовательских и части прикладных задач они могут быть избыточны по сложности и стоимости внедрения.

1) HashiCorp Vault

HashiCorp Vault – это кроссплатформенное программное обеспечение с открытым исходным кодом, предназначенное для централизованного хранения, защиты и управления доступом к чувствительным данным (секретам), таким как пароли, API-ключи, сертификаты и токены.

Сильные стороны:

- широкий набор механизмов аутентификации и авторизации;
- динамические секреты и развитые политики доступа;

- интеграции с Kubernetes, облачными провайдерами и PKI.

Ограничения для данной задачи:

- существенная операционная сложность (инициализация, unseal, сопровождение кластера);
- более высокий порог входа для учебного проекта и небольших команд.

2) Облачные сервисы (AWS KMS/Secrets Manager, Azure Key Vault, Google Cloud KMS/Secret Manager)

В современной практике разработки выделяют две основные группы облачных инструментов: KMS (Key Management Service) для управления ключами шифрования и Secret Manager для хранения паролей и токенов.

2.1) AWS (Amazon Web Services)

- AWS KMS – Централизованное управление ключами (CMK). Использует аппаратные модули безопасности (HSM) уровня FIPS 140-2. Основная задача – шифрование дисков (EBS), баз данных (RDS) и объектов (S3);
- AWS Secrets Manager – Сервис для хранения секретов. Главная функциональность – нативная ротация паролей для баз данных AWS (RDS, Redshift) без участия разработчика.

2.2) Microsoft Azure

- Azure Key Vault – универсальный сервис «три в одном». Позволяет хранить:
  - Secrets (пароли, строки подключения);
  - Keys (ключи для шифрования дисков и данных);
  - Certificates (управление SSL/TLS сертификатами).

Интегрируется с Azure Managed Identities, что позволяет приложениям получать доступ к секретам без использования паролей.

2.3) Google Cloud Platform (GCP)

- Cloud KMS – Позволяет управлять симметричными и асимметричными ключами. Поддерживает внешние менеджеры ключей (EKM), если данные должны шифроваться ключами вне облака Google;
- Secret Manager – Глобальный сервис для хранения конфиденциальных данных с удобным версионированием секретов и разделением доступа через Cloud IAM.

Сильные стороны облачных сервисов:

- высокая надежность;

- интеграция с конкретными облачными сервисами;
- встроенное логирование и IAM-модели.

Ограничения для данной задачи:

- привязка к конкретному облачному провайдеру;
- зависимость от внешней инфраструктуры.

3) Корпоративные PAM/Secrets-решения (например, CyberArk, Delinea)

PAM (Privileged Access Management) – это комплекс стратегий и технологий для контроля, мониторинга и аудита привилегированного доступа к критическим ресурсам организации. В отличие от легковесных инструментов для DevOps, PAM-системы исторически сфокусированы на управлении доступом администраторов и операторов (Human-to-Machine).

Сильные стороны:

- интеграция с корпоративной политикой информационной безопасности;
- глубокие процессы аудита и контроля.

Ограничения для данной задачи:

- высокая стоимость;
- избыточность в малых и средних компаниях.

4) Open-source self-hosted решения (например, OpenBao, Infisical self-hosted)

Сильные стороны:

- контроль над инфраструктурой и отсутствие жесткой привязки к конкретному облаку;
- более низкий порог по стоимости владения по сравнению с enterprise PAM;
- гибкость кастомизации под внутренние процессы команды.

Ограничения для данной задачи:

- ответственность за эксплуатацию (обновления, бэкапы, мониторинг) полностью на команде;
- часто менее зрелая экосистема интеграций по сравнению с лидерами рынка.

5) SaaS-сервисы для разработчиков (например, Doppler, 1Password Secrets Automation)

Это полностью облачные платформы, которые берут на себя всю сложность управления инфраструктурой (шифрование, репликацию, бэкапы), предо-

ставляя разработчикам только интерфейс и API. Основной акцент сделан на Developer Experience (DX) – максимальную простоту интеграции секретов в код.

Сильные стороны:

- быстрый старт и удобная интеграция в CI/CD;
- низкая операционная нагрузка на команду;
- хороший UX для небольших и средних продуктовых команд.

Ограничения для данной задачи:

- зависимость от внешнего провайдера и его модели безопасности;
- ограниченная гибкость в нетиповых сценариях и в локальных окружениях;
- возможный рост затрат при масштабировании.

6) GitOps-инструменты шифрования конфигураций (например, Mozilla SOPS, git-crypt, Sealed Secrets)

Сильные стороны:

- удобная работа с секретами в Git-репозиториях и workflow инфраструктурного кода;
- простая интеграция в существующие процессы поставки конфигураций;
- низкий порог внедрения для задач «шифровать и хранить в репозитории».

Ограничения для данной задачи:

- это не полноценные платформы управления жизненным циклом секретов;
- ограниченные возможности централизованной выдачи секретов по API и детального policy-management;
- при росте масштаба требуется дополнительная инфраструктура и процессы контроля.

1.2.3 Выводы по анализу и обоснование выбора собственного решения

Результаты проведенного анализа схематично показаны на рисунке 5.

Также сравнительный анализ решений представлен в таблице 1.

Таблица 1 – Сравнительный анализ существующих решений для управления секретами

Тип решения	Сильные стороны	Ограничения для данной задачи
1	2	3
SaaS-сервисы (Doppler, 1Password Secrets Automation)	Быстрый старт; удобная интеграция в CI/CD; низкая операционная нагрузка;	Зависимость от провайдера; ограниченная гибкость; возможный рост затрат
Корпоративные PAM/Secrets-решения (CyberArk, Delinea)	Интеграция с корпоративной политикой информационной безопасности; глубокие процессы аудита и контроля	Высокая стоимость; избыточность для малых и средних компаний и учебных задач
HashiCorp Vault	Широкий набор механизмов аутентификации и авторизации; динамические секреты; развитые политики доступа; интеграции с Kubernetes, облачными провайдерами и PKI	Существенная операционная сложность; более высокий порог входа для учебного проекта и небольших команд
Облачные сервисы (AWS KMS / Secrets Manager, Azure Key Vault, Google Cloud KMS / Secret Manager)	Высокая надёжность; интеграция с конкретными облачными сервисами; встроенное логирование и IAM-модели	Привязка к конкретному облачному провайдеру; зависимость от внешней инфраструктуры
Open-source self-hosted решения (OpenBao, Infisical self-hosted)	Контроль над инфраструктурой; отсутствие привязки к облаку; более низкая стоимость владения; гибкость кастомизации	Ответственность за эксплуатацию полностью лежит на команде; менее зрелая экосистема интеграций
GitOps-инструменты (SOPS, git-crypt, Sealed Secrets)	Удобная работа с секретами в Git; простая интеграция; низкий порог внедрения	Не являются полноценными системами управления секретами; ограниченный lifecycle management

Выполненный анализ показал, что существующие на данный момент платформы хорошо решают задачу управления секретами в промышленном масштабе, но не оптимальны для сценария, в котором необходимы:

- прозрачная архитектура;
- контролируемая сложность внедрения;
- отсутствие зависимости от внешних облачных сервисов;
- единый поток работы для «dev» и «prod» с воспроизводимыми коман-

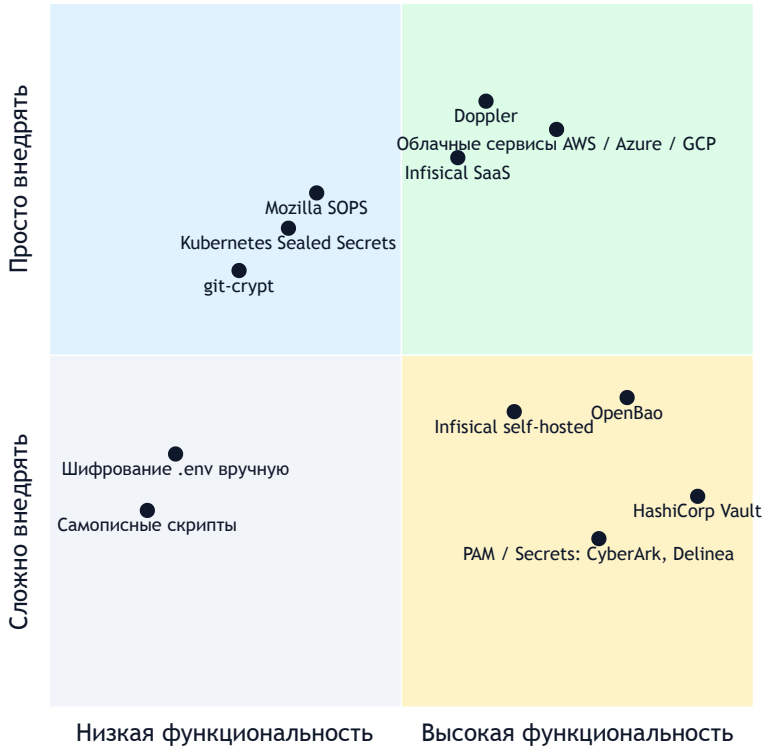

Рисунок 5 – Сравнение решений для управления секретами

дами.

Именно поэтому в данной работе выбран подход создания специализированного сервиса. Он решает ключевые практические потребности: безопасное хранение DEK в обернутом виде, использование KEK вне базы данных, перешифрование данных между алгоритмами, авторизованный доступ по API и тестируемость всех критических операций. Такой подход обеспечивает баланс между реализуемостью и безопасностью.

2 ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1 Постановка задачи и требования к системе

Назначение разрабатываемой системы заключается в реализации безопасного процесса работы с криптографическими ключами и данными в средах разработки и «production». Постановка задачи состоит из следующих частей:

- функциональные требования;
- нефункциональные требования;
- ограничения реализации;
- критерии приемки.

Функциональные требования определяют минимально необходимый набор возможностей системы. Они описывают, какие операции пользователь и администратор должны выполнять в рамках единого эксплуатационного контура.

- управление ключами через CLI. Система должна поддерживать генерацию и импорт ключевого материала командами `generate-kek`,`generate-dek`,`add-kek` и `add-dek`. Это необходимо для того, чтобы администратор мог как создавать новые ключи внутри системы, так и подключать заранее подготовленные значения. Одновременно интерфейс обязан валидировать формат данных, допустимые алгоритмы и логические имена, чтобы в хранилище не попадали некорректные или двусмысленные записи;

- безопасная загрузка и перешифрование данных. Команда `load` должна принимать входной контейнер, выполнять его расшифрование с использованием исходного DEK и затем повторное шифрование в целевой алгоритм с использованием выходного DEK. Такое требование отражает прикладной сценарий миграции данных между ключами и алгоритмами, где важно не просто хранить шифртекст, а контролируемо преобразовывать его внутри доверенного процесса;

- хранение данных и метаданных в БД. После выполнения криптографических операций система должна сохранять не только полезную нагрузку, но и достаточный набор служебных данных: имена ключей, алгоритмы, заголовок контейнера, временные метки и авторизационный секрет в защищенном виде. Это делает каждую запись самодостаточной для последующего чтения через API, но не приводит к хранению открытых секретов;

- выдача данных по API. Эндпоинт `GET/data/{name}` должен возвращать расшифрованные данные только после успешной проверки Bearer-авторизации,

корректного чтения ключевой записи и успешного извлечения целевого DEK. Тем самым система выступает не только как инструмент подготовки контейнеров, но и как прикладной сервис контролируемого доступа к данным;

- сохранение нового открытого текста без вспомогательных скриптов. Поскольку в реальном эксплуатационном контуре возникают не только задачи перешифрования, но и задачи первичного ввода данных, система должна поддерживать сценарий добавления новых открытых данных. Для этого реализуется команда `store-plaintext`, которая читает открытый текст из файла, автоматически шифрует его выбранным алгоритмом и сохраняет результат в `data_records`. Это требование закрывает практический пробел между криптографическим API и пользовательским сценарием административной загрузки данных.

Нефункциональные требования описывают следующие характеристики:

- безопасность. Это базовое нефункциональное требование для всего проекта. Система должна использовать современные AEAD-алгоритмы («AES-128-GCM», «AES-256-GCM», «ChaCha20-Poly1305»), чтобы одновременно обеспечивать конфиденциальность и контроль целостности данных. Дополнительно требуется строгое разделение ролей KEK и DEK: master KEK не должен сохраняться в БД, а DEK должны храниться только в обернутом виде. Такое требование снижает последствия компрометации хранилища и соответствует принятой модели конвертного шифрования;

- легкость тестирования. Поведение системы должно быть формализуемым и воспроизводимо проверяемым через unit-, интеграционные и сценарные тесты. Для защищенного сервиса этого недостаточно описать словами: ключевые инварианты, такие как запрет хранения открытых DEK, отказ при неверном Bearer-токене, проверка заголовка контейнера и обязательность master KEK, должны подтверждаться автоматическими проверками;

- воспроизводимость. Сервис должен одинаково работать в dev- и production-подобных средах при различии только параметров конфигурации, а не программной логики. Это означает единый набор CLI-команд, единые форматы контейнеров, единое поведение API и предсказуемую работу через переменные окружения;

- переносимость. Решение не должно быть жестко привязано к конкретному облачному провайдеру, внешнему KMS или специализированной

платформе. Такой подход важен для учебно-прикладного проекта, поскольку позволяет запускать проект локально, в лаборатории или на выделенном сервере без внешней инфраструктурной зависимости.

Ограничения и допущения описывают рамки текущей версии разработки:

- Master KEK хранится только в env-переменной;
- открытый KEK не хранится в базе данных и не фиксируется в репозитории;
- доступ к данным авторизуется через Bearer-токен, который хранится в виде PBKDF2-хеша[1];
- в качестве хранилища используется реляционная база данных.

Критерии приемки обозначают условия, при которых задача считается выполненной:

- криптографические преобразования выполняются корректно;
- CLI-сценарии завершаются успешно;
- API-сценарии завершаются успешно;
- автотесты проекта выполняются успешно.

2.2 Проектирование системы и модели данных

2.2.1 Модули приложения

Проектирование системы выполнено в модульно-слоистой архитектуре, где каждый слой решает ограниченный набор задач и имеет явные точки взаимодействия. Распределение модулей по семантическим слоям представлено в таблице 2.

Таблица 2 – Распределение модулей по семантическим слоям

Семантическое значение	Модуль
Точка входа	`main.py`,`vault/cli.py`,`vault/api.py`
Сервисный слой	`vault/crypto.py`,`vault/key_manager.py`,`vault/auth.py`
Слой доступа к данным	`vault/db.py` (репозиторий и ORM-модели)
Слой валидации контрактов	`vault/schemas.py`,`vault/db_schemas.py`
Конфигурационный слой	`vault/settings.py` и переменные окружения

Такое разделение снижает связанность кода: CLI и API используют общие криптографические и валидационные модули, а работа с базой данных изолирована в репозитории `VaultRepository`.

Схема модулей приложения представлена на рисунке 6.

[1] См. раздел с описанием работы алгоритма PBKDF2 на с. 10

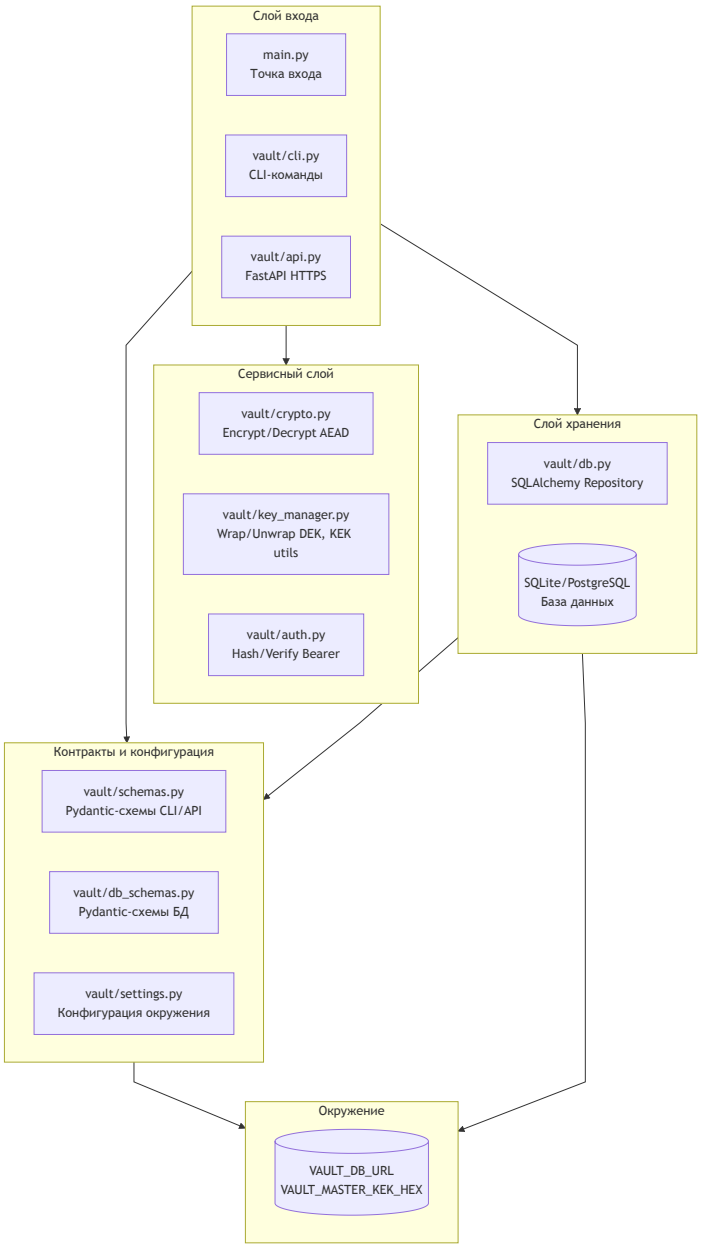

Рисунок 6 – Схема модулей приложения

2.2.2 Модель данных

Модель данных построена на двух ключевых сущностях: `keys`[2] и `data_records`[3]. Диаграмма сущностей представлена на рисунке 7.

Таблица `keys` хранит ключевой материал и метаданные ключей:

1) Ограничение уникальности `uq_keys_name_type` не допускает дубли пары (`name`,`key_type`).

2) Для `key_type="kek"` поле `key_hex` хранит ключ в шестнадцатеричном

[2]См. определение таблицы `key_records`: метка 5, рисунок 37, приложение А, с. 80

[3]См. определение таблицы `data_records`: метка 6, рисунок 37, приложение А, с. 81

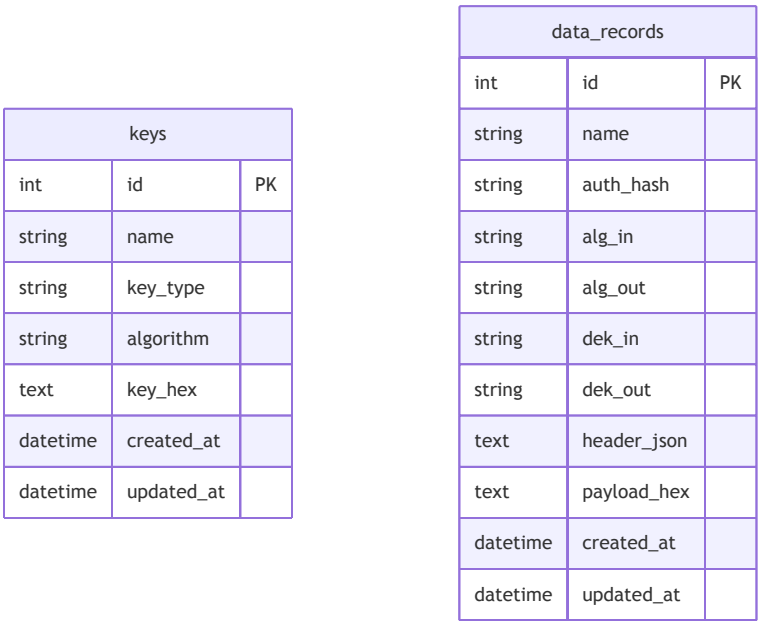

Рисунок 7 – Таблицы keys и data_records

представлении.

3) Для key_type="dek" поле key_hex хранит обернутое представление DEK.

Таблица data_records хранит зашифрованные полезные данные и параметры их обработки:

1) Ограничение уникальности uq_data_records_name обеспечивает upsert-семантику по имени записи.

2) Поле auth_hash хранит PBKDF2-хеш, а не исходный заголовок авторизации.

3) Поля alg_in и alg_out фиксируют входной и выходной алгоритмы в процессе перешифрования.

4) Поля dek_in и dek_out содержат ссылки на логические имена DEK, используемые в load и GET/data/{name}.

5) Поля header_json и payload_hex хранят контейнер шифртекста: заголовок AEAD (alg/nonce/tag) и шифртекст в шестнадцатеричном представлении.

На уровне контрактов данных применяются строгие ограничения:

1) Имена сущностей ограничены по длине (1..128), алгоритмы (1..64), заголовок авторизации (1..512)

2) payload_hex валидируется как корректная шестнадцатеричная строка

3) header_json проверяется как JSON-объект с ожидаемой структурой

4) Обернутый DEK валидируется как JSON с полями nonce,wrapped,tag

2.2.3 Потоки обработки

Ключевые потоки обработки строятся вокруг двух сценариев.

- сценарий load:

1) Валидация входных параметров команды и формата контейнера.

2) Получение dek_in и dek_out из таблицы keys.

3) Извлечение DEK через master KEK из VAULT_MASTER_KEK_HEX.

4) Дешифрование входных данных (alg_in) и повторное шифрование (alg_out).

5) Upsert записи в data_records с обновлением auth_hash, header_json,payload_hex и ссылок на DEK.

Диаграмма сценария представлена на рисунке 8;

- сценарий GET/data/\{name\}

1) Поиск записи в data_records по имени.

2) Проверка Authorization:Bearer против auth_hash.

3) Получение dek_out, извлечение ключа и валидация заголовка контейнера.

4) Расшифрование payload_hex и возврат открытого текста в API-ответе.

Диаграмма сценария представлена на рисунке 9;

2.2.4 Обоснование выбора технологического стека

Выбор технологического стека для проекта сделан на основе двух критериев:

1) обеспечение корректной и безопасной реализации криптографических операций.

2) минимизация трудозатрат на разработку и верификацию в рамках дипломного цикла.

Фактический стек реализации:

1) Python 3.11+ – основной язык реализации CLI, API и сервисной логики.

2) cryptography – реализация AEAD-примитивов[4] (AES-128-GCM, AES-256-GCM, ChaCha20-Poly1305) и криптоопераций на уровне production-библиотеки.

3) FastAPI + uvicorn – HTTP(S)-интерфейс и запуск API-сервера.

[4]См. раздел с определением AEAD-алгоритмов на с. 9.

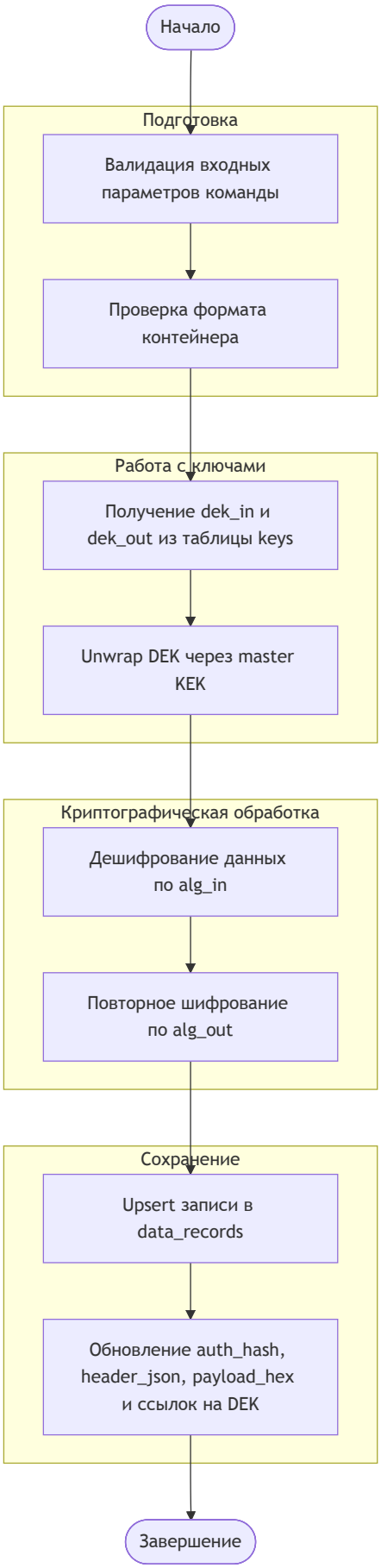

Рисунок 8 – Диаграмма сценария `load`

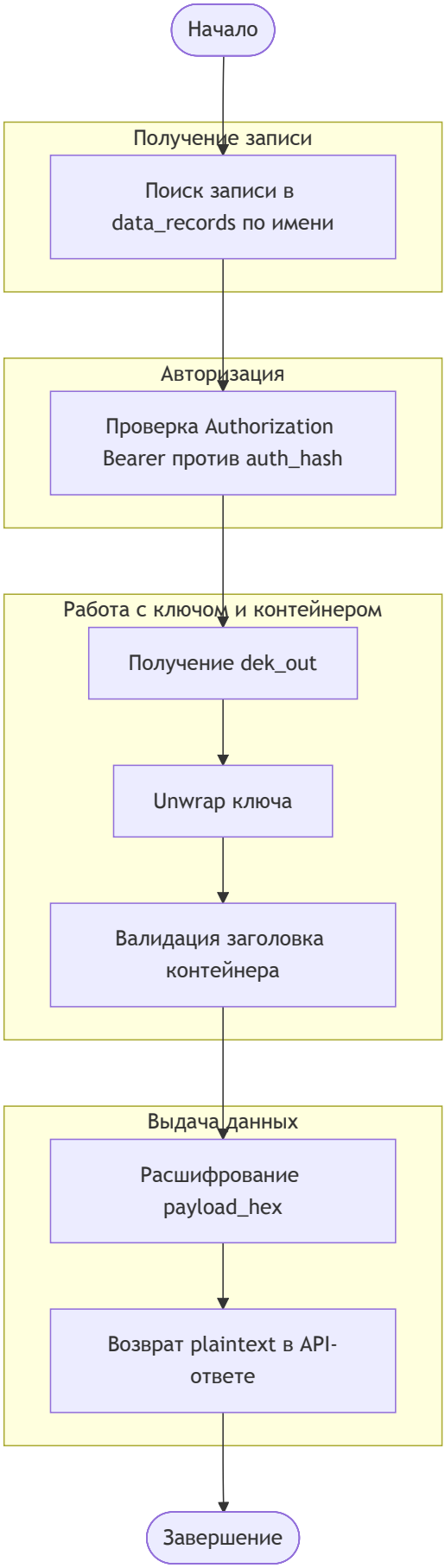

Рисунок 9 – Диаграмма сценария `GET/data/{name}`

4) Pydantic v2 + pydantic-settings – строгая валидация входных данных, параметров CLI/API и конфигурации окружения.

5) SQLAlchemy 2.x – доступ к БД и реализация слоя хранения (`keys`,`data_records`) через ORM.

6) pytest – модульное и интеграционное тестирование критичных сценариев.

Обоснование выбора языка программирования

Основным языком программирования для проекта выбран Python 3.13+, так как он обеспечивает высокий темп написания кода, имеет зрелую экосистему для backend и криптографии, а также удобен для тестирования и описания интерфейсов.

Python характеризуется как хорошо читаемый, гибкий, платформонезависимый язык программирования, доступный для быстрого освоения как базового, так и профессионального уровня программирования. Он сочетает в себе возможности встроенных высокоуровневых структур данных и объектно-ориентированность с кратким и понятным синтаксисом. Благодаря своим качествам он подходит для решения широкого спектра задач, в том числе для автоматизации, анализа данных, машинного обучения, а также для программной реализации научных исследований и вычислений. Это универсальный и гибкий инструмент как для новичков, так и для опытных профессиональных разработчиков. Единственным недостатком Python как современного языка программирования стоит отметить сравнительно невысокую скорость выполнения программ, написанных на нем, что связано с их интерпретируемостью [18].

Сравнение с другими языками программирования показало, что для данного проекта Python имеет хороший баланс «скорость реализации»-«читаемость кода» и наличие библиотек под задачу конвертного шифрования. При этом ограничения Python, такие как интерпретируемость, зависимость от качества runtime-конфигурации, меньшая производительность в CPU-bound сценариях компенсируются архитектурными мерами: разделением KEK/DEK, хранением KEK вне БД, применением проверенных внешних библиотек и покрытием критичных ветвей тестами.

На рисунке 10 представлено сравнение языков программирования по сложности написания и поддержки с учетом специфики данного проекта.

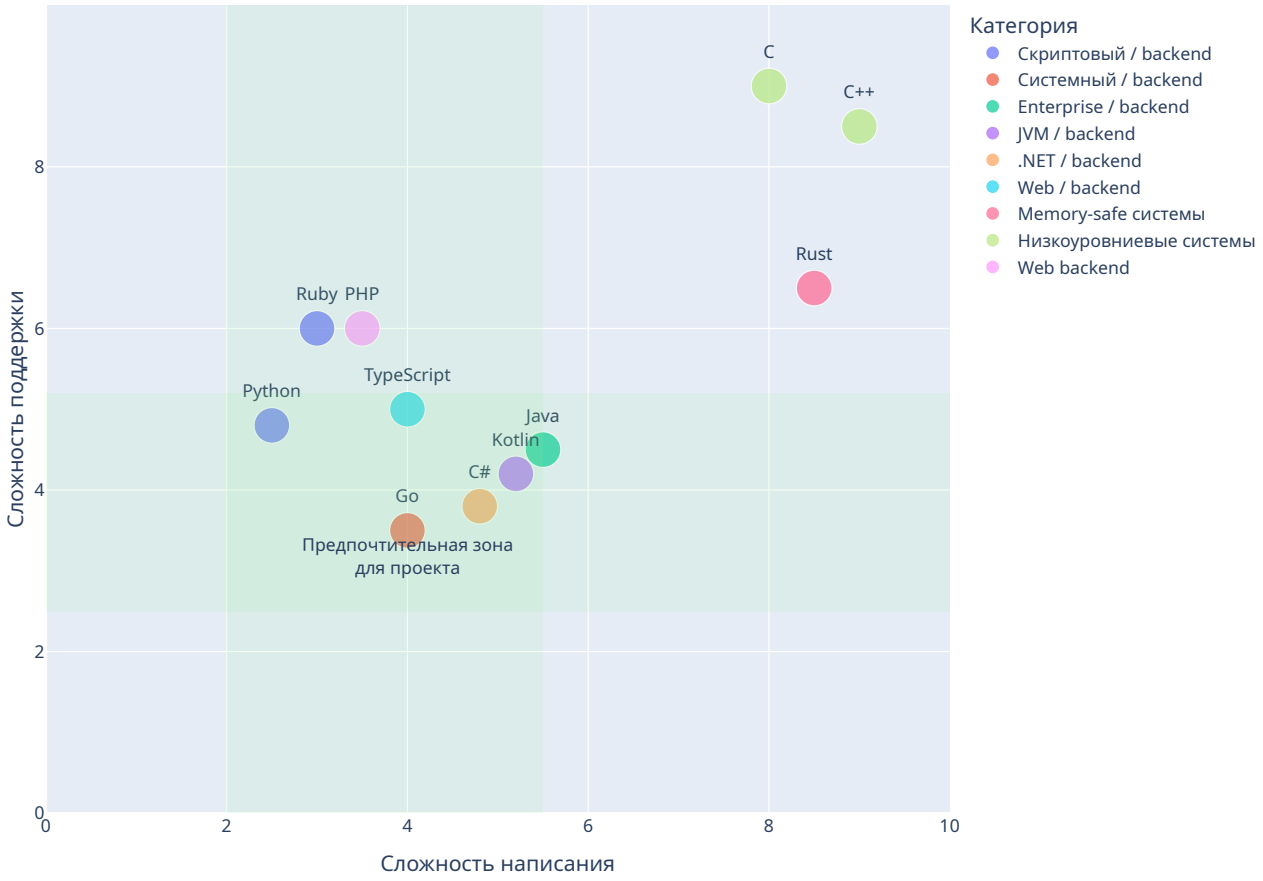

Рисунок 10 – Сравнение языков программирования по сложности написания и поддержки

Обоснование выбора технологий

Анализ выбранного технологического стека проводился с учётом требований к безопасности, производительности и удобству поддержки системы. Для каждой библиотеки были рассмотрены альтернативные решения, а также оценены их преимущества и ограничения в контексте данного проекта.

Результаты сравнительного анализа представлены в таблице 3.

Таблица 3 – Сравнительный анализ выбранных технологий и альтернатив

Технология	Причины выбора	Альтернативы и ограничения
1	2	3
uvicorn	Лёгкий и производительный ASGI-сервер; простота запуска и интеграции с FastAPI; минимальная операционная сложность	gunicorn – более сложная конфигурация и ориентирован на production-оркестрацию; hypercorn – менее распространён и имеет меньшую экосистему

Продолжение таблицы 3

1	2	3
cryptography	Production-уровень реализации криптографических примитивов; поддержка AEAD-алгоритмов (AES-GCM, ChaCha20-Poly1305); использование OpenSSL; высокая надёжность и широкое применение в индустрии	PyNaCl – проще, но ограниченный набор алгоритмов; низкоуровневый OpenSSL API – более гибкий, но существенно увеличивает риск ошибок реализации
FastAPI	Нативная поддержка асинхронности (ASGI); высокая производительность; автоматическая генерация OpenAPI; тесная интеграция с Pydantic и строгая типизация	Flask – менее строгая типизация, требуется ручная валидация; Django REST Framework – избыточен для лёгкого сервиса, выше порог сложности
Pydantic v2 + pydantic-settings	Строгая валидация данных на основе аннотаций типов; высокая производительность (v2); единая модель данных для CLI, API и конфигурации; снижение количества ошибок	marshmallow – более многословен и менее удобен; dataclasses – не предоставляют встроенной валидации и требуют ручной реализации
SQLAlchemy 2.x	Зрелый ORM с гибкой архитектурой; поддержка как ORM, так и Core; контроль транзакций; независимость от фреймворка	raw SQL – больше контроля, но выше сложность и риск ошибок; Django ORM – привязан к Django и избыточен для данного проекта
pytest	Лаконичный синтаксис тестов; мощная система фикстур; удобство модульного и интеграционного тестирования; де-факто стандарт в Python	unittest – более громоздкий синтаксис и меньшая гибкость при организации тестов

Вывод: выбранный технологический стек выбран правильно с учетом специфики проекта и соответствует всем требованиям.

2.3 Модель угроз и допущений

Угроза безопасности информации – это условия и факторы, создающие потенциальную или уже существующую опасность нарушения безопасности информации (ее конфиденциальности, целостности и доступности). Реализация угроз безопасности информации злоумышленником, как правило, возмож-

на в информационных системах, автоматизированных системах управления, информационно-телекоммуникационных сетях, облачных инфраструктурах.

Модель угроз (безопасности информации) – это физическое, математическое и описательное представление свойств или характеристик угроз безопасности информации

Модель угроз – это документ, назначение которого - определить угрозы, актуальные для конкретной системы [15].

Модель угроз для данного проекта построена для архитектуры «CLI/API + БД + криптографический модуль» с учетом используемой схемы конвертного шифрования. Анализ выполняется в предположении, что основной ущерб связан с нарушением конфиденциальности хранимых данных и компрометацией ключевого материала.

2.3.1 Ключевые активы системы

В рамках рассматриваемой системы выделяется набор критически важных активов, компрометация которых может привести к раскрытию конфиденциальных данных или нарушению целостности системы. Основной упор делается на защите криптографических ключей, данных и механизмов аутентификации.

1) `VAULT_MASTER_KEK_HEX` (master KEK), используемый для извлечения DEK.

2) Обернутый DEK в таблице `keys`.

3) Зашифрованные контейнеры и метаданные в `data_records`.

4) Заголовок авторизации (Authorization: Bearer) и его PBKDF2-представление.

5) Исполняемая среда сервиса (процесс API/CLI, файловая система, переменные окружения).

Перечисленные активы образуют основу модели защиты системы и определяют направления анализа потенциальных угроз.

2.3.2 Нарушители и их модель возможностей

Модель нарушителя строится с учётом различных уровней доступа к системе: от внешнего сетевого взаимодействия до компрометации инфраструктуры выполнения. Для каждого типа нарушителя предполагается различный набор возможностей и сценариев атак.

1) Внешний сетевой нарушитель: перехват/модификация трафика, попытки неавторизованного вызова API.

2) Нарушитель с доступом к БД: чтение дампа таблиц, попытки восстановления открытого текста по содержимому БД.

3) Нарушитель с компрометацией клиентского секрета: использование украденного Bearer-токена.

4) Нарушитель с доступом к хосту выполнения: чтение переменных окружения и runtime-памяти процесса.

Данная классификация позволяет системно рассмотреть угрозы, соответствующие различным уровням компрометации.

2.3.3 Границы доверия

Для корректного анализа угроз важно определить границы доверия, разделяющие компоненты системы по уровню контролируемости и потенциальной уязвимости.

1) Доверенная зона: процессы `vault/cli.py` и `vault/api.py`, выполняющие криптооперации и проверку авторизации.

2) Условно доверенная зона: хранилище БД (`vault.db`), где данные считаются потенциально компрометируемыми.

3) Недоверенная зона: сеть и внешние клиенты.

Такое разделение позволяет применять различные механизмы защиты в зависимости от уровня доверия к компоненту.

2.3.4 Принятые допущения модели

При построении модели угроз были приняты следующие допущения, ограничивающие область анализа и определяющие базовые условия безопасности системы:

1) `VAULT_MASTER_KEK_HEX` не хранится в БД и передается только через защищенное окружение выполнения.

2) Хост сервиса администрируется корректно, а доступ к переменным окружения ограничен.

3) TLS-сертификат и ключ сервера управляются безопасно и не скомпрометированы.

4) Используются только поддерживаемые AEAD-алгоритмы и корректные размеры nonce/tag.

Нарушение данных допущений существенно изменяет модель угроз и требует дополнительного анализа.

2.3.5 Основные сценарии угроз и контрмеры

На основе выделенных активов, модели нарушителя и границ доверия формируются ключевые сценарии угроз, а также соответствующие им меры защиты.

Компрометация БД

В данном сценарии предполагается получение злоумышленником доступа к содержимому базы данных.

Контрмеры: DEK хранится только в обернутом виде; KEK отсутствует в БД; данные в `data_records` сохраняются зашифрованными.

Перехват/подмена сетевого трафика

Сценарий предполагает возможность атаки типа Man-in-the-Middle и анализ сетевого взаимодействия.

Контрмеры: HTTPS при запуске API; обязательная проверка Authorization; AEAD-проверка тега целостности при расшифровании.

Компрометация Bearer-токена

Предполагается утечка или перехват токена доступа клиента.

Контрмеры: хранение не исходного токена, а PBKDF2-хеша; отказ в доступе при несовпадении; минимизация поверхности использования токена только в защищенном канале.

Компрометация runtime-окружения

Данный сценарий соответствует наиболее сильному нарушителю с доступом к системе выполнения.

Контрмеры: вынос KEK в переменную окружения (не в БД), организационные меры hardening хоста, ограничение прав процесса.

2.3.6 Диаграмма модели угроз

Финальная диаграмма модели угроз приложения показана на рисунке 11.

2.4 Схема взаимодействия с приложением

2.4.1 Общая схема

Схема взаимодействия отражает два ключевых эксплуатационных сценария системы: загрузка и перешифрование данных через CLI, выдача данных через защищенный API по авторизации.

В реализованном проекте взаимодействие построено вокруг принципа конвертного шифрования. Данные шифруются на уровне DEK (Data Encryption Key), а сами DEK не хранятся в открытом виде: перед записью в БД они

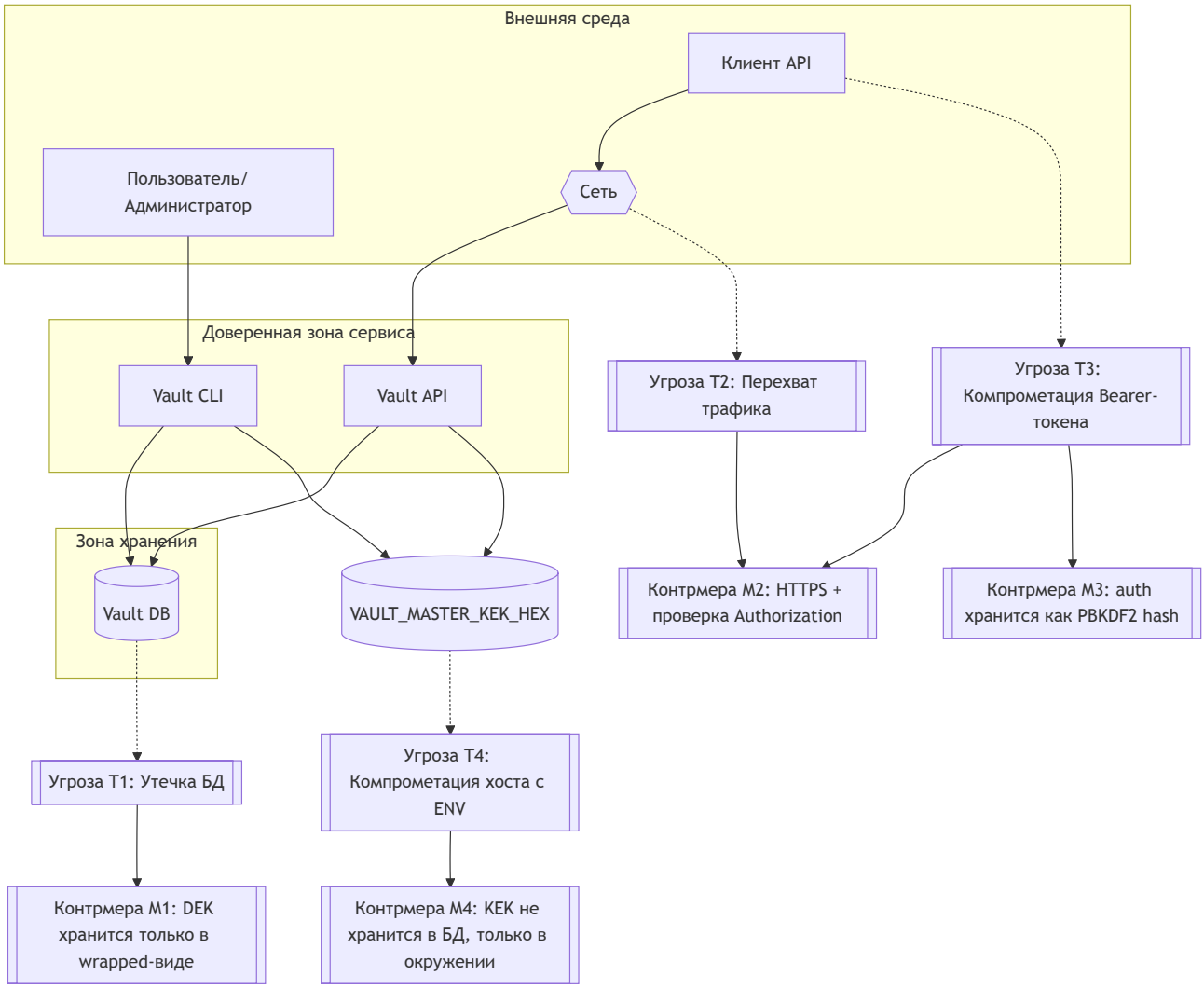

Рисунок 11 – Модель угроз приложения

оборачиваются master KEK (Key Encryption Key). Master KEK передается в процесс через переменную окружения `VAULT_MASTER_KEK_HEX` и не сохраняется в таблицах приложения. Такое разделение уменьшает последствия компрометации базы данных: злоумышленник может получить зашифрованные записи и обернутый DEK, но без доступа к окружению выполнения не сможет корректно расшифровать полезную нагрузку.

Участники процесса:

1) Администратор/разработчик, выполняющий команды управления и загрузки.

2) CLI, реализующий операции `load`, работу с контейнером и запись в БД.

3) Переменная окружения `VAULT_MASTER_KEK_HEX`, из которой берется

master KEK.

4) Vault DB, хранящая обернутый DEK и зашифрованные записи.

5) Vault API (HTTPS), предоставляющий endpoint чтения данных.

6) Клиент приложения, запрашивающий данные через `GET/data/{name}`.

Принципиальная особенность взаимодействия состоит в разделении ролей ключей: KEK никогда не сохраняется в БД, а DEK хранится только в обернутом виде. За счет этого даже при компрометации хранилища нарушитель не получает готовый ключ расшифрования без доступа к окружению выполнения.

На рисунках 12 и 13 представлены диаграммы загрузки данных через CLI и получения данных через API соответственно.

Интерпретация потока 1 (загрузка и перешифрование через CLI):

1) Администратор инициирует команду `load` с указанием входного и выходного алгоритмов, а также имен DEK.

2) Входной контейнер может быть передан двумя способами: как JSON-файл со структурой header + data либо как нагрузка в шестнадцатеричном представлении вместе с отдельным `--header-json`. В обоих вариантах данные проходят валидацию через Pydantic-модели. В менее безопасном варианте данные могут передаваться через команду `store-plaintext` в открытом виде.

3) CLI извлекает из БД обернутые ключи `dek-in` и `dek-out`, после чего считывает master KEK из окружения.

4) Модуль `key_manager`[5] выполняет извлечение обоих DEK. Если master KEK отсутствует, имеет неверную длину или не позволяет расшифровать обернутый DEK, операция прекращается без записи результата.

5) Модуль `crypto`[6] проверяет заголовок входного контейнера: наличие полей `alg`,`nonce`,`tag`, соответствие ожидаемому алгоритму, размер nonce 12 байт и размер tag 16 байт.

6) После успешной проверки выполняется расшифрование входного контейнера алгоритмом `alg-in`, затем повторное шифрование открытого текста алгоритмом `alg-out`.

7) В БД сохраняется новый контейнер: `header_json`,`payload_hex`, имена использованных DEK, входной и выходной алгоритмы, а также `auth_hash`. Открытый Bearer-токен не сохраняется.

[5]Рисунок 39, приложение А, с. 86

[6]Рисунок 36, приложение А, с. 78

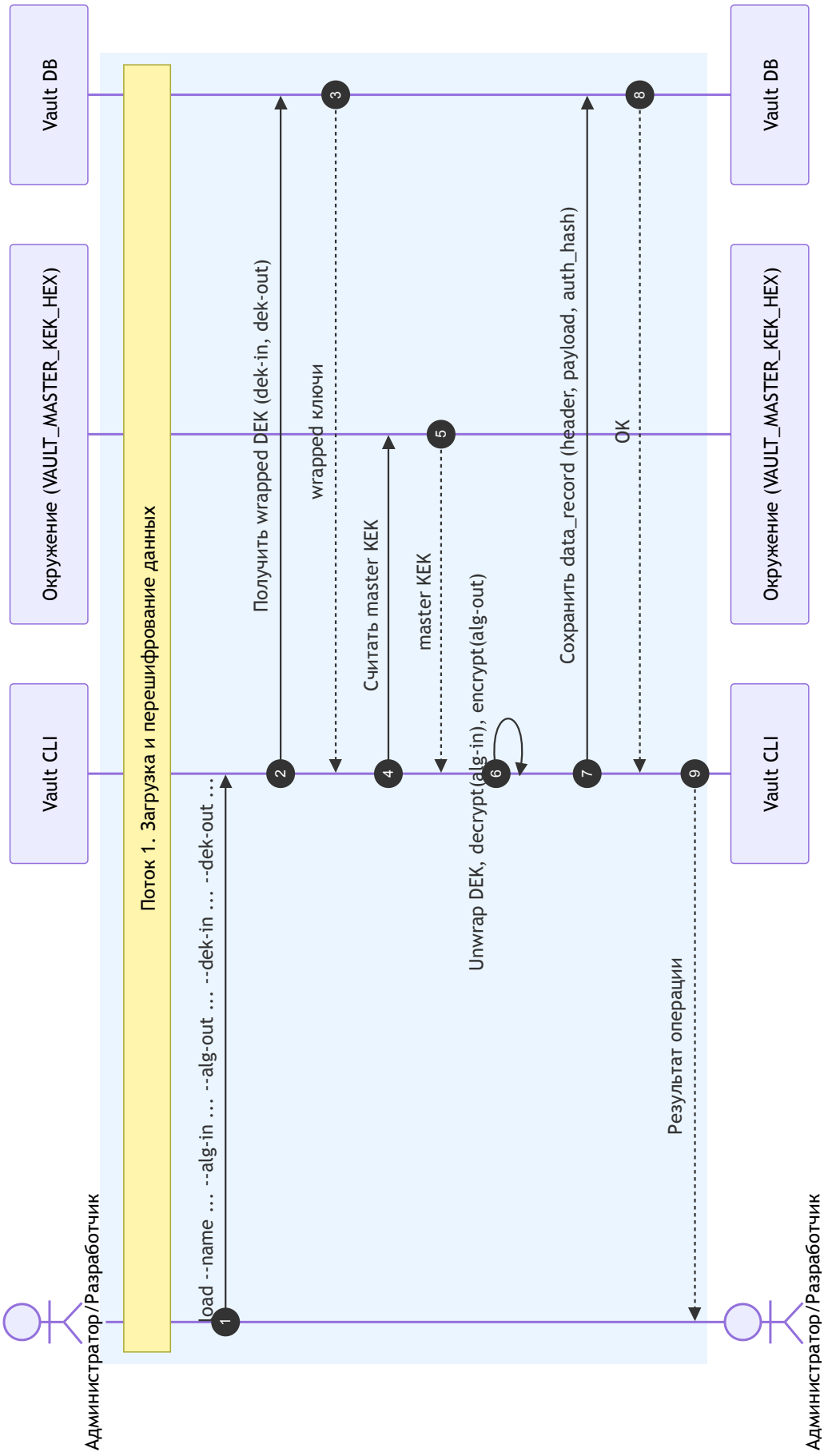

Рисунок 12 – Схема загрузки данных в приложение

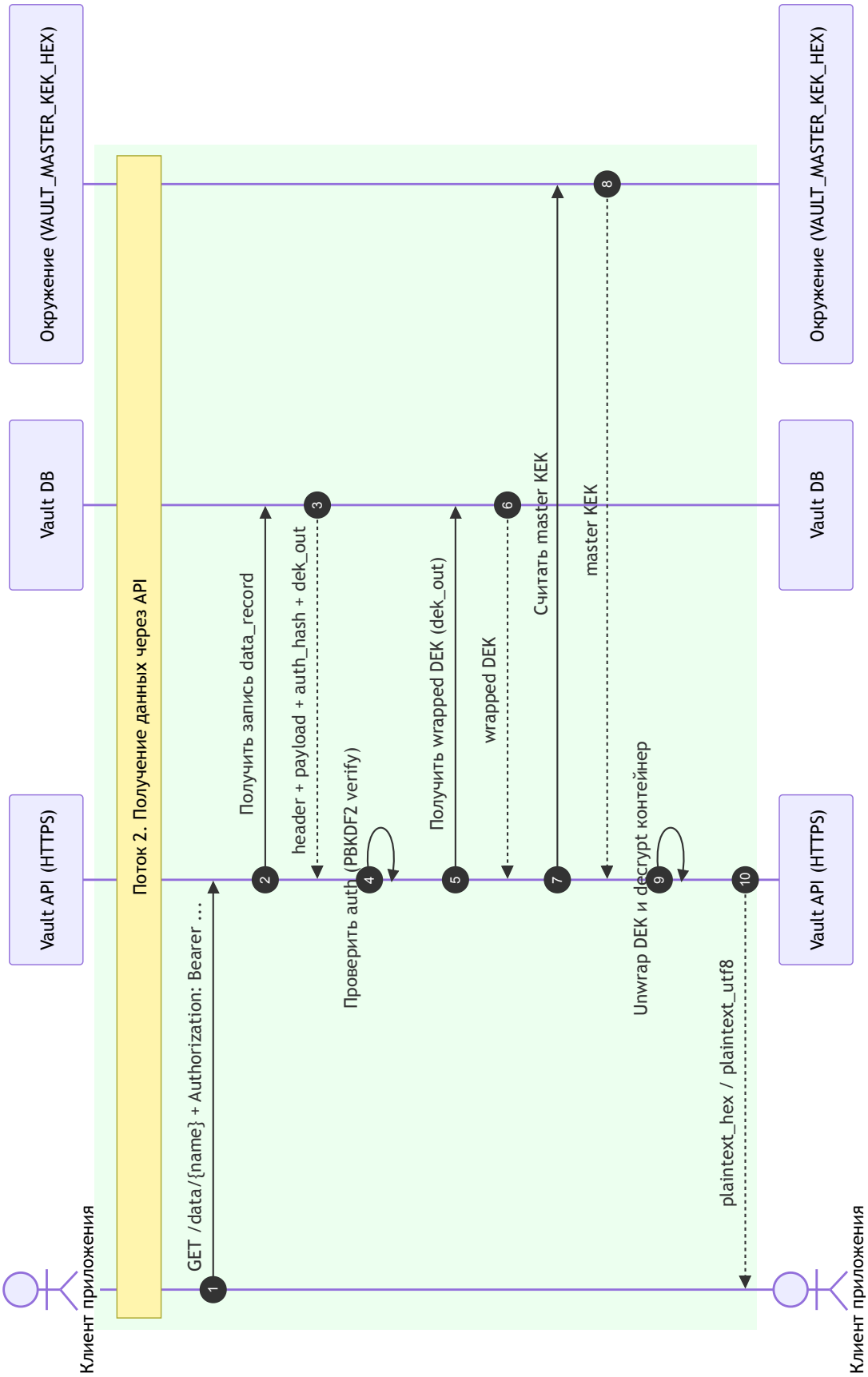

Рисунок 13 – Схема получения данных из приложения

8) Администратор получает структурированный JSON-ответ со статусом операции, выходным заголовком, размером открытого текста и сведениями о записи.

Интерпретация потока 2 (получение данных через API):

1) Клиент отправляет `GET/data/\{name\}` с заголовком Authorization: Bearer по HTTPS.

2) API ищет запись в `data_records` по логическому имени. Если запись отсутствует, возвращается 404 Not Found.

3) Если заголовок Authorization отсутствует или не проходит проверку через `verify_auth_header`, выполнение прерывается с 401 Unauthorized.

4) После успешной авторизации API проверяет наличие заголовка контейнера и валидирует его относительно `alg_out`, то есть относительно фактического алгоритма хранения данных после перешифрования.

5) API извлекает ключевую запись `dek_out` из таблицы `keys`. Если ключ отсутствует или master KEK не передан в процесс, сервер возвращает ошибку уровня 500, поскольку нарушена внутренняя целостность конфигурации.

6) Master KEK нормализуется и проверяется, затем `key_manager` выполняет извлечение целевого DEK.

7) Контейнер расшифровывается «на лету». Открытый текст не записывается обратно в БД и существует только в памяти процесса во время обработки запроса.

8) Клиенту возвращается JSON-ответ с именем записи, алгоритмом, `plaintext_hex`, опциональным `plaintext_utf8` и числом байт. Если данные не являются валидной UTF-8 строкой, поле `plaintext_utf8` возвращается как `null`, но шестнадцатеричное представление остается доступным.

Обработка ошибок также является частью схемы взаимодействия. Ошибки пользовательского ввода на стороне CLI приводят к прекращению команды через `SystemExit` с диагностическим сообщением: например, при отсутствии файла, некорректном JSON, неверном шестнадцатеричном представлении или неизвестном DEK. Ошибки API преобразуются в HTTP-статусы: отсутствие записи соответствует 404, неверный Bearer – 401, а повреждение внутренней конфигурации или контейнера – 500. Такое разделение позволяет отличать ошибки клиента от нарушений целостности серверной конфигурации.

Таким образом, схема взаимодействия демонстрирует сквозной защищен-

ный конвейер обработки данных: от административной загрузки и перешифрования до контролируемой выдачи клиенту по API. Эта схема служит опорной для последующих разделов, где формализуются модель угроз, математическая модель конвертного шифрования и критерии тестирования.

2.4.2 Схема работы веб-сервера

Веб-сервер используется для сетевой выдачи данных через HTTPS API. В отличие от CLI-команд, которые выполняются как разовые административные операции, API-сервер должен запускаться как отдельный процесс и сохранять состояние своего запуска: адрес, порт, URL базы данных и идентификатор процесса. Поэтому для эксплуатации важны два симметричных сценария: безопасный старт сервера и корректная остановка ранее запущенного процесса.

В проекте эти сценарии реализованы командами `web-serverup` и `web-serverdown` в модуле `vault.cli`[7]. Команда запуска формирует процесс `uvicorn`, который поднимает приложение `vault.api:app` с TLS-ключом и сертификатом. Команда остановки использует служебный PID-файл `.vault_web_server.json`, чтобы найти ранее запущенный процесс и завершить его средствами операционной системы. Это решение не требует отдельного systemd-unit, Docker Compose или внешнего process manager, но при этом демонстрирует полный жизненный цикл API-сервиса.

Для иллюстрации этих сценариев подготовлены две диаграммы последовательностей

Первая диаграмма описывает процесс запуска сервера. Важно отметить, что запуск API является не просто вызовом uvicorn, а последовательностью проверок, которые защищают систему от конфликтов и некорректного состояния. CLI сначала валидирует параметры команды через `WebServerUpCommand`: проверяет существование TLS-ключа и сертификата, корректность host/port и допустимый диапазон порта. Затем выполняется проверка PID-файла `.vault_web_server.json`. Если файл существует и указанный в нем процесс еще активен, команда не запускает второй экземпляр сервера, а возвращает ошибку `alreadyrunning`. Если PID-файл отсутствует, поврежден или указывает на уже завершенный процесс, он рассматривается как устаревший и удаляется.

После проверки состояния CLI формирует команду запуска:

[7] Рисунок 35, приложение А, с. 68

```
python -m uvicorn vault.api:app --host <host> --port <port> --ssl-keyfile <key> --ssl-certfile <
    cert>
```

Рисунок 14 – Строка запуска веб-сервера

При запуске в окружение дочернего процесса передается VAULT_DB_URL, чтобы API использовал ту же БД, что и CLI-команды. Далее CLI ожидает короткий интервал и проверяет, не завершился ли процесс сразу после старта. Если процесс не смог запуститься, пользователь получает ошибку Failedtostartwebserverprocess. Если запуск успешен, в .vault_web_server.json записываются pid,host,port и db_url, а CLI возвращает JSON-ответ со статусом ok и действием web-server-up.

Вторая диаграмма описывает обратный сценарий – остановку сервера. Ее следует рассматривать как эксплуатационную пару к схеме запуска: если web-serverup создает фоновый процесс и PID-файл, то web-serverdown должен корректно обработать все возможные состояния этого файла. Сначала CLI проверяет наличие .vault_web_server.json. Если файла нет, команда не считает это ошибкой и возвращает результат stopped=false с причиной not_running. Это удобно для повторного вызова команды остановки: она остается идемпотентной и не вызывает ошибок при создании сценариев автоматизации.

Если PID-файл существует, CLI пытается извлечь из него идентификатор процесса. При поврежденном JSON или некорректном PID файл удаляется, а пользователю возвращается причина invalid_pid_file. Если PID валиден, CLI передает управление операционной системе: в Windows используется taskkill/PID<pid>/T/F, а в Linux/macOS применяется сигнал завершения через os.kill(pid,15). После этого CLI делает короткую паузу, проверяет, остался ли процесс активным, удаляет PID-файл и возвращает JSON-ответ web-server-down с признаком stopped.

Диаграммы 1 и 2 представлены на рисунках 15 и 16 соответственно.

2.4.3 Схема управления ключами

Для фиксации механизма управления ключами в приложении ниже приведены три отдельные диаграммы последовательностей. Разбиение делает явными разные фазы жизненного цикла ключевого материала: подготовку master KEK, перевод DEK в оберутое представление и последующее использование

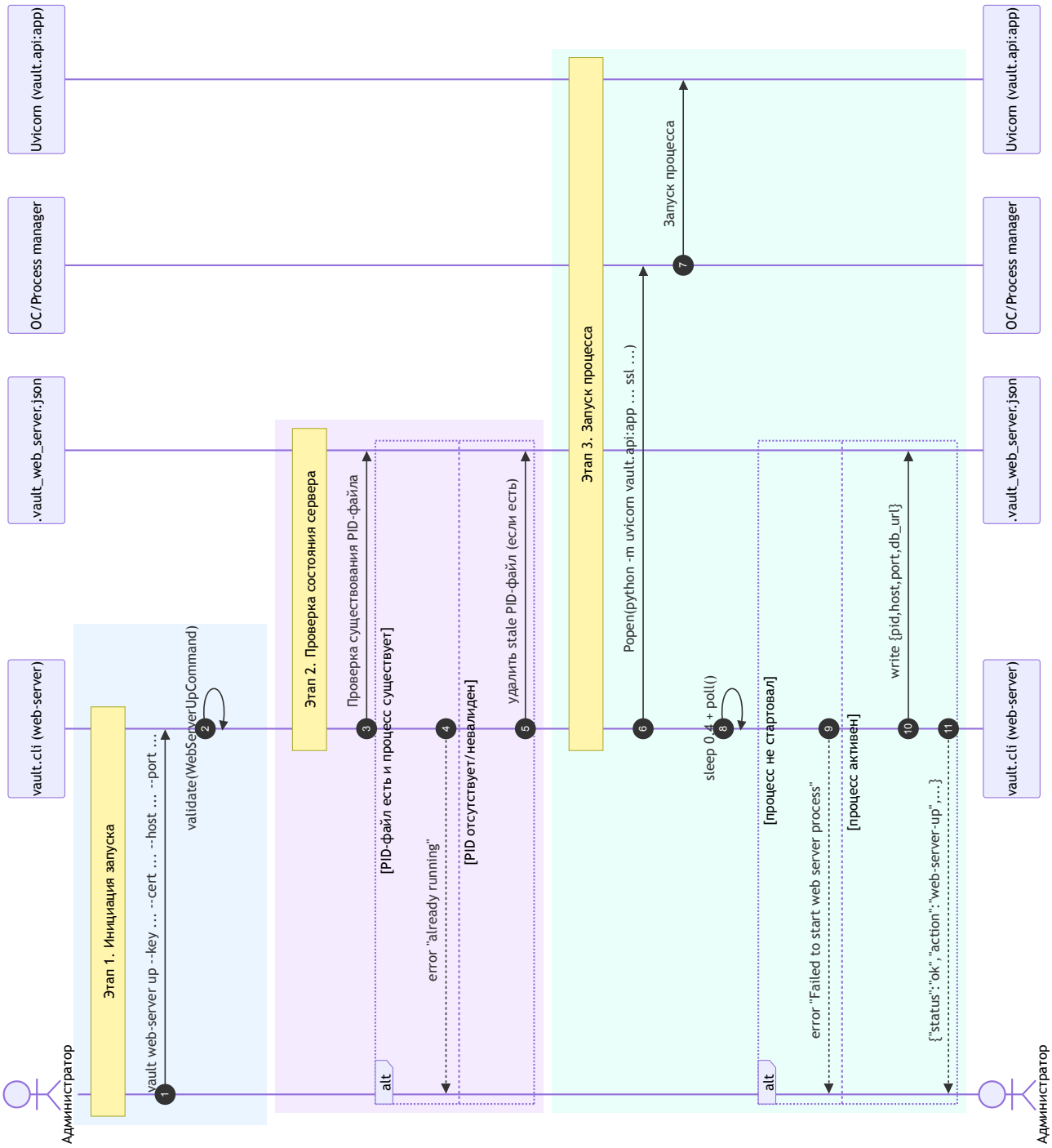

Рисунок 15 – Диаграмма запуска веб-сервера

DEK в runtime.

Подготовка master KEK

Первая диаграмма описывает подготовку master KEK. В обоих вариантах, генерации и импорте, CLI либо создает значение ключа, либо нормализует уже существующий ключ в шестнадцатеричном формате, после чего возвращает его оператору без записи в таблицу `keys`. Завершающим шагом является перенос

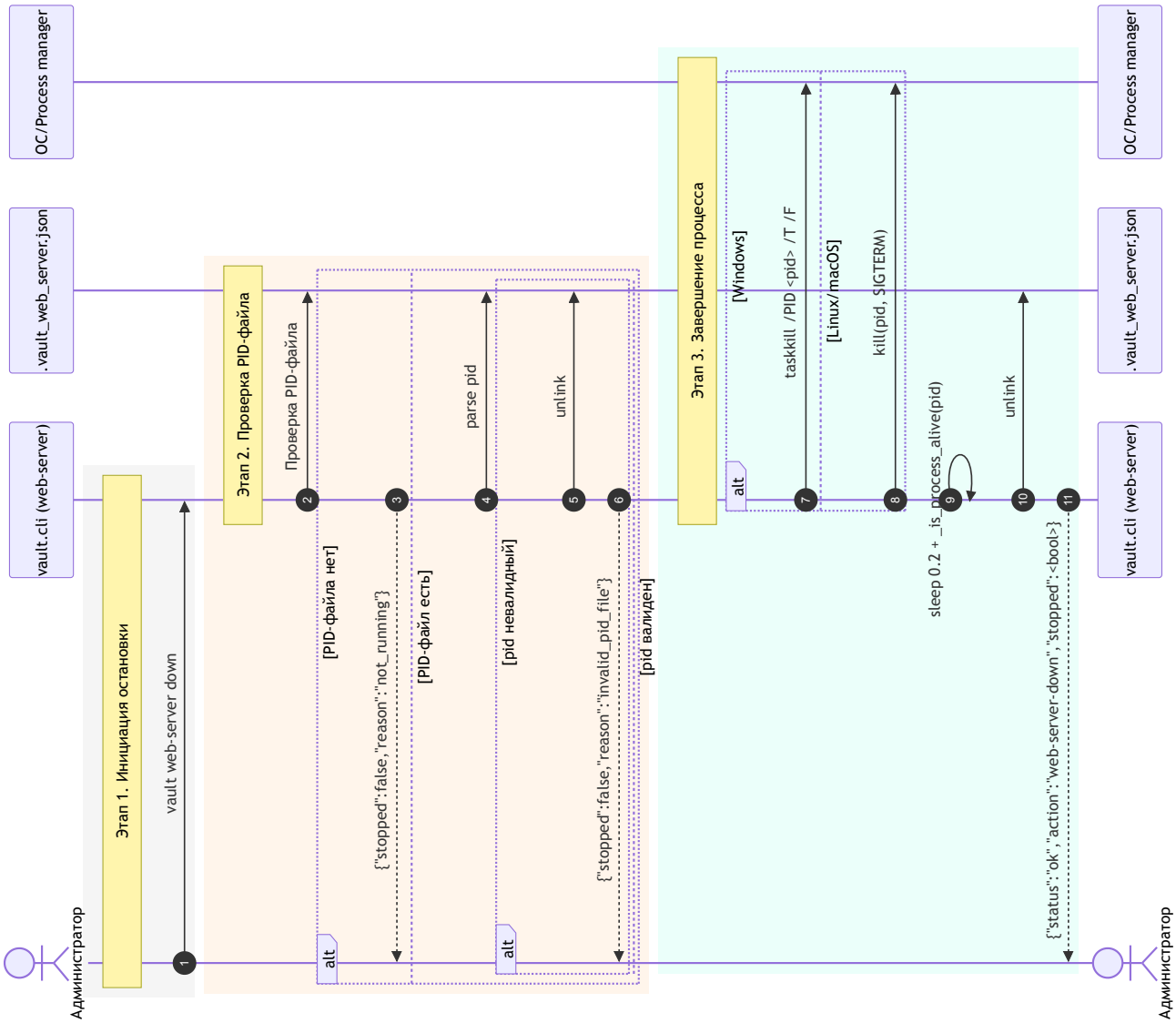

Рисунок 16 – Диаграмма остановки веб-сервера

значения в `VAULT_MASTER_KEK_HEX`, то есть в контур runtime-конфигурации, а не в контур долговременного хранения.

Диаграмма представлена на рисунке 17.

Создание или импорт DEK и его сохранение в обернутом виде

Вторая диаграмма показывает цикл постановки DEK на хранение. Независимо от того, был ли ключ данных сгенерирован внутри приложения или импортирован из файла, перед сохранением он обязательно связывается с master KEK через операцию `wrap_dek_hex`. В результате в БД попадает не открытый DEK, а сериализованный обернутый пакет с `nonce`, шифртекстом и тегом аутентификации, что снижает последствия компрометации таблицы `keys`.

Диаграмма представлена на рисунке 18.

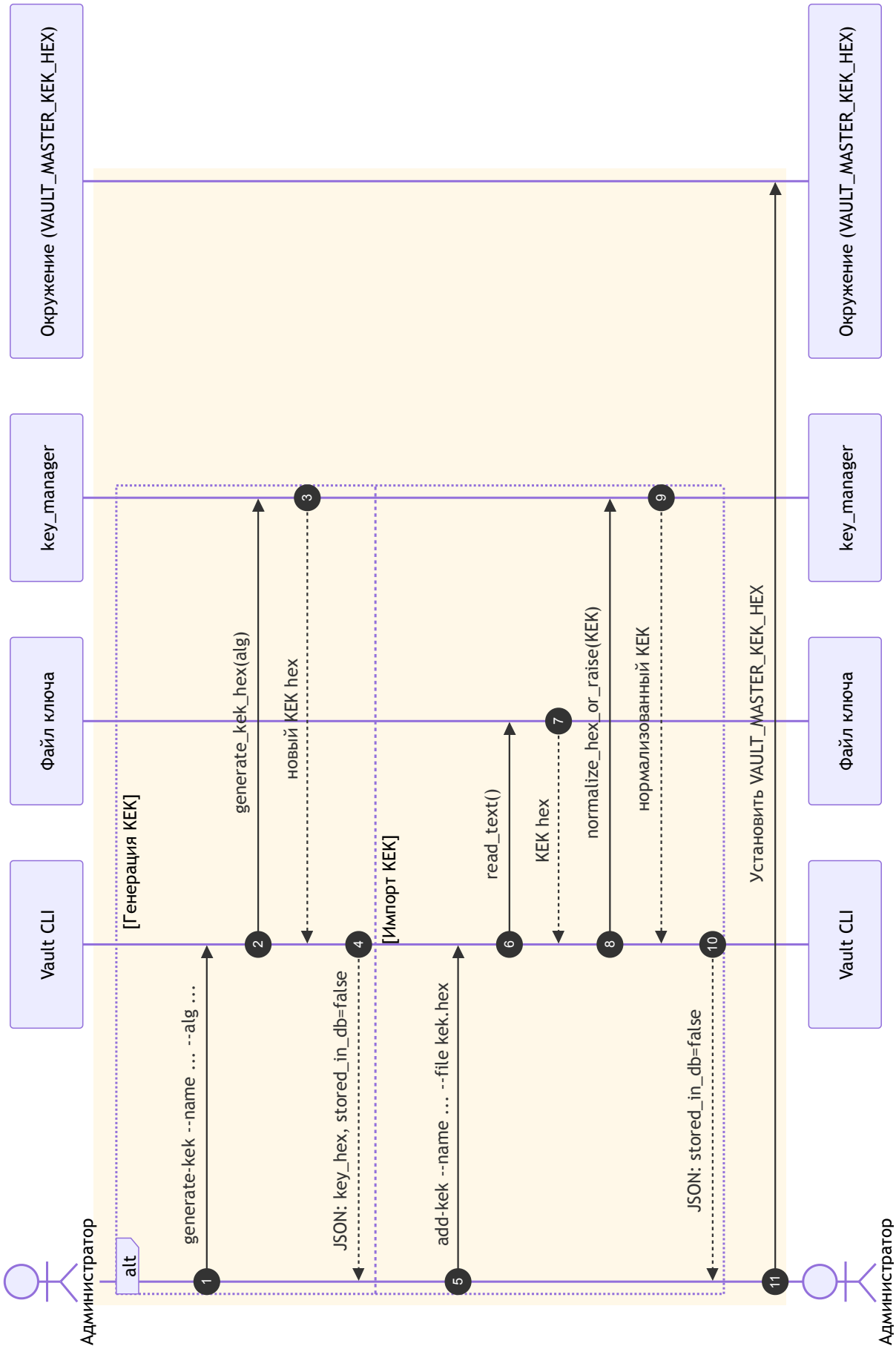

Рисунок 17 – Подготовка master KEK

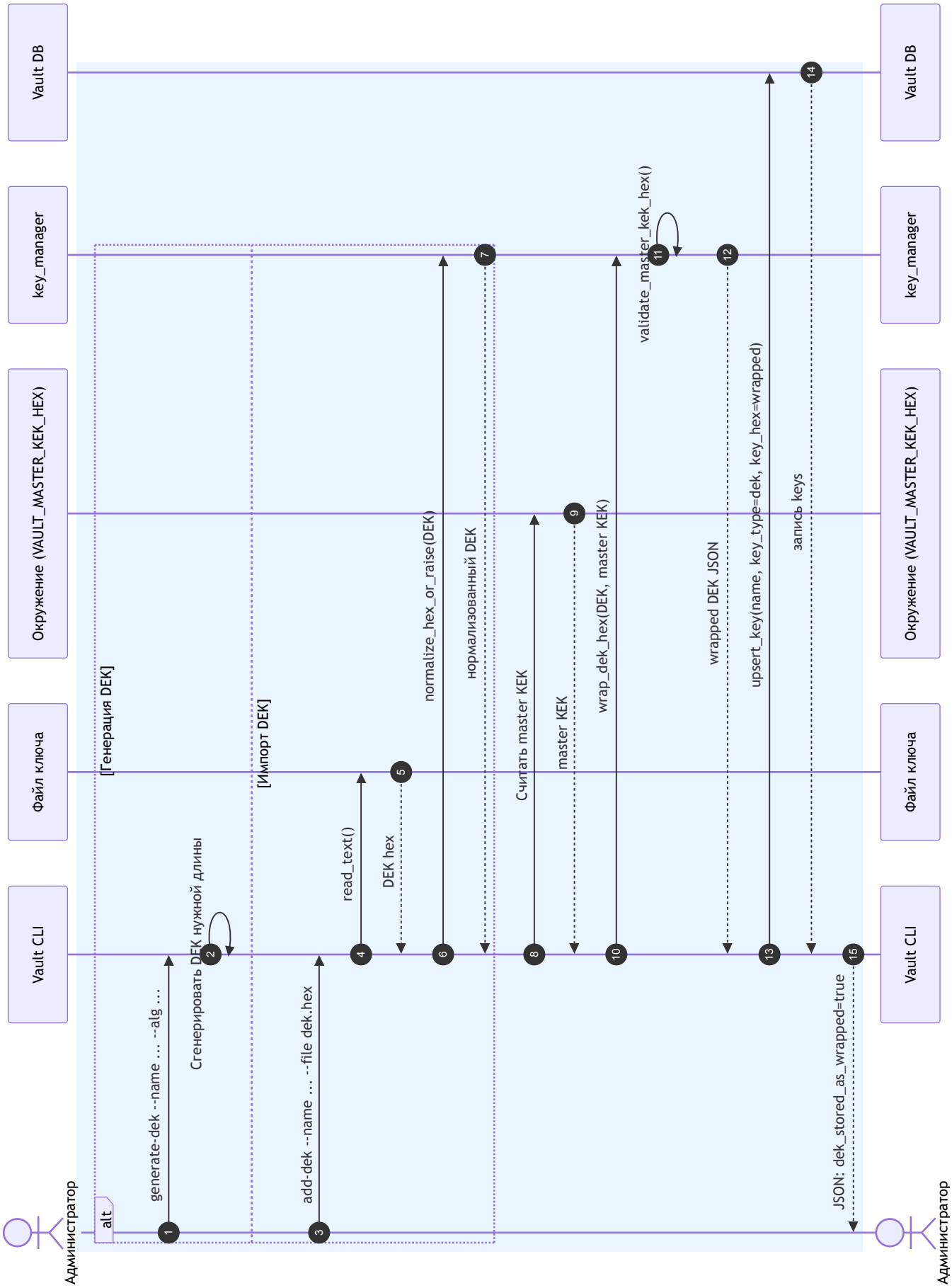

Рисунок 18 – Цикл постановки DEK на хранение

Использование DEK в runtime

Третья диаграмма отражает эксплуатационную фазу работы с ключом данных. Runtime-компонент CLI или API извлекает из БД обернутое представление DEK, считывает master KEK из окружения и выполняет `unwrap_dek_hex` через `key_manager`. Открытый DEK появляется только в оперативной памяти процесса и используется немедленно для криптографической операции, после чего не сохраняется обратно в хранилище.

Диаграмма представлена на рисунке 19.

2.5 Обоснование выбора методики разработки

Чтобы успешно справляться с постоянно меняющимися потребностями и вызовами, с которыми сталкивается индустрия информационных технологий, разработчики и управляющие проектами активно развивают и применяют разнообразные методологии. История этих методологий богата и увлекательна, и она играет критическую роль в формировании современных подходов к разработке программного обеспечения и управлению IT-проектами [7].

Методология TDD (Test-Driven-Development) – это подход к разработке ПО, который предполагает написание тестов перед написанием программного кода. Этот подход позволяет программистам убедиться, что их код работает корректно.

Основная идея TDD заключается в том, чтобы начинать с написания тестов для функциональности, которую вы планируете реализовать, а затем создавать код, который будет проходить эти тесты. Таким образом, тесты становятся не только инструментом для проверки работоспособности программного кода, но и своего рода спецификацией, определяющей требования к функционалу [20].

Процесс TDD обычно следует циклу, известному как «красный, зелёный, рефакторинг»:

1) Красный (Red) – Написание тестов, которые описывают ожидаемое поведение новой функциональности или исправление ошибок в существующем коде. На этом этапе тесты должны не проходить, что создаст «красный» сигнал [20].

2) Зелёный (Green) – Написание минимального кода, необходимого для прохождения написанных тестов. Цель – сделать тесты проходящими и получить «зелёный» сигнал [20].

3) Рефакторинг (Refactor) – Улучшение кода без изменения его функцио-

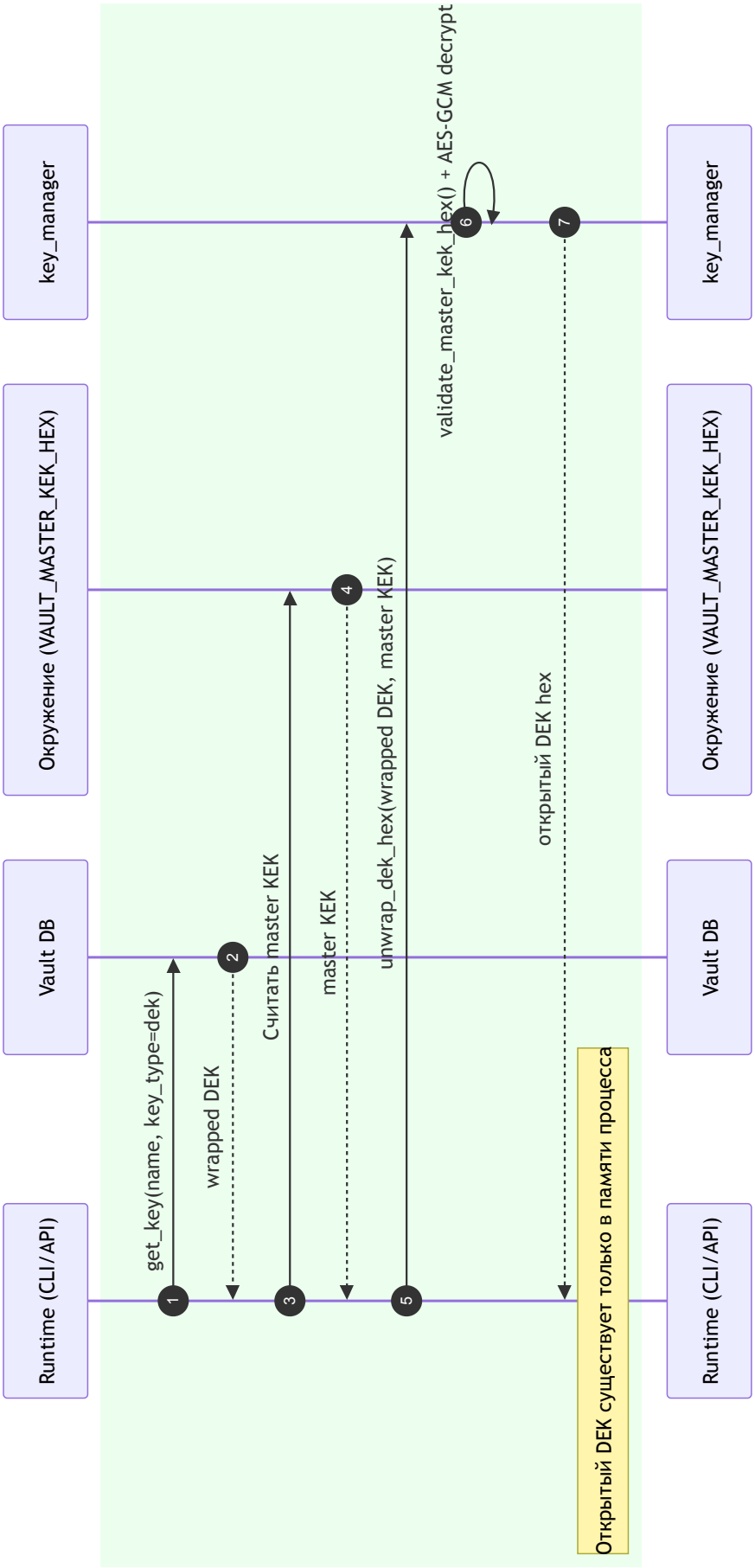

Рисунок 19 – Эксплуатационная фаза работы с ключом данных

нальности, чтобы сделать его более чистым, эффективным и поддерживаемым [20].

Для реализации проекта была выбрана методика, близкая к TDD (Test-Driven Development). Для данного проекта такой подход особенно уместен, поскольку основные требования можно выразить как набор воспроизводимых инвариантов:

1) DEK не должен сохраняться в БД в открытом виде.

2) Master KEK должен поступать из окружения и проходить проверку размера.

3) Контейнер данных должен содержать корректные `alg`,`nonce`,`tag` и `ciphertext`.

4) Расшифрование должно завершаться ошибкой при неверном ключе, алгоритме, теге или формате данных.

5) API не должен возвращать открытый текст без корректного Authorization: Bearer.

6) Хранимый Bearer-секрет должен быть представлен PBKDF2-хешем, а не исходной строкой.

7) CLI-команды должны завершаться предсказуемо при ошибках входных параметров.

На практике TDD в проекте использовался следующим образом: критичные сценарии сначала формулировались через тестируемое поведение, после чего реализовывались модули `crypto`,`key_manager`,`auth`,`db`,`cliapi`. Такой подход позволил разделить разработку на короткие проверяемые итерации. Например, для криптографического слоя проверяется корректность шифрования и расшифрования контейнера, для авторизации – хеширование и проверка Bearer-заголовка, для API – успешная выдача данных и отказ при неверном токене.

Альтернативой мог быть классический подход test-after development, при котором тесты пишутся после завершения реализации. Он проще на старте, однако хуже подходит для задач, где цена ошибки высока и требуется постоянно контролировать регрессии. Waterfall-подход также не является оптимальным: он предполагает жесткую последовательность этапов и хуже реагирует на уточнение контрактов данных, которое неизбежно возникает при разработке криптографического контейнера, CLI-команд и API. Итеративная Agile-модель

близка по духу к выбранному подходу, но сама по себе не гарантирует тестируемости. Code-and-Fix обеспечивает максимальную скорость первого прототипа, однако для системы безопасности создает неприемлемый риск накопления скрытых дефектов.

2.6 Реализация ключевых модулей

Процесс разработки системы был организован вокруг постепенного наращивания функциональности: от минимального криптографического ядра к полноценному контуру CLI → БД → API. Такой порядок был выбран осознанно. В системе, работающей с ключами и зашифрованными данными, нельзя начинать с внешнего интерфейса и затем «подключать безопасность» как отдельный слой. Напротив, сначала необходимо зафиксировать корректное поведение базовых операций: проверку ключей, формат контейнера, шифрование, расшифрование, оборачивание DEK и отказ при некорректных данных.

С учетом выбранной TDD-методики разработка велась короткими итерациями. Каждая итерация включала три шага: формулирование ожидаемого поведения, реализацию минимального кода для прохождения проверки и последующее уточнение структуры модуля. Это позволило не только быстрее обнаруживать ошибки, но и документировать саму логику системы через набор проверяемых сценариев. Например, требования к размеру nonce, формату Authorization, невозможности выдачи данных без Bearer и сохранению DEK только в обернутом виде были выражены не только в тексте, но и в тестируемом поведении.

Разработка выполнялась в следующей последовательности:

1) Формирование криптографического ядра `vault/crypto.py`.

2) Реализация управления ключами и конвертного шифрования в `vault/key_manager.py`.

3) Описание контрактов входных данных через Pydantic-схемы в `vault/schemas.py`[8] и `vault/db_schemas.py`[9].

4) Реализация слоя хранения через SQLAlchemy в `vault/db.py`[10].

5) Реализация авторизации и безопасного хранения Bearer-секрета в `vault/auth.py`[11].

[8]Рисунок 40, приложение А, с. 88

[9]Рисунок 38, приложение А, с. 84

[10]Рисунок 37, приложение А, с. 80

[11]Рисунок 34, приложение А, с. 67

6) Разработка CLI-команд в `vault/cli.py`.

7) Разработка API-слоя в `vault/api.py`[12].

8) Добавление интеграционных сценариев и сквозной проверки через pytest и `scripts/check_all.py`[13].

Первым этапом стало создание криптографического модуля `vault/crypto.py`. Его задача состоит в том, чтобы изолировать непосредственные операции шифрования и расшифрования от остальной бизнес-логики. В модуле были зафиксированы поддерживаемые размеры ключей: 16 байт для AES-128-GCM, 32 байта для AES-256-GCM и 32 байта для ChaCha20-Poly1305. Это решение позволило не передавать ответственность за проверку длины ключа в CLI или API, а централизовать ее в одной функции `key_bytes_for_algorithm`.

Для контейнера данных был выбран компактный формат: отдельно хранится шифртекст в шестнадцатеричном виде и заголовок `header`, содержащий `alg`,`nonce` и `tag`. Такой формат удобен для записи в БД и для передачи между CLI/API. При этом заголовок не считается доверенным: перед расшифрованием он проходит проверку через `validate_header`[14]. Проверяются наличие обязательных полей, корректность шестнадцатеричных значений, совпадение алгоритма с ожидаемым, размер nonce 12 байт и размер AEAD-тега 16 байт. Если любое из условий нарушено, операция завершается ошибкой до попытки вернуть открытый текст.

Следующим этапом была реализована работа с ключевым материалом в `vault/key_manager.py`. Здесь было закреплено разделение ролей KEK и DEK. DEK используется для шифрования пользовательских данных, а master KEK используется для оборачивания DEK перед сохранением в БД. Функция `validate_master_kek_hex`[15] проверяет, что master KEK задан корректной шестнадцатеричной строкой и имеет размер 32 байта. Функция `wrap_dek_hex`[16] формирует JSON-структуру обернутого DEK с полями `v`,`wrap_alg`,`nonce`,`wrapped` и `tag`; для оборачивания используется AES-256-GCM. Обратная операция `unwrap_dek_hex` восстанавливает исходный DEK только при наличии корректного master KEK.

После криптографического слоя были описаны контракты данных.

[12]Рисунок 33, приложение А, с. 65

[13]Рисунок 50, приложение А, с. 101

[14]Метка 4, рисунок 36, приложение А, с. 79

[15]Метка 8, рисунок 39, приложение А, с. 87

[16]Метка 9, рисунок 39, приложение А, с. 87

В vault/schemas.py описаны модели команд и входных контейнеров: GenerateKeyCommand,AddKeyCommand,LoadCommand,WebServerUpCommand,HeaderMod и FlatJsonContainerModel. Это позволило перенести большую часть проверок с уровня ручных if-условий в декларативные Pydantic-модели. Например, LoadCommand проверяет существование входного файла, формат Bearer-заголовка, допустимость алгоритмов и зависимость между --data-hex и --header-json. Если данные передаются напрямую через --data-hex, заголовок становится обязательным, иначе система не сможет проверить алгоритм, nonce и tag.

Для слоя хранения были созданы отдельные схемы vault/db_schemas.py. Они валидируют данные перед записью в репозиторий и перед выдачей наружу. Такое разделение важно: схемы CLI описывают пользовательский ввод, а схемы БД описывают внутренние контракты хранения. Например, DataRecordUpsertIn проверяет auth_header, алгоритмы, имена DEK, JSON-заголовок контейнера и шестнадцатеричное представление данных.

Слой хранения реализован в vault/db.py через SQLAlchemy. В нем определены две ORM-модели: KeyRecord и DataRecord. Таблица keys хранит логическое имя ключа, тип ключа, алгоритм и значение key_hex. Для DEK это значение в целевом режиме содержит обернутый JSON. Таблица data_records хранит имя записи, auth_hash, входной и выходной алгоритмы, ссылки на dek_in и dek_out, header_json и payload_hex. В репозитории VaultRepository реализованы операции upsert_key, get_key, upsert_data_record и get_data_record_by_name[17].

Модуль авторизации vault/auth.py был выделен отдельно, чтобы не смешивать проверку Bearer с API-кодом или репозиторием. Функция normalize_bearer_auth_header[18] задает минимальный контракт заголовка: строка должна начинаться с Bearer и содержать непустой токен. Функция hash_auth_header сохраняет нормализованное значение в виде PBKDF2-хеша с алгоритмом sha256, солью 16 байт и 200000 итераций. Функция verify_auth_header[19] проверяет кандидатный заголовок через hmac.compare_digest, что снижает риск утечки информации через различия во времени сравнения.

[17]Метка 7, рисунок 37, приложение А, с. 82

[18]Метка 2, рисунок 34, приложение А, с. 67

[19]Метка 3, рисунок 34, приложение А, с. 68

После готовности сервисных модулей был реализован CLI-слой `vault/cli.py`. CLI стал основной административной точкой входа и получил команды `generate-dek`, `generate-kek`, `add-dek`, `add-kek`, `load`, `web-serverup` и `web-serverdown`. Для каждой команды используется общий принцип: сначала аргументы разбираются через `argparse`, затем валидируются через Pydantic-модель, после чего вызываются сервисные функции и репозиторий. Результаты команд возвращаются в JSON-формате, что удобно для демонстрации, автоматизации и интеграционных проверок.

Наиболее важной CLI-командой является `load`, поскольку она объединяет почти все компоненты системы. Команда получает входной контейнер, проверяет формат, извлекает `dek_in` и `dek_out` из БД, считывает master KEK из `VAULT_MASTER_KEK_HEX`, выполняет извлечение обоих DEK, расшифровывает входной `payload` и повторно шифрует его целевым алгоритмом. После этого в `data_records` сохраняется уже новый контейнер, а Bearer-заголовок передается в репозиторий только для хеширования и сохранения в поле `auth_hash`.

Завершающим прикладным слоем стал модуль `vault/api.py`. В нем реализована фабрика `create_app`[20], которая создает FastAPI-приложение, и endpoint `GET/data/{name}`. API выполняет последовательность проверок в безопасном порядке: сначала ищет запись, затем проверяет `Authorization`, затем валидирует заголовок контейнера, после этого получает `dek_out`, выполняет извлечение через master KEK и только затем расшифровывает `payload`. Такой порядок принципиален: извлечение ключа и криптографическая операция не выполняются до успешной авторизации клиента. Процесс разработки производился как последовательное расширение ядра приложения. В начале были реализованы функции, отвечающие за криптографическую обработку данных. Затем появились контракты данных и слой хранения. После этого были добавлены команды CLI и сетевой API. Такой порядок позволил сохранить контролируемую сложность проекта: каждый новый слой опирался на уже проверенные нижележащие модули, а не дублировал их логику.

[20]Метка 1, рисунок 33, приложение А, с. 66

3 ДЕМОНСТРАЦИЯ РЕЗУЛЬТАТА РАБОТЫ

Результат выполненной работы целесообразно показывать не только как перечень реализованных модулей, но и как воспроизводимый эксплуатационный сценарий.

Генерация master KEK

Эта команда демонстрирует подготовку мастер-ключа, который используется не для шифрования пользовательских данных напрямую, а для оборачивания ключей данных. Важно подчеркнуть, что результат команды возвращается оператору в JSON-формате и не записывается в таблицу `keys`. Тем самым подтверждается один из ключевых инвариантов проекта: master KEK существует вне БД и должен передаваться в процесс только через защищенный канал конфигурации. Команда представлена на рисунке 20.

```
PS> python main.py --db-url sqlite:///vault.db generate-kek --name master --alg AES-256-GCM
{
  "status": "ok",
  "action": "generate-kek",
  "algorithm": "AES-256-GCM",
  "key_hex": "592395cfaaaf4b42ec40f06336fe173e72b6a8137ec1abcc9b28b78467f80ee8",
  "stored_in_db": false,
  "note": "Store this KEK outside DB and set VAULT_MASTER_KEK_HEX for runtime operations."
}
```

Рисунок 20 – Создание KEK

Импорт ключей данных

Эти команды показывают постановку DEK на хранение. На вход CLI получает ключи в шестнадцатеричном формате из файлов, однако в БД попадают не исходные значения, а результат `wrap_dek_hex`, сформированный с использованием master KEK. Важно подчеркнуть, что логические имена «`src`» (рисунок 21) и «`dst`» (рисунок 22) позволяют явно разделить входной и выходной контур шифрования при последующем перешифровании контейнера.

Загрузка и перешифрование контейнера

Данная команда является ключевой демонстрацией результата работы системы, поскольку объединяет все основные подсистемы проекта. CLI считывает входной контейнер из `container.json`, проверяет структуру заголовка, извлекает из БД `src` и `dst` в обернутом виде, выполняет их извлечение через `VAULT_MASTER_KEK_HEX`, расшифровывает данные исходным алгоритмом AES-

```
PS> python main.py --db-url sqlite:///vault.db add-dek --name src -f src_dek.hex
{
  "status": "ok",
  "action": "add-dek",
  "source_file": "src_dek.hex",
  "record": {
    "id": 2,
    "name": "src",
    "key_type": "dek",
    "algorithm": "IMPORTED",
    "key_bytes": 16,
    "created_at": "2026-03-01T05:35:34.545522",
    "updated_at": "2026-04-25T08:02:17.088711Z"
  },
  "dek_stored_as_wrapped": true
}
```

Рисунок 21 – Создание src DEK

```
PS> python main.py --db-url sqlite:///vault.db add-dek --name dst -f dst_dek.hex
{
  "status": "ok",
  "action": "add-dek",
  "source_file": "dst_dek.hex",
  "record": {
    "id": 3,
    "name": "dst",
    "key_type": "dek",
    "algorithm": "IMPORTED",
    "key_bytes": 32,
    "created_at": "2026-03-01T05:35:35.108768",
    "updated_at": "2026-04-25T08:02:30.774061Z"
  },
  "dek_stored_as_wrapped": true
}
```

Рисунок 22 – Создание dst DEK

128-GCM, затем повторно шифрует алгоритмом ChaCha20-Poly1305 и сохраняет новый контейнер в таблице `data_records`. Одновременно `Authorization: Bearer123` не сохраняется в открытом виде, а преобразуется в PBKDF2-хеш. Этот рисунок подтверждает достижение основной цели работы: централизованное управление ключами и контролируемое перешифрование данных. Команда представлена на рисунке 23.

Загрузка и перешифрование открытых данных в произвольной форме

Программа также поддерживает загрузку данных произвольной формы из файла. Для этого используется команда `store-plaintext`. Дальнейшее взаимодействие с такими данными ничем не отличается от основного метода взаимодействия с программой. Команда представлена на рисунке 24.

```
PS> python main.py --db-url sqlite:///vault.db load --name admin_data -f container.json --auth "
    Bearer 123" --alg-in AES-128-GCM --alg-out ChaCha20-Poly1305 --dek-in src --dek-out dst
{
  "status": "ok",
  "action": "load",
  "output_header": {
    "alg": "ChaCha20-Poly1305",
    "nonce": "a8b514fcc23093a22173349f",
    "tag": "f6aad059dd4b48ff91a16f8138273428"
  },
  "plaintext_bytes": 11,
  "record": {
    "id": 3,
    "name": "admin_data",
    "alg_in": "AES-128-GCM",
    "alg_out": "ChaCha20-Poly1305",
    "dek_in": "src",
    "dek_out": "dst",
    "auth_protected": true,
    "payload_bytes": 11,
    "has_header": true,
    "created_at": "2026-04-25T08:02:57.816990Z",
    "updated_at": "2026-04-25T08:02:57.816995Z"
  }
}
```

Рисунок 23 – Загрузка данных из контейнера

```
PS> python main.py --db-url sqlite:///vault.db store-plaintext --name admin_data2 -f .\sample.
    txt --auth "Bearer 123" --alg ChaCha20-Poly1305 --dek dst
{
  "status": "ok",
  "action": "store-plaintext",
  "output_header": {
    "alg": "ChaCha20-Poly1305",
    "nonce": "9263a2ebececb77e4d785752",
    "tag": "c960a5f4b351e7ba19cd2984f43e2d93"
  },
  "plaintext_bytes": 7,
  "record": {
    "id": 4,
    "name": "admin_data2",
    "alg_in": "ChaCha20-Poly1305",
    "alg_out": "ChaCha20-Poly1305",
    "dek_in": "dst",
    "dek_out": "dst",
    "auth_protected": true,
    "payload_bytes": 7,
    "has_header": true,
    "created_at": "2026-04-25T09:08:58.294473Z",
    "updated_at": "2026-04-25T09:08:58.294477Z"
  }
}
```

Рисунок 24 – Загрузка данных произвольного формата

Запуск защищенного API-сервера

Здесь демонстрируется перевод подготовленных данных в режим сетевой выдачи. Команда проверяет наличие TLS-ключа и сертификата, запускает `uvicorn` в фоновом процессе и сохраняет служебную информацию о процессе в `.vault_web_server.json`. Административная загрузка данных и их последу-

ющая выдача намеренно разделены: сначала запись формируется через CLI, затем уже существующая запись читается клиентом через минимальный HTTPS endpoint. Команда представлена на рисунке 25.

```
PS> python main.py --db-url sqlite:///vault.db web-server up --key server.key --cert server.crt
    --host 127.0.0.1 --port 9000
{
  "status": "ok",
  "action": "web-server-up",
  "pid": 16468,
  "host": "127.0.0.1",
  "port": 9000
}
```

Рисунок 25 – Запуск веб-сервера

Проверка доступности сервиса

Этот вспомогательный запрос показывает, что веб-сервер успешно поднят и готов обслуживать клиентские обращения. Он не затрагивает криптографические операции и не требует авторизации, поэтому его удобно использовать как промежуточную проверку корректности запуска перед основной демонстрацией выдачи данных. Команда представлена на рисунке 26.

```
PS> curl.exe -k https://127.0.0.1:9000/health
{"status":"ok"}
```

Рисунок 26 – Проверка состояния веб-сервера

Выдача данных по авторизованному запросу

Этот запрос является итоговой демонстрацией прикладного результата работы. API находит запись `admin_data` в таблице `data_records`, проверяет Bearer-заголовок относительно сохраненного `auth_hash`, извлекает `dek_out` из таблицы `keys`, считывает master KEK из окружения, выполняет извлечение ключа данных и расшифровывает контейнер непосредственно перед возвратом ответа. В ответ клиент получает JSON со значениями `name`,`algorithm`,`plaintext_hex`, опциональным `plaintext_utf8` и размером полезной нагрузки в байтах. Таким образом подтверждается, что сервис не только хранит зашифрованные данные, но и выдает их только при корректной авторизации и наличии согласованной ключевой конфигурации. Команда представлена на рисунке 27.

```
PS> curl.exe -k -H "Authorization: Bearer 123" https://127.0.0.1:9000/data/admin_data
{"name":"admin_data","algorithm":"ChaCha20-Poly1305","plaintext_hex":"68656c6c6f2d7661756c74","
    plaintext_utf8":"hello-vault","bytes":11}
```

Рисунок 27 – Выдача данных через API

Остановка API-сервера

Финальная команда завершает сценарий и показывает, что жизненный цикл веб-сервера также управляется средствами самого приложения. CLI считывает PID-файл, завершает фоновый процесс и удаляет служебное состояние. Команда показана на рисунке 28.

```
PS> python main.py web-server down
{
  "status": "ok",
  "action": "web-server-down",
  "stopped": true,
  "pid": 16468
}
```

Рисунок 28 – Остановка веб-сервера

Создание контейнера

Для глубокой интеграции с процессом разработки для создания контейнеров был выбран подход создания программного API для шифрования контейнеров вместо традиционного создания CLI команд. API реализован в виде функции `encrypt_container` из модуля `vault.crypto`. Пример скрипта, который создает зашифрованный контейнер представлен на рисунке 51 (Приложение Б, с. 109).

Сценарии, в которых такой подход оправдан:

- внутренний сервис хранения чувствительных данных. Приложение сохраняет не открытый текст, а контейнер, а потом получает данные обратно только по авторизованному запросу через `GET/data/{name}`;
- backend для нескольких микросервисов. Один сервис отвечает за создание и хранение контейнеров, а другие не работают с ключами напрямую, а обращаются к Vault API за расшифрованными данными;
- защищенное хранение конфигурации или технических секретов. Например, токены интеграций, ключи внешних API, параметры доступа к стендам. Их можно хранить в виде контейнеров и выдавать только авторизованным

клиентам;

- локальные и тестовые контуры разработки. Когда команде нужен не тяжелый KMS, а компактный сервис, который дает единый процесс: положить данные, зашифровать, хранить, выдать по токену.

ЗАКЛЮЧЕНИЕ

В рамках дипломной работы была рассмотрена задача разработки защищенного сервиса управления криптографическими ключами и зашифрованными данными. Актуальность выбранной темы определяется тем, что современные программные системы используют большое количество чувствительных данных: ключи шифрования, токены доступа, технические учетные данные и конфиденциальные пользовательские сведения. При отсутствии централизованного и воспроизводимого механизма работы с такими данными возрастают риски их утечки, некорректного хранения, случайной публикации в репозиториях и несанкционированного доступа.

Цель работы заключалась в проектировании и реализации прикладного сервиса Vault, который обеспечивает безопасную обработку зашифрованных данных, управление ключевым материалом и авторизованную выдачу данных через API. Поставленная цель была достигнута: разработан программный комплекс, включающий консольный интерфейс администратора, криптографический модуль, модуль управления ключами, слой хранения на базе SQLAlchemy, FastAPI-приложение и набор автоматизированных проверок.

В ходе анализа предметной области были рассмотрены существующие подходы и решения для управления секретами и ключами: HashiCorp Vault, облачные KMS/Secrets Manager-сервисы, PAM/Secrets-платформы, self-hosted решения и GitOps-инструменты шифрования конфигураций. По результатам анализа был обоснован выбор собственного специализированного решения для учебно-прикладного проекта. Такой подход позволил сосредоточиться на ключевых механизмах безопасности: конвертное шифрование, разделении KEK и DEK, контроле формата криптографического контейнера, авторизации запросов и воспроизводимости операций через CLI.

В проектной части была разработана модульно-слоистая архитектура системы. Входными точками стали `vault/cli.py` и `vault/api.py`, сервисный слой представлен модулями `vault/crypto.py`, `vault/key_manager.py` и `vault/auth.py`, а доступ к данным изолирован в `VaultRepository`. Такое разделение позволило снизить связанность компонентов: CLI и API используют общую криптографическую и валидационную логику, а работа с БД не дублируется в прикладных сценариях.

Ключевым архитектурным решением стала модель конвертного шифро-

вания. В реализованной системе данные шифруются с использованием DEK, а сами DEK сохраняются в базе данных только в обернутом виде. Master KEK не записывается в БД и передается в runtime через переменную окружения `VAULT_MASTER_KEK_HEX`. Благодаря этому компрометация базы данных сама по себе не дает злоумышленнику готового материала для расшифрования данных. В БД остаются зашифрованные контейнеры, обернутый DEK, метаданные алгоритмов и PBKDF2-хеши авторизационных заголовков.

Криптографическая часть реализована на основе AEAD-алгоритмов AES-128-GCM, AES-256-GCM и ChaCha20-Poly1305. Использование AEAD позволило объединить конфиденциальность и контроль целостности: при повреждении шифротекста, неверном ключе, неправильном nonce, несоответствии алгоритма или некорректном теге расшифрование завершается ошибкой. Для контейнера данных были зафиксированы обязательные поля `alg`,`nonce`,`tag` и `payload` в шестнадцатеричном представлении, а проверка этих полей вынесена в отдельный слой валидации.

Важным результатом работы стала реализация двух основных эксплуатационных сценариев. Первый сценарий связан с загрузкой и перешифрованием данных через CLI: администратор передает входной контейнер, указывает входной и выходной алгоритмы, имена DEK и Bearer-секрет, после чего система выполняет извлечение ключей, расшифрование, повторное шифрование и сохранение нового контейнера. Второй сценарий связан с выдачей данных через `GET/data/{name}`: API проверяет Authorization, извлекает запись, выполняет извлечение нужного DEK и расшифровывает данные только после успешной авторизации.

Также внимание было уделено безопасному хранению авторизационных данных. Bearer-заголовок не сохраняется в БД в открытом виде. Вместо этого используется PBKDF2-хеширование с солью и 200000 итераций. Проверка заголовка выполняется через `hmac.compare_digest`, что снижает риск утечки информации через различия во времени сравнения.

Для реализации проекта была выбрана методика разработки, близкая к TDD. Это решение оказалось обоснованным, поскольку безопасность системы зависит от набора строгих инвариантов: DEK не должен храниться открыто, API не должен выдавать открытый текст без валидного Bearer, контейнер должен проходить проверку структуры, а ошибки криптографических операций не

должны приводить к раскрытию данных. Автоматизированные тесты позволили закрепить эти инварианты и контролировать регрессии при изменении модулей.

Проверка решения выполнялась с помощью unit-тестов, интеграционных тестов и сквозного сценария `scripts/check_all.py`. Проверялись криптографические roundtrip-сценарии, валидация схем, работа CLI, API-доступ, отказ при неверной авторизации, миграционная совместимость БД и полный поток загрузки данных. Дополнительно был выполнен ручной тест, включающий создание ключей, формирование входного контейнера, перешифрование через CLI и чтение результата через API. По результатам проверки автоматизированные тесты и интеграционный сценарий завершились успешно.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 ГОСТ Р 34.14-2025. Национальный стандарт Российской Федерации. Информационная технология. Криптографическая защита информации. Термины и определения. – 15.04.2026.

2 CNSSI 4009-2015 National Information Assurance (IA) Glossary. – 17.04.2026.

3 NIST Special Publication 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf. – 18.04.2026.

4 NIST SP 800-57 Part 1 Rev. 5 Recommendation for Key Management: Part 1 – General. – 17.04.2026.

5 RFC 8018. PKCS #5: Password-Based Cryptography Specification. Version 2.1. URL: https://datatracker.ietf.org/doc/html/rfc8018. – 16.04.2026.

6 RFC 8439 ChaCha20 and Poly1305 for IETF Protocols. URL: https://datatracker.ietf.org/doc/html/rfc8439. – 18.04.2026.

7 Баланов, А. Н. Внедрение методологий в IT: Agile, Scrum и другие : учебное пособие для вузов / А. Н. Баланов. – 2-е изд., стер. – Санкт-Петербург : Лань, 2025. – 188 с.

8 Баланов, А. Н. Защита информационных систем. Кибербезопасность : учебное пособие для вузов / А. Н. Баланов. – 3-е изд., стер. – Санкт-Петербург : Лань, 2026. – 280 с.

9 Борисова, С. Н. Криптографические методы защиты информации: классическая криптография : учебное пособие / С. Н. Борисова. – Пенза : ПГУ, 2018. – 186 с.

10 Ермакова, А. Н. Управление ИТ-проектами : учебник / А. Н. Ермакова. – Ставрополь : СтГАУ, 2024 – Часть 1 – 2024. – 196 с.

11 Иванюгин, В. М. Основы информационной безопасности. Алгоритмы шифрования данных : методические указания / В. М. Иванюгин. – Москва : РТУ МИРЭА, 2021. – 30 с.

12 Карпов М. А., Лиманова Н. И. ВОПРОСЫ ПРАКТИЧЕСКОГО ПРИМЕНЕНИЯ КРИПТОГРАФИИ ДЛЯ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ДАННЫХ // Бюллетень науки и практики. 2023. №12.

13 Каширская, Е. Н. Криптографические системы : учебное пособие / Е. Н. Каширская, А. П. Кушнир. – Москва : РТУ МИРЭА, 2021. – 66 с.

14 Косицин, Д. Ю. Язык программирования Python : учебно-методическое пособие / Д. Ю. Косицин. – Минск : БГУ, 2021. – 136 с.

15 Методы оценки безопасности компьютерных систем : учебно-методическое пособие / составители Д. И. Раковский, А. Г. Симонян. – Москва : МТУСИ, 2025. – 43 с.

16 Панасенко Сергей Петрович НИЗКОРЕСУРСНОЕ АУТЕНТИФИЦИРОВАННОЕ ШИФРОВАНИЕ: ОБЗОР ПОДХОДОВ // ПДМ. Приложение. 2024. №17.

17 Рацеев, С. М. Криптография. Безопасные многосторонние вычисления : учебное пособие для вузов / С. М. Рацеев. – 2-е изд., испр. и доп. – Санкт-Петербург : Лань, 2025. – 540 с.

18 Сергеева, О. А. Программирование на Python : учебно-методическое пособие / О. А. Сергеева. – Кемерово : КемГУ, 2024. – 157 с.

19 Смуглов Александр Николаевич, Мельничук Виктор Алексеевич ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ // Альманах Пермского военного института войск национальной гвардии. 2026. №.

20 Формирование ответов на большие вызовы в контексте психолого-педагогической науки: сб. науч. ст. по материалам VII всерос. молодеж. науч.-практ. конф., 12 апр. 2024 г : сборник научных трудов. – Шадринск : ШГПУ, 2024. – 496 с.

ПРИЛОЖЕНИЕ А

```python
from vault.cli import main


if __name__ == "__main__":
    raise SystemExit(main())
```

Рисунок 29 – main.py

```
[pytest]
testpaths = tests
norecursedirs = vault_test_* integration_tmp_* .pytest_cache
addopts = -p no:cacheprovider
asyncio_default_fixture_loop_scope = function
```

Рисунок 30 – pytest.ini

```
SQLAlchemy>=2.0,<3.0
cryptography>=42.0,<46.0
pydantic>=2.10,<3.0
pydantic-settings>=2.6,<3.0
fastapi>=0.110,<1.0
uvicorn>=0.30,<1.0
```

Рисунок 31 – requirements.txt

```python
"""Vault CLI package."""

from .cli import main

__all__ = ["main"]
```

Рисунок 32 – vault/__init__.py

```python
from __future__ import annotations

import json
from contextlib import asynccontextmanager

from fastapi import FastAPI, Header, HTTPException, status

from vault.crypto import decrypt_container, validate_header
from vault.db import VaultRepository
from vault.settings import VaultSettings
from vault.key_manager import unwrap_dek_hex, validate_master_kek_hex
from vault.auth import verify_auth_header


```

```
15 ❶
16 def create_app(db_url: str, master_kek_hex: str | None = None) -> FastAPI:
17     @asynccontextmanager
18     async def lifespan(app_instance: FastAPI):
19         try:
20             yield
21         finally:
22             app_instance.state.repo.engine.dispose()
23 
24     app = FastAPI(title="Vault API", lifespan=lifespan)
25     repo = VaultRepository(db_url)
26     repo.init_db()
27     app.state.repo = repo
28     app.state.master_kek_hex = master_kek_hex
29 
30     @app.get("/health")
31     def health() -> dict[str, str]:
32         return {"status": "ok"}
33 
34     @app.get("/data/{name}")
35     def get_data(name: str, authorization: str | None = Header(default=None)) -> dict[str,
    object]:
36         repo: VaultRepository = app.state.repo
37         master_kek_hex: str | None = app.state.master_kek_hex
38         record = repo.get_data_record_by_name(name=name)
39         if record is None:
40             raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Record not found.
    ")
41 
42         if authorization is None or not verify_auth_header(authorization, record.auth_hash):
43             raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Unauthorized."
    )
44 
45         if not record.header_json:
46             raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="
    Missing record header.")
47 
48         try:
49             header = validate_header(json.loads(record.header_json), expected_alg=record.alg_out
    )
50         except (json.JSONDecodeError, ValueError) as exc:
51             raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail=str(
    exc)) from exc
52 
53         key_record = repo.get_key(name=record.dek_out, key_type="dek")
54         if key_record is None:
55             raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="DEK
    not found.")
56         if not master_kek_hex:
57             raise HTTPException(
58                 status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
59                 detail="VAULT_MASTER_KEK_HEX is required to unwrap DEK.",
60             )
61 
62         try:
63             normalized_master_kek = validate_master_kek_hex(master_kek_hex)
64             dek_hex = unwrap_dek_hex(key_record.key_hex, normalized_master_kek)
65             plaintext = decrypt_container(
66                 algorithm=record.alg_out,
```

```
                key_hex=dek_hex,
                header=header,
                payload_hex=record.payload_hex,
            )
        except ValueError as exc:
            raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail=str(
    exc)) from exc

        utf8_value: str | None
        try:
            utf8_value = plaintext.decode("utf-8")
        except UnicodeDecodeError:
            utf8_value = None

        return {
            "name": record.name,
            "algorithm": record.alg_out,
            "plaintext_hex": plaintext.hex(),
            "plaintext_utf8": utf8_value,
            "bytes": len(plaintext),
        }

    return app

_settings = VaultSettings()
app = create_app(_settings.db_url, _settings.master_kek_hex)
```

Рисунок 33 – vault/api.py

```
from __future__ import annotations

import hashlib
import hmac
import secrets

PBKDF2_ALGO = "sha256"
PBKDF2_ITERATIONS = 200_000
SALT_BYTES = 16
HASH_PREFIX = "pbkdf2_sha256"


(2)
def normalize_bearer_auth_header(value: str) -> str:
    text = value.strip()
    if not text.startswith("Bearer "):
        raise ValueError("Auth header must start with 'Bearer '.")
    if not text[len("Bearer ") :].strip():
        raise ValueError("Bearer token is empty.")
    return text


def hash_auth_header(auth_header: str) -> str:
    normalized = normalize_bearer_auth_header(auth_header)
    salt = secrets.token_bytes(SALT_BYTES)
    digest = hashlib.pbkdf2_hmac(
        PBKDF2_ALGO,
        normalized.encode("utf-8"),
```

```
        salt,
        PBKDF2_ITERATIONS,
    )
    return f"{HASH_PREFIX}${PBKDF2_ITERATIONS}${salt.hex()}${digest.hex()}"


❸
def verify_auth_header(candidate_header: str, stored_value: str) -> bool:
    try:
        normalized_candidate = normalize_bearer_auth_header(candidate_header)
    except ValueError:
        return False

    if not stored_value.startswith(f"{HASH_PREFIX}$"):
        # Backward compatibility for legacy plain-text rows.
        return hmac.compare_digest(normalized_candidate, stored_value)

    parts = stored_value.split("$")
    if len(parts) != 4:
        return False

    _, iterations_raw, salt_hex, digest_hex = parts
    try:
        iterations = int(iterations_raw)
        salt = bytes.fromhex(salt_hex)
        expected_digest = bytes.fromhex(digest_hex)
    except ValueError:
        return False

    actual_digest = hashlib.pbkdf2_hmac(
        PBKDF2_ALGO,
        normalized_candidate.encode("utf-8"),
        salt,
        iterations,
    )
    return hmac.compare_digest(actual_digest, expected_digest)
```

Рисунок 34 – vault/auth.py

```
from __future__ import annotations

import argparse
import json
import os
import secrets
import subprocess
import sys
import time
from pathlib import Path
from typing import Sequence

from pydantic import TypeAdapter, ValidationError

from vault.schemas import (
    AddKeyCommand,
    GenerateKeyCommand,
    HexStr,
```

```
    LoadCommand,
    StorePlaintextCommand,
    SUPPORTED_ALGORITHMS,
    WebServerUpCommand,
    parse_container_json,
)
from vault.settings import VaultSettings
from vault.key_manager import generate_kek_hex, normalize_hex_or_raise, unwrap_dek_hex,
    validate_master_kek_hex, wrap_dek_hex

PID_FILE = Path(".vault_web_server.json")


def _repo_or_exit(db_url: str):
    try:
        from vault.db import VaultRepository
    except ModuleNotFoundError as exc:  # pragma: no cover - depends on environment
        raise SystemExit(str(exc)) from exc

    repo = VaultRepository(db_url)
    repo.init_db()
    return repo


def _crypto_or_exit():
    try:
        from vault import crypto
    except ModuleNotFoundError as exc:  # pragma: no cover - depends on environment
        raise SystemExit(str(exc)) from exc
    return crypto


def build_parser() -> argparse.ArgumentParser:
    settings = VaultSettings()
    parser = argparse.ArgumentParser(
        prog="vault",
        description="Vault CLI: key management, data loading, and web-server lifecycle.",
    )
    parser.add_argument(
        "--db-url",
        default=settings.db_url,
        help="Database URL for SQLAlchemy, e.g. sqlite:///vault.db",
    )
    parser.set_defaults(handler=None)

    subparsers = parser.add_subparsers(dest="command", required=True)

    _add_generate_key_parser(subparsers, command_name="generate-dek", key_type="dek")
    _add_generate_key_parser(subparsers, command_name="generate-kek", key_type="kek")
    _add_import_key_parser(subparsers, command_name="add-dek", key_type="dek")
    _add_import_key_parser(subparsers, command_name="add-kek", key_type="kek")
    _add_load_parser(subparsers)
    _add_store_plaintext_parser(subparsers)
    _add_web_server_parser(subparsers)

    return parser


def _master_kek_or_exit() -> str:
```

```
    settings = VaultSettings()
    if not settings.master_kek_hex:
        raise SystemExit("VAULT_MASTER_KEK_HEX is required for DEK operations.")
    try:
        return validate_master_kek_hex(settings.master_kek_hex)
    except ValueError as exc:
        raise SystemExit(str(exc)) from exc


def _add_generate_key_parser(subparsers: argparse._SubParsersAction, command_name: str, key_type: str) -> None:
    parser = subparsers.add_parser(command_name, help=f"Generate a new {key_type.upper()} key.")
    parser.add_argument("--name", required=True, help="Logical key name.")
    parser.add_argument("--alg", required=True, help=f"Encryption algorithm: {', '.join(SUPPORTED_ALGORITHMS)}")
    parser.set_defaults(handler=handle_generate_key, key_type=key_type)


def _add_import_key_parser(subparsers: argparse._SubParsersAction, command_name: str, key_type: str) -> None:
    parser = subparsers.add_parser(command_name, help=f"Import an existing {key_type.upper()} key from file.")
    parser.add_argument("--name", required=True, help="Logical key name.")
    parser.add_argument("-f", "--file", required=True, help="Path to text file with hex key.")
    parser.set_defaults(handler=handle_add_key, key_type=key_type)


def _add_load_parser(subparsers: argparse._SubParsersAction) -> None:
    parser = subparsers.add_parser("load", help="Load encrypted data and re-encrypt it.")
    parser.add_argument("--name", required=True, help="Record name.")
    parser.add_argument("-f", "--file", required=True, help="Input file path.")
    parser.add_argument("--auth", required=True, help="Authorization header, e.g. Bearer 123.")
    parser.add_argument("--alg-in", required=True, help=f"Input data algorithm: {', '.join(SUPPORTED_ALGORITHMS)}")
    parser.add_argument("--alg-out", required=True, help=f"Target data algorithm: {', '.join(SUPPORTED_ALGORITHMS)}")
    parser.add_argument("--dek-in", required=True, help="Source DEK name.")
    parser.add_argument("--dek-out", required=True, help="Target DEK name.")
    parser.add_argument(
        "--header-json",
        metavar="JSON",
        help='Optional JSON header with fields like {"alg":"AES-256-GCM","nonce":"...","tag":"..."}.',
    )
    parser.add_argument("--data-hex", help="Optional encrypted payload in hex.")
    parser.set_defaults(handler=handle_load)


def _add_store_plaintext_parser(subparsers: argparse._SubParsersAction) -> None:
    parser = subparsers.add_parser("store-plaintext", help="Encrypt plaintext file and store resulting container.")
    parser.add_argument("--name", required=True, help="Record name.")
    parser.add_argument("-f", "--file", required=True, help="Plaintext input file path.")
    parser.add_argument("--auth", required=True, help="Authorization header, e.g. Bearer 123.")
    parser.add_argument("--alg", required=True, help=f"Target data algorithm: {', '.join(SUPPORTED_ALGORITHMS)}")
    parser.add_argument("--dek", required=True, help="Target DEK name.")
    parser.set_defaults(handler=handle_store_plaintext)

```

```
127
128 def _add_web_server_parser(subparsers: argparse._SubParsersAction) -> None:
129     parser = subparsers.add_parser("web-server", help="Manage Vault web server.")
130     action_parsers = parser.add_subparsers(dest="web_server_action", required=True)
131
132     up_parser = action_parsers.add_parser("up", help="Start HTTPS web server.")
133     up_parser.add_argument("--key", required=True, help="TLS private key path.")
134     up_parser.add_argument("--cert", required=True, help="TLS certificate path.")
135     up_parser.add_argument("--host", default="127.0.0.1", help="Bind host.")
136     up_parser.add_argument("--port", type=int, default=9000, help="Bind port.")
137     up_parser.set_defaults(handler=handle_web_server_up)
138
139     down_parser = action_parsers.add_parser("down", help="Stop web server.")
140     down_parser.set_defaults(handler=handle_web_server_down)
141
142
143 def _print_result(status: str, action: str, **details: object) -> int:
144     payload = {"status": status, "action": action, **details}
145     print(json.dumps(payload, ensure_ascii=False, indent=2, default=str))
146     return 0
147
148
149 def _validate_or_exit(model_cls, payload: dict):
150     try:
151         return model_cls.model_validate(payload)
152     except ValidationError as exc:
153         raise SystemExit(str(exc)) from exc
154
155
156 def handle_generate_key(args: argparse.Namespace) -> int:
157     cmd = _validate_or_exit(
158         GenerateKeyCommand,
159         {
160             "db_url": args.db_url,
161             "key_type": args.key_type,
162             "name": args.name,
163             "alg": args.alg,
164         },
165     )
166     if cmd.key_type == "kek":
167         kek_hex = generate_kek_hex(cmd.alg)
168         return _print_result(
169             status="ok",
170             action="generate-kek",
171             algorithm=cmd.alg,
172             key_hex=kek_hex,
173             stored_in_db=False,
174             note="Store this KEK outside DB and set VAULT_MASTER_KEK_HEX for runtime operations.
    ",
175         )
176
177     crypto = _crypto_or_exit()
178     dek_hex = secrets.token_hex(crypto.KEY_SIZES_BYTES[cmd.alg])
179     master_kek_hex = _master_kek_or_exit()
180     wrapped_dek = wrap_dek_hex(dek_hex, master_kek_hex)
181     repo = _repo_or_exit(cmd.db_url)
182     try:
183         record = repo.upsert_key(
184             name=cmd.name,
```

```
            key_type=cmd.key_type,
            algorithm=cmd.alg,
            key_hex=wrapped_dek,
        )
        return _print_result(
            status="ok",
            action=f"generate-{cmd.key_type}",
            record=repo.to_dict_key(record),
            dek_stored_as_wrapped=True,
        )
    finally:
        repo.close()


def handle_add_key(args: argparse.Namespace) -> int:
    cmd = _validate_or_exit(
        AddKeyCommand,
        {
            "db_url": args.db_url,
            "key_type": args.key_type,
            "name": args.name,
            "file": args.file,
        },
    )
    key_hex = cmd.file.read_text(encoding="utf-8").strip()
    try:
        normalized_key = normalize_hex_or_raise(key_hex)
    except ValueError as exc:
        raise SystemExit(str(exc)) from exc

    if cmd.key_type == "kek":
        return _print_result(
            status="ok",
            action="add-kek",
            source_file=str(cmd.file),
            key_bytes=len(normalized_key) // 2,
            stored_in_db=False,
            note="Set this value to VAULT_MASTER_KEK_HEX in your runtime environment.",
        )

    repo = _repo_or_exit(cmd.db_url)
    master_kek_hex = _master_kek_or_exit()
    wrapped_dek = wrap_dek_hex(normalized_key, master_kek_hex)
    try:
        record = repo.upsert_key(
            name=cmd.name,
            key_type=cmd.key_type,
            algorithm="IMPORTED",
            key_hex=wrapped_dek,
        )
        return _print_result(
            status="ok",
            action=f"add-{cmd.key_type}",
            source_file=str(cmd.file),
            record=repo.to_dict_key(record),
            dek_stored_as_wrapped=True,
        )
    finally:
        repo.close()
```

```
244
245
246 def handle_load(args: argparse.Namespace) -> int:
247     header_json = None
248     if args.header_json:
249         try:
250             header_json = json.loads(args.header_json)
251         except json.JSONDecodeError as exc:
252             raise SystemExit(f"Invalid JSON in --header-json: {exc}") from exc
253
254     cmd = _validate_or_exit(
255         LoadCommand,
256         {
257             "db_url": args.db_url,
258             "name": args.name,
259             "file": args.file,
260             "auth": args.auth,
261             "alg_in": args.alg_in,
262             "alg_out": args.alg_out,
263             "dek_in": args.dek_in,
264             "dek_out": args.dek_out,
265             "header_json": header_json,
266             "data_hex": args.data_hex,
267         },
268     )
269
270     crypto = _crypto_or_exit()
271     master_kek_hex = _master_kek_or_exit()
272     repo = _repo_or_exit(cmd.db_url)
273     try:
274         dek_in_record = repo.get_key(name=cmd.dek_in, key_type="dek")
275         if dek_in_record is None:
276             raise SystemExit(f"DEK not found: {cmd.dek_in}")
277
278         dek_out_record = repo.get_key(name=cmd.dek_out, key_type="dek")
279         if dek_out_record is None:
280             raise SystemExit(f"DEK not found: {cmd.dek_out}")
281
282         try:
283             dek_in_key_hex = unwrap_dek_hex(dek_in_record.key_hex, master_kek_hex)
284             dek_out_key_hex = unwrap_dek_hex(dek_out_record.key_hex, master_kek_hex)
285         except ValueError as exc:
286             raise SystemExit(str(exc)) from exc
287
288         input_payload_hex, input_header = _resolve_input_container(cmd)
289         try:
290             normalized_header = crypto.validate_header(input_header, expected_alg=cmd.alg_in)
291             plaintext = crypto.decrypt_container(
292                 algorithm=cmd.alg_in,
293                 key_hex=dek_in_key_hex,
294                 header=normalized_header,
295                 payload_hex=input_payload_hex,
296             )
297             output_payload_hex, output_header = crypto.encrypt_container(
298                 algorithm=cmd.alg_out,
299                 key_hex=dek_out_key_hex,
300                 plaintext=plaintext,
301             )
302         except ValueError as exc:
```

```
            raise SystemExit(str(exc)) from exc

        record = repo.upsert_data_record(
            name=cmd.name,
            auth_header=cmd.auth,
            alg_in=cmd.alg_in,
            alg_out=cmd.alg_out,
            dek_in=cmd.dek_in,
            dek_out=cmd.dek_out,
            header_json=json.dumps(output_header, ensure_ascii=False),
            payload_hex=output_payload_hex,
        )

        return _print_result(
            status="ok",
            action="load",
            output_header=output_header,
            plaintext_bytes=len(plaintext),
            record=repo.to_dict_data_record(record),
        )
    finally:
        repo.close()


def handle_web_server_up(args: argparse.Namespace) -> int:
    cmd = _validate_or_exit(
        WebServerUpCommand,
        {
            "key": args.key,
            "cert": args.cert,
            "host": args.host,
            "port": args.port,
        },
    )
    if PID_FILE.exists():
        try:
            current = json.loads(PID_FILE.read_text(encoding="utf-8"))
            existing_pid = int(current.get("pid", 0))
        except (json.JSONDecodeError, ValueError):
            existing_pid = 0
        if existing_pid and _is_process_alive(existing_pid):
            raise SystemExit(f"Web server is already running with pid={existing_pid}.")
        PID_FILE.unlink(missing_ok=True)

    command = [
        sys.executable,
        "-m",
        "uvicorn",
        "vault.api:app",
        "--host",
        cmd.host,
        "--port",
        str(cmd.port),
        "--ssl-keyfile",
        str(cmd.key),
        "--ssl-certfile",
        str(cmd.cert),
    ]
    env = os.environ.copy()
```

```
    env["VAULT_DB_URL"] = args.db_url

    creationflags = 0
    if os.name == "nt":
        creationflags = subprocess.CREATE_NEW_PROCESS_GROUP | subprocess.DETACHED_PROCESS

    try:
        process = subprocess.Popen(
            command,
            env=env,
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL,
            stdin=subprocess.DEVNULL,
            creationflags=creationflags,
        )
    except OSError:
        # Some environments reject detached creation flags; fallback to plain spawn.
        process = subprocess.Popen(
            command,
            env=env,
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL,
            stdin=subprocess.DEVNULL,
        )
    time.sleep(0.4)
    if process.poll() is not None:
        raise SystemExit("Failed to start web server process.")

    PID_FILE.write_text(
        json.dumps(
            {"pid": process.pid, "host": cmd.host, "port": cmd.port, "db_url": args.db_url},
            ensure_ascii=False,
            indent=2,
        ),
        encoding="utf-8",
    )
    return _print_result(
        status="ok",
        action="web-server-up",
        pid=process.pid,
        host=cmd.host,
        port=cmd.port,
    )


def handle_store_plaintext(args: argparse.Namespace) -> int:
    cmd = _validate_or_exit(
        StorePlaintextCommand,
        {
            "db_url": args.db_url,
            "name": args.name,
            "file": args.file,
            "auth": args.auth,
            "alg": args.alg,
            "dek": args.dek,
        },
    )

    crypto = _crypto_or_exit()
```

```
    master_kek_hex = _master_kek_or_exit()
    repo = _repo_or_exit(cmd.db_url)
    try:
        key_record = repo.get_key(name=cmd.dek, key_type="dek")
        if key_record is None:
            raise SystemExit(f"DEK not found: {cmd.dek}")

        try:
            dek_key_hex = unwrap_dek_hex(key_record.key_hex, master_kek_hex)
        except ValueError as exc:
            raise SystemExit(str(exc)) from exc

        plaintext = cmd.file.read_bytes()
        if not plaintext:
            raise SystemExit("Plaintext file is empty.")

        try:
            output_payload_hex, output_header = crypto.encrypt_container(
                algorithm=cmd.alg,
                key_hex=dek_key_hex,
                plaintext=plaintext,
            )
        except ValueError as exc:
            raise SystemExit(str(exc)) from exc

        record = repo.upsert_data_record(
            name=cmd.name,
            auth_header=cmd.auth,
            alg_in=cmd.alg,
            alg_out=cmd.alg,
            dek_in=cmd.dek,
            dek_out=cmd.dek,
            header_json=json.dumps(output_header, ensure_ascii=False),
            payload_hex=output_payload_hex,
        )
        return _print_result(
            status="ok",
            action="store-plaintext",
            output_header=output_header,
            plaintext_bytes=len(plaintext),
            record=repo.to_dict_data_record(record),
        )
    finally:
        repo.close()


def handle_web_server_down(_: argparse.Namespace) -> int:
    if not PID_FILE.exists():
        return _print_result(status="ok", action="web-server-down", stopped=False, reason="
    not_running")

    try:
        current = json.loads(PID_FILE.read_text(encoding="utf-8"))
        pid = int(current.get("pid", 0))
    except (json.JSONDecodeError, ValueError):
        pid = 0

    if not pid:
        PID_FILE.unlink(missing_ok=True)
```

```
        return _print_result(status="ok", action="web-server-down", stopped=False, reason="
    invalid_pid_file")

    if os.name == "nt":
        subprocess.run(["taskkill", "/PID", str(pid), "/T", "/F"], capture_output=True, text=
    True)
    else:
        try:
            os.kill(pid, 15)
        except ProcessLookupError:
            pass

    time.sleep(0.2)
    stopped = not _is_process_alive(pid)
    PID_FILE.unlink(missing_ok=True)
    return _print_result(status="ok", action="web-server-down", stopped=stopped, pid=pid)


def _is_process_alive(pid: int) -> bool:
    try:
        os.kill(pid, 0)
    except Exception:
        return False
    return True


def _resolve_input_container(cmd: LoadCommand) -> tuple[str, dict[str, str]]:
    if cmd.data_hex is not None:
        return cmd.data_hex, cmd.header_json.model_dump()

    content = cmd.file.read_text(encoding="utf-8").strip()
    if cmd.header_json is not None:
        try:
            payload_hex = TypeAdapter(HexStr).validate_python(content)
        except ValidationError as exc:
            raise SystemExit(str(exc)) from exc
        return payload_hex, cmd.header_json.model_dump()

    try:
        container = parse_container_json(content)
    except (json.JSONDecodeError, ValidationError) as exc:
        raise SystemExit(
            "Input file must contain valid JSON container with header/data, "
            "or pass --header-json with hex payload file."
        ) from exc
    return container.data, container.header.model_dump()


def main(argv: Sequence[str] | None = None) -> int:
    parser = build_parser()
    args = parser.parse_args(argv)
    if args.handler is None:
        parser.print_help()
        return 2
    return args.handler(args)


if __name__ == "__main__":
    raise SystemExit(main())
```

```
536
537 #
```

Рисунок 35 – vault/cli.py

```
1  from __future__ import annotations
2
3  import secrets
4
5  try:
6      from cryptography.hazmat.primitives.ciphers.aead import AESGCM, ChaCha20Poly1305
7  except ModuleNotFoundError as exc:  # pragma: no cover - depends on local env
8      raise ModuleNotFoundError(
9          "cryptography is required for encryption/decryption operations. Install dependencies
       first."
10     ) from exc
11
12
13 KEY_SIZES_BYTES = {
14     "AES-128-GCM": 16,
15     "AES-256-GCM": 32,
16     "ChaCha20-Poly1305": 32,
17 }
18
19 TAG_SIZE_BYTES = 16
20 NONCE_SIZE_BYTES = 12
21
22
23 def _normalize_hex(value: str) -> str:
24     raw = value.strip().lower()
25     if raw.startswith("0x"):
26         raw = raw[2:]
27     if not raw:
28         raise ValueError("Hex value is empty.")
29     if len(raw) % 2:
30         raise ValueError("Hex value length must be even.")
31     try:
32         bytes.fromhex(raw)
33     except ValueError as exc:
34         raise ValueError("Hex value contains non-hex characters.") from exc
35     return raw
36
37
38 def _aead_instance(algorithm: str, key: bytes):
39     if algorithm.startswith("AES-"):
40         return AESGCM(key)
41     if algorithm == "ChaCha20-Poly1305":
42         return ChaCha20Poly1305(key)
43     raise ValueError(f"Unsupported algorithm: {algorithm}")
44
45
46 def key_bytes_for_algorithm(key_hex: str, algorithm: str) -> bytes:
47     normalized = _normalize_hex(key_hex)
48     key = bytes.fromhex(normalized)
49     expected_size = KEY_SIZES_BYTES[algorithm]
50     if len(key) != expected_size:
51         raise ValueError(
```

```
            f"Invalid key size for {algorithm}: expected {expected_size} bytes, got {len(key)} bytes."
        )
    return key


(4)
def validate_header(header: dict[str, str], expected_alg: str | None = None) -> dict[str, str]:
    missing = {"alg", "nonce", "tag"} - set(header)
    if missing:
        raise ValueError(f"Header missing required fields: {', '.join(sorted(missing))}")

    normalized = {
        "alg": str(header["alg"]),
        "nonce": _normalize_hex(str(header["nonce"])),
        "tag": _normalize_hex(str(header["tag"])),
    }
    if expected_alg and normalized["alg"] != expected_alg:
        raise ValueError(f"Header alg mismatch: expected {expected_alg}, got {normalized['alg']}")

    nonce = bytes.fromhex(normalized["nonce"])
    tag = bytes.fromhex(normalized["tag"])
    if len(nonce) != NONCE_SIZE_BYTES:
        raise ValueError(f"Invalid nonce size: expected {NONCE_SIZE_BYTES} bytes, got {len(nonce)} bytes.")
    if len(tag) != TAG_SIZE_BYTES:
        raise ValueError(f"Invalid tag size: expected {TAG_SIZE_BYTES} bytes, got {len(tag)} bytes.")
    return normalized


def decrypt_container(*, algorithm: str, key_hex: str, header: dict[str, str], payload_hex: str) -> bytes:
    normalized_payload = _normalize_hex(payload_hex)
    normalized_header = validate_header(header, expected_alg=algorithm)
    key = key_bytes_for_algorithm(key_hex, algorithm)
    nonce = bytes.fromhex(normalized_header["nonce"])
    tag = bytes.fromhex(normalized_header["tag"])
    ciphertext = bytes.fromhex(normalized_payload)

    cipher = _aead_instance(algorithm, key)
    encrypted = ciphertext + tag
    return cipher.decrypt(nonce, encrypted, None)


def encrypt_container(*, algorithm: str, key_hex: str, plaintext: bytes) -> tuple[str, dict[str, str]]:
    key = key_bytes_for_algorithm(key_hex, algorithm)
    nonce = secrets.token_bytes(NONCE_SIZE_BYTES)
    cipher = _aead_instance(algorithm, key)
    encrypted = cipher.encrypt(nonce, plaintext, None)
    ciphertext = encrypted[:-TAG_SIZE_BYTES]
    tag = encrypted[-TAG_SIZE_BYTES:]

    header = {
        "alg": algorithm,
        "nonce": nonce.hex(),
        "tag": tag.hex(),
```

```
    }
    return ciphertext.hex(), header
```

Рисунок 36 – vault/crypto.py

```
from __future__ import annotations

from datetime import datetime, timezone
from typing import Any
import json
from pydantic import ValidationError
from vault.auth import hash_auth_header
from vault.db_schemas import (
    DataRecordLookupIn,
    DataRecordOut,
    DataRecordUpsertIn,
    KeyLookupIn,
    KeyRecordOut,
    KeyUpsertIn,
)

try:
    from sqlalchemy import DateTime, String, Text, UniqueConstraint, create_engine, inspect, select, text
    from sqlalchemy.orm import DeclarativeBase, Mapped, Session, mapped_column, sessionmaker
except ModuleNotFoundError as exc:  # pragma: no cover - depends on local env
    raise ModuleNotFoundError(
        "SQLAlchemy is required for database operations. Install dependencies first."
    ) from exc


def utcnow() -> datetime:
    return datetime.now(timezone.utc)


class Base(DeclarativeBase):
    pass


❺
class KeyRecord(Base):
    __tablename__ = "keys"
    __table_args__ = (UniqueConstraint("name", "key_type", name="uq_keys_name_type"),)

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(128), nullable=False)
    key_type: Mapped[str] = mapped_column(String(16), nullable=False)
    algorithm: Mapped[str] = mapped_column(String(64), nullable=False)
    key_hex: Mapped[str] = mapped_column(Text, nullable=False)
    created_at: Mapped[datetime] = mapped_column(DateTime(timezone=True), default=utcnow, nullable=False)
    updated_at: Mapped[datetime] = mapped_column(
        DateTime(timezone=True), default=utcnow, onupdate=utcnow, nullable=False
    )
```

```
51  ❻
52  class DataRecord(Base):
53      __tablename__ = "data_records"
54      __table_args__ = (UniqueConstraint("name", name="uq_data_records_name"),)
55
56      id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
57      name: Mapped[str] = mapped_column(String(128), nullable=False)
58      auth_hash: Mapped[str] = mapped_column(String(512), nullable=False)
59      alg_in: Mapped[str] = mapped_column(String(64), nullable=False)
60      alg_out: Mapped[str] = mapped_column(String(64), nullable=False)
61      dek_in: Mapped[str] = mapped_column(String(128), nullable=False)
62      dek_out: Mapped[str] = mapped_column(String(128), nullable=False)
63      header_json: Mapped[str | None] = mapped_column(Text, nullable=True)
64      payload_hex: Mapped[str] = mapped_column(Text, nullable=False)
65      created_at: Mapped[datetime] = mapped_column(DateTime(timezone=True), default=utcnow,
    nullable=False)
66      updated_at: Mapped[datetime] = mapped_column(
67          DateTime(timezone=True), default=utcnow, onupdate=utcnow, nullable=False
68      )
69
70
71  class VaultRepository:
72      def __init__(self, db_url: str) -> None:
73          self.engine = create_engine(db_url, future=True)
74          self.session_factory = sessionmaker(bind=self.engine, expire_on_commit=False, class_=
    Session, future=True)
75
76      def init_db(self) -> None:
77          self._migrate_data_records_auth_column()
78          Base.metadata.create_all(self.engine)
79
80      def _migrate_data_records_auth_column(self) -> None:
81          with self.engine.begin() as conn:
82              inspector = inspect(conn)
83              if "data_records" not in inspector.get_table_names():
84                  return
85
86              column_names = {column["name"] for column in inspector.get_columns("data_records")}
87              if "auth_hash" in column_names or "auth_header" not in column_names:
88                  return
89
90              conn.execute(text("ALTER TABLE data_records RENAME COLUMN auth_header TO auth_hash")
    )
91
92      def close(self) -> None:
93          self.engine.dispose()
94
95
```

```
96      ❼
97      def upsert_key(self, *, name: str, key_type: str, algorithm: str, key_hex: str) -> KeyRecord
        :
98          try:
99              payload = KeyUpsertIn.model_validate(
100                 {"name": name, "key_type": key_type, "algorithm": algorithm, "key_hex": key_hex}
101             )
102         except ValidationError as exc:
103             raise ValueError(str(exc)) from exc
104
105         with self.session_factory() as session:
106             record = session.execute(
107                 select(KeyRecord).where(KeyRecord.name == payload.name, KeyRecord.key_type ==
        payload.key_type)
108             ).scalar_one_or_none()
109             if record is None:
110                 record = KeyRecord(
111                     name=payload.name,
112                     key_type=payload.key_type,
113                     algorithm=payload.algorithm,
114                     key_hex=payload.key_hex,
115                 )
116                 session.add(record)
117             else:
118                 record.algorithm = payload.algorithm
119                 record.key_hex = payload.key_hex
120             session.commit()
121             return record
122
123     def get_key(self, *, name: str, key_type: str) -> KeyRecord | None:
124         try:
125             lookup = KeyLookupIn.model_validate({"name": name, "key_type": key_type})
126         except ValidationError as exc:
127             raise ValueError(str(exc)) from exc
128
129         with self.session_factory() as session:
130             return session.execute(
131                 select(KeyRecord).where(KeyRecord.name == lookup.name, KeyRecord.key_type ==
        lookup.key_type)
132             ).scalar_one_or_none()
133
134     def upsert_data_record(
135         self,
136         *,
137         name: str,
138         auth_header: str,
139         alg_in: str,
140         alg_out: str,
141         dek_in: str,
142         dek_out: str,
143         header_json: str | None,
144         payload_hex: str,
145     ) -> DataRecord:
146         try:
147             payload = DataRecordUpsertIn.model_validate(
148                 {
149                     "name": name,
150                     "auth_header": auth_header,
151                     "alg_in": alg_in,
```

```
                    "alg_out": alg_out,
                    "dek_in": dek_in,
                    "dek_out": dek_out,
                    "header_json": header_json,
                    "payload_hex": payload_hex,
                }
            )
        except ValidationError as exc:
            raise ValueError(str(exc)) from exc

        with self.session_factory() as session:
            record = session.execute(select(DataRecord).where(DataRecord.name == payload.name)).
    scalar_one_or_none()
            if record is None:
                record = DataRecord(
                    name=payload.name,
                    auth_hash=hash_auth_header(payload.auth_header),
                    alg_in=payload.alg_in,
                    alg_out=payload.alg_out,
                    dek_in=payload.dek_in,
                    dek_out=payload.dek_out,
                    header_json=payload.header_json,
                    payload_hex=payload.payload_hex,
                )
                session.add(record)
            else:
                record.auth_hash = hash_auth_header(payload.auth_header)
                record.alg_in = payload.alg_in
                record.alg_out = payload.alg_out
                record.dek_in = payload.dek_in
                record.dek_out = payload.dek_out
                record.header_json = payload.header_json
                record.payload_hex = payload.payload_hex
            session.commit()
            return record

    def get_data_record_by_name(self, *, name: str) -> DataRecord | None:
        try:
            lookup = DataRecordLookupIn.model_validate({"name": name})
        except ValidationError as exc:
            raise ValueError(str(exc)) from exc

        with self.session_factory() as session:
            return session.execute(select(DataRecord).where(DataRecord.name == lookup.name)).
    scalar_one_or_none()

    def to_dict_key(self, record: KeyRecord) -> dict[str, Any]:
        key_bytes = _stored_key_bytes(record.key_hex)
        out = KeyRecordOut.model_validate(
            {
                "id": record.id,
                "name": record.name,
                "key_type": record.key_type,
                "algorithm": record.algorithm,
                "key_bytes": key_bytes,
                "created_at": record.created_at,
                "updated_at": record.updated_at,
            }
        )
```

```
        return out.model_dump(mode="json")

    def to_dict_data_record(self, record: DataRecord) -> dict[str, Any]:
        out = DataRecordOut.model_validate(
            {
                "id": record.id,
                "name": record.name,
                "alg_in": record.alg_in,
                "alg_out": record.alg_out,
                "dek_in": record.dek_in,
                "dek_out": record.dek_out,
                "auth_protected": True,
                "payload_bytes": len(record.payload_hex) // 2,
                "has_header": record.header_json is not None,
                "created_at": record.created_at,
                "updated_at": record.updated_at,
            }
        )
        return out.model_dump(mode="json")


def _stored_key_bytes(value: str) -> int:
    text = value.strip()
    try:
        return len(bytes.fromhex(text))
    except ValueError:
        pass
    try:
        payload = json.loads(text)
    except json.JSONDecodeError:
        return 0
    if isinstance(payload, dict) and isinstance(payload.get("wrapped"), str):
        try:
            return len(bytes.fromhex(payload["wrapped"]))
        except ValueError:
            return 0
    return 0
```

Рисунок 37 – vault/db.py

```
from __future__ import annotations

import json
from datetime import datetime
from typing import Literal

from pydantic import BaseModel, ConfigDict, Field, TypeAdapter, field_validator, model_validator

from vault.schemas import HexStr
from vault.auth import normalize_bearer_auth_header


class KeyUpsertIn(BaseModel):
    model_config = ConfigDict(extra="forbid")

    name: str = Field(min_length=1, max_length=128)
    key_type: Literal["dek", "kek"]
```

```
    algorithm: str = Field(min_length=1, max_length=64)
    key_hex: str

    @model_validator(mode="after")
    def validate_key_payload(self) -> "KeyUpsertIn":
        if self.key_type == "kek":
            _ = TypeAdapter(HexStr).validate_python(self.key_hex)
            return self

        # DEK may be stored as wrapped JSON or legacy hex.
        try:
            _ = TypeAdapter(HexStr).validate_python(self.key_hex)
            return self
        except Exception:
            pass

        try:
            payload = json.loads(self.key_hex)
        except json.JSONDecodeError as exc:
            raise ValueError("DEK value must be valid hex or wrapped JSON.") from exc
        if not isinstance(payload, dict):
            raise ValueError("Wrapped DEK payload must be a JSON object.")
        missing = {"nonce", "wrapped", "tag"} - set(payload)
        if missing:
            raise ValueError(f"Wrapped DEK payload missing fields: {', '.join(sorted(missing))}"
    )
        return self


class KeyLookupIn(BaseModel):
    model_config = ConfigDict(extra="forbid")

    name: str = Field(min_length=1, max_length=128)
    key_type: Literal["dek", "kek"]


class DataRecordUpsertIn(BaseModel):
    model_config = ConfigDict(extra="forbid")

    name: str = Field(min_length=1, max_length=128)
    auth_header: str = Field(min_length=1, max_length=512)
    alg_in: str = Field(min_length=1, max_length=64)
    alg_out: str = Field(min_length=1, max_length=64)
    dek_in: str = Field(min_length=1, max_length=128)
    dek_out: str = Field(min_length=1, max_length=128)
    header_json: str | None = None
    payload_hex: HexStr

    @field_validator("header_json")
    @classmethod
    def validate_header_json(cls, value: str | None) -> str | None:
        if value is None:
            return value
        try:
            parsed = json.loads(value)
        except json.JSONDecodeError as exc:
            raise ValueError("header_json must be valid JSON string.") from exc
        if not isinstance(parsed, dict):
            raise ValueError("header_json must encode a JSON object.")
```

```
        return value

    @field_validator("auth_header")
    @classmethod
    def validate_auth_header(cls, value: str) -> str:
        return normalize_bearer_auth_header(value)


class DataRecordLookupIn(BaseModel):
    model_config = ConfigDict(extra="forbid")

    name: str = Field(min_length=1, max_length=128)


class KeyRecordOut(BaseModel):
    model_config = ConfigDict(extra="forbid")

    id: int
    name: str
    key_type: str
    algorithm: str
    key_bytes: int
    created_at: datetime
    updated_at: datetime


class DataRecordOut(BaseModel):
    model_config = ConfigDict(extra="forbid")

    id: int
    name: str
    alg_in: str
    alg_out: str
    dek_in: str
    dek_out: str
    auth_protected: bool
    payload_bytes: int
    has_header: bool
    created_at: datetime
    updated_at: datetime
```

Рисунок 38 – vault/db_schemas.py

```
from __future__ import annotations

import json
import secrets

from pydantic import TypeAdapter, ValidationError

from vault.crypto import KEY_SIZES_BYTES
from vault.schemas import HexStr

try:
    from cryptography.hazmat.primitives.ciphers.aead import AESGCM
except ModuleNotFoundError as exc:  # pragma: no cover - depends on local env
    raise ModuleNotFoundError(
```

```
        "cryptography is required for key wrapping operations. Install dependencies first."
    ) from exc


WRAP_ALG = "AES-256-GCM"
WRAP_NONCE_BYTES = 12
WRAP_TAG_BYTES = 16


def normalize_hex_or_raise(value: str) -> str:
    try:
        return TypeAdapter(HexStr).validate_python(value)
    except ValidationError as exc:
        raise ValueError(str(exc)) from exc


❽
def validate_master_kek_hex(master_kek_hex: str) -> str:
    normalized = normalize_hex_or_raise(master_kek_hex)
    if len(bytes.fromhex(normalized)) != 32:
        raise ValueError("VAULT_MASTER_KEK_HEX must be 32 bytes (64 hex chars).")
    return normalized


def generate_kek_hex(algorithm: str) -> str:
    size = KEY_SIZES_BYTES[algorithm]
    return secrets.token_hex(size)


❾
def wrap_dek_hex(dek_hex: str, master_kek_hex: str) -> str:
    dek = bytes.fromhex(normalize_hex_or_raise(dek_hex))
    master_kek = bytes.fromhex(validate_master_kek_hex(master_kek_hex))
    nonce = secrets.token_bytes(WRAP_NONCE_BYTES)
    encrypted = AESGCM(master_kek).encrypt(nonce, dek, None)
    payload = {
        "v": 1,
        "wrap_alg": WRAP_ALG,
        "nonce": nonce.hex(),
        "wrapped": encrypted[:-WRAP_TAG_BYTES].hex(),
        "tag": encrypted[-WRAP_TAG_BYTES:].hex(),
    }
    return json.dumps(payload, ensure_ascii=False)


def unwrap_dek_hex(stored_value: str, master_kek_hex: str) -> str:
    text = stored_value.strip()

    # Backward compatibility for legacy plaintext DEK values.
    try:
        return normalize_hex_or_raise(text)
    except ValueError:
        pass

    try:
        payload = json.loads(text)
    except json.JSONDecodeError as exc:
        raise ValueError("Stored DEK value is neither valid hex nor wrapped JSON.") from exc

```

```
    if not isinstance(payload, dict):
        raise ValueError("Wrapped DEK payload must be a JSON object.")
    missing = {"nonce", "wrapped", "tag"} - set(payload)
    if missing:
        raise ValueError(f"Wrapped DEK payload missing fields: {', '.join(sorted(missing))}")

    nonce = bytes.fromhex(normalize_hex_or_raise(str(payload["nonce"])))
    wrapped = bytes.fromhex(normalize_hex_or_raise(str(payload["wrapped"])))
    tag = bytes.fromhex(normalize_hex_or_raise(str(payload["tag"])))
    if len(nonce) != WRAP_NONCE_BYTES:
        raise ValueError(f"Invalid wrap nonce size: expected {WRAP_NONCE_BYTES} bytes.")
    if len(tag) != WRAP_TAG_BYTES:
        raise ValueError(f"Invalid wrap tag size: expected {WRAP_TAG_BYTES} bytes.")

    master_kek = bytes.fromhex(validate_master_kek_hex(master_kek_hex))
    dek = AESGCM(master_kek).decrypt(nonce, wrapped + tag, None)
    return dek.hex()
```

Рисунок 39 – vault/key_manager.py

```
from __future__ import annotations

from pathlib import Path
from typing import Annotated, Any, Literal

from pydantic import BaseModel, BeforeValidator, ConfigDict, Field, ValidationError,
    field_validator, model_validator
from vault.auth import normalize_bearer_auth_header

SUPPORTED_ALGORITHMS = (
    "AES-128-GCM",
    "AES-256-GCM",
    "ChaCha20-Poly1305",
)
SupportedAlgorithm = Literal["AES-128-GCM", "AES-256-GCM", "ChaCha20-Poly1305"]

TAG_SIZE_BYTES = 16
NONCE_SIZE_BYTES = 12


def _normalize_hex(value: Any) -> str:
    if not isinstance(value, str):
        raise ValueError("Hex value must be a string.")
    raw = value.strip().lower()
    if raw.startswith("0x"):
        raw = raw[2:]
    if not raw:
        raise ValueError("Hex value is empty.")
    if len(raw) % 2:
        raise ValueError("Hex value length must be even.")
    try:
        bytes.fromhex(raw)
    except ValueError as exc:
        raise ValueError("Hex value contains non-hex characters.") from exc
    return raw


```

```
HexStr = Annotated[str, BeforeValidator(_normalize_hex)]


class HeaderModel(BaseModel):
    model_config = ConfigDict(extra="forbid")

    alg: SupportedAlgorithm
    nonce: HexStr
    tag: HexStr

    @field_validator("nonce")
    @classmethod
    def validate_nonce_size(cls, value: str) -> str:
        nonce = bytes.fromhex(value)
        if len(nonce) != NONCE_SIZE_BYTES:
            raise ValueError(f"Invalid nonce size: expected {NONCE_SIZE_BYTES} bytes, got {len(nonce)} bytes.")
        return value

    @field_validator("tag")
    @classmethod
    def validate_tag_size(cls, value: str) -> str:
        tag = bytes.fromhex(value)
        if len(tag) != TAG_SIZE_BYTES:
            raise ValueError(f"Invalid tag size: expected {TAG_SIZE_BYTES} bytes, got {len(tag)} bytes.")
        return value


class JsonContainerModel(BaseModel):
    model_config = ConfigDict(extra="forbid")

    header: HeaderModel
    data: HexStr


class FlatJsonContainerModel(BaseModel):
    model_config = ConfigDict(extra="forbid")

    alg: SupportedAlgorithm
    nonce: HexStr
    tag: HexStr
    data: HexStr

    def to_nested(self) -> JsonContainerModel:
        return JsonContainerModel(
            header=HeaderModel(alg=self.alg, nonce=self.nonce, tag=self.tag),
            data=self.data,
        )


class GenerateKeyCommand(BaseModel):
    model_config = ConfigDict(extra="forbid")

    db_url: str
    key_type: Literal["dek", "kek"]
    name: str = Field(min_length=1)
    alg: SupportedAlgorithm

```

```
94
95  class AddKeyCommand(BaseModel):
96      model_config = ConfigDict(extra="forbid")
97
98      db_url: str
99      key_type: Literal["dek", "kek"]
100     name: str = Field(min_length=1)
101     file: Path
102
103     @field_validator("file")
104     @classmethod
105     def validate_file_exists(cls, value: Path) -> Path:
106         if not value.is_file():
107             raise ValueError(f"File does not exist: {value}")
108         return value
109
110
111 class LoadCommand(BaseModel):
112     model_config = ConfigDict(extra="forbid")
113
114     db_url: str
115     name: str = Field(min_length=1)
116     file: Path
117     auth: str
118     alg_in: SupportedAlgorithm
119     alg_out: SupportedAlgorithm
120     dek_in: str = Field(min_length=1)
121     dek_out: str = Field(min_length=1)
122     header_json: HeaderModel | None = None
123     data_hex: HexStr | None = None
124
125     @field_validator("file")
126     @classmethod
127     def validate_file_exists(cls, value: Path) -> Path:
128         if not value.is_file():
129             raise ValueError(f"File does not exist: {value}")
130         return value
131
132     @field_validator("auth")
133     @classmethod
134     def validate_auth(cls, value: str) -> str:
135         return normalize_bearer_auth_header(value)
136
137     @model_validator(mode="after")
138     def validate_header_dependency(self) -> "LoadCommand":
139         if self.data_hex is not None and self.header_json is None:
140             raise ValueError("header-json is required when --data-hex is provided.")
141         return self
142
143
144 class StorePlaintextCommand(BaseModel):
145     model_config = ConfigDict(extra="forbid")
146
147     db_url: str
148     name: str = Field(min_length=1)
149     file: Path
150     auth: str
151     alg: SupportedAlgorithm
152     dek: str = Field(min_length=1)
```

```
153
154     @field_validator("file")
155     @classmethod
156     def validate_file_exists(cls, value: Path) -> Path:
157         if not value.is_file():
158             raise ValueError(f"File does not exist: {value}")
159         return value
160
161     @field_validator("auth")
162     @classmethod
163     def validate_auth(cls, value: str) -> str:
164         return normalize_bearer_auth_header(value)
165
166
167 class WebServerUpCommand(BaseModel):
168     model_config = ConfigDict(extra="forbid")
169
170     key: Path
171     cert: Path
172     host: str = "127.0.0.1"
173     port: int = Field(default=9000, ge=1, le=65535)
174
175     @field_validator("key", "cert")
176     @classmethod
177     def validate_file_exists(cls, value: Path) -> Path:
178         if not value.is_file():
179             raise ValueError(f"File does not exist: {value}")
180         return value
181
182
183 def parse_container_json(raw: str) -> JsonContainerModel:
184     import json
185
186     data = json.loads(raw)
187     try:
188         return JsonContainerModel.model_validate(data)
189     except ValidationError:
190         flat = FlatJsonContainerModel.model_validate(data)
191         return flat.to_nested()
```

Рисунок 40 – vault/schemas.py

```
1  from __future__ import annotations
2
3  from pydantic_settings import BaseSettings, SettingsConfigDict
4
5
6  class VaultSettings(BaseSettings):
7      model_config = SettingsConfigDict(env_prefix="VAULT_", extra="ignore")
8
9      db_url: str = "sqlite:///vault.db"
10     master_kek_hex: str | None = None
```

Рисунок 41 – vault/settings.py

```python
from __future__ import annotations

import time
from pathlib import Path


def _unlink_with_retries(path: Path, retries: int = 10, delay_sec: float = 0.1) -> None:
    for _ in range(retries):
        try:
            path.unlink()
            return
        except FileNotFoundError:
            return
        except OSError:
            time.sleep(delay_sec)


def pytest_sessionfinish(session, exitstatus):  # type: ignore[no-untyped-def]
    root = Path(session.config.rootpath)
    for pattern in ("test_api_*.db", "test_cli_*.db"):
        for db_file in root.glob(pattern):
            _unlink_with_retries(db_file)
```

Рисунок 42 – tests/conftest.py

```python
import json
import uuid
from pathlib import Path

import pytest
from fastapi.testclient import TestClient

from vault.api import create_app
from vault.crypto import encrypt_container
from vault.db import VaultRepository


@pytest.fixture
def api_env():
    db_path = Path(f"test_api_{uuid.uuid4().hex}.db").resolve()
    db_url = f"sqlite:///{db_path.as_posix()}"

    repo = VaultRepository(db_url)
    repo.init_db()
    master_kek_hex = "aa" * 32

    dek_hex = "33" * 32
    repo.upsert_key(name="api_dek", key_type="dek", algorithm="AES-256-GCM", key_hex=dek_hex)

    plaintext = b"api-plaintext"
    payload_hex, header = encrypt_container(
        algorithm="AES-256-GCM",
        key_hex=dek_hex,
        plaintext=plaintext,
    )
    repo.upsert_data_record(
```

```
        name="api_record",
        auth_header="Bearer 123",
        alg_in="AES-256-GCM",
        alg_out="AES-256-GCM",
        dek_in="api_dek",
        dek_out="api_dek",
        header_json=json.dumps(header),
        payload_hex=payload_hex,
    )

    client = TestClient(create_app(db_url, master_kek_hex=master_kek_hex))
    yield {"client": client, "repo": repo, "db_path": db_path}

    client.close()
    repo.close()
    try:
        db_path.unlink()
    except OSError:
        pass


def test_get_data_success(api_env):
    client = api_env["client"]
    response = client.get("/data/api_record", headers={"Authorization": "Bearer 123"})
    assert response.status_code == 200
    payload = response.json()
    assert payload["plaintext_utf8"] == "api-plaintext"
    assert payload["bytes"] == len(b"api-plaintext")


def test_get_data_unauthorized(api_env):
    client = api_env["client"]
    response = client.get("/data/api_record", headers={"Authorization": "Bearer wrong"})
    assert response.status_code == 401


def test_auth_is_hashed_in_storage(api_env):
    repo = api_env["repo"]
    record = repo.get_data_record_by_name(name="api_record")
    assert record is not None
    assert record.auth_hash != "Bearer 123"
    assert record.auth_hash.startswith("pbkdf2_sha256$")
```

Рисунок 43 – tests/test_api.py

```
import pytest
from pydantic import ValidationError

from vault.auth import hash_auth_header, normalize_bearer_auth_header, verify_auth_header
from vault.db_schemas import DataRecordUpsertIn


def test_empty_bearer_token_is_rejected():
    with pytest.raises(ValueError):
        normalize_bearer_auth_header("Bearer   ")


```

```
def test_malformed_hash_is_rejected():
    assert verify_auth_header("Bearer 123", "pbkdf2_sha256$not-a-number$aa$bb") is False
    assert verify_auth_header("Bearer 123", "pbkdf2_sha256$200000$zz$11") is False
    assert verify_auth_header("Bearer 123", "pbkdf2_sha256$200000$aa") is False


def test_long_bearer_token_limits():
    max_token = "t" * (512 - len("Bearer "))
    too_long_token = "t" * (513 - len("Bearer "))

    valid = DataRecordUpsertIn.model_validate(
        {
            "name": "n",
            "auth_header": f"Bearer {max_token}",
            "alg_in": "AES-256-GCM",
            "alg_out": "AES-256-GCM",
            "dek_in": "src",
            "dek_out": "dst",
            "header_json": None,
            "payload_hex": "00",
        }
    )
    assert valid.auth_header.startswith("Bearer ")

    with pytest.raises(ValidationError):
        DataRecordUpsertIn.model_validate(
            {
                "name": "n",
                "auth_header": f"Bearer {too_long_token}",
                "alg_in": "AES-256-GCM",
                "alg_out": "AES-256-GCM",
                "dek_in": "src",
                "dek_out": "dst",
                "header_json": None,
                "payload_hex": "00",
            }
        )


def test_hash_roundtrip_for_valid_bearer():
    stored = hash_auth_header("Bearer 123")
    assert verify_auth_header("Bearer 123", stored) is True
    assert verify_auth_header("Bearer wrong", stored) is False
```

Рисунок 44 – tests/test_auth.py

```
import uuid
from pathlib import Path

import pytest

from vault.cli import build_parser, main


@pytest.fixture
def parser():
    return build_parser()
```

```
12
13
14  def test_generate_dek_parser(parser):
15      args = parser.parse_args([
16          "generate-dek",
17          "--name",
18          "main",
19          "--alg",
20          "AES-256-GCM",
21      ])
22      assert args.command == "generate-dek"
23      assert args.name == "main"
24      assert args.alg == "AES-256-GCM"
25      assert args.key_type == "dek"
26
27
28  def test_web_server_down_parser(parser):
29      args = parser.parse_args(["web-server", "down"])
30      assert args.command == "web-server"
31      assert args.web_server_action == "down"
32
33
34  def test_store_plaintext_parser(parser):
35      plaintext_file = Path(__file__).resolve()
36      args = parser.parse_args([
37          "store-plaintext",
38          "--name",
39          "record-1",
40          "-f",
41          str(plaintext_file),
42          "--auth",
43          "Bearer 123",
44          "--alg",
45          "AES-256-GCM",
46          "--dek",
47          "main",
48      ])
49      assert args.command == "store-plaintext"
50      assert args.name == "record-1"
51      assert args.alg == "AES-256-GCM"
52      assert args.dek == "main"
53
54
55  def test_invalid_algorithm_fails():
56      db_file = Path(f"test_cli_invalid_{uuid.uuid4().hex}.db")
57      db_url = f"sqlite:///{db_file.resolve().as_posix()}"
58      try:
59          with pytest.raises(SystemExit) as exc:
60              main(
61                  [
62                      "--db-url",
63                      db_url,
64                      "generate-kek",
65                      "--name",
66                      "main",
67                      "--alg",
68                      "INVALID",
69                  ]
70              )
```

```
        assert "AES-128-GCM" in str(exc.value)
    finally:
        try:
            db_file.unlink()
        except FileNotFoundError:
            pass


def test_main_executes_command():
    db_file = Path(f"test_cli_main_{uuid.uuid4().hex}.db")
    db_url = f"sqlite:///{db_file.resolve().as_posix()}"
    try:
        try:
            code = main(
                [
                    "--db-url",
                    db_url,
                    "generate-kek",
                    "--name",
                    "master",
                    "--alg",
                    "AES-256-GCM",
                ]
            )
            assert code == 0
        except SystemExit as exc:
            message = str(exc)
            assert "SQLAlchemy is required" in message or "cryptography is required" in message
    finally:
        try:
            db_file.unlink()
        except FileNotFoundError:
            pass
```

Рисунок 45 – tests/test_cli.py

```
from vault.crypto import decrypt_container, encrypt_container


def test_aes256_roundtrip():
    key_hex = "11" * 32
    plaintext = b"vault-roundtrip-aes"
    payload_hex, header = encrypt_container(
        algorithm="AES-256-GCM",
        key_hex=key_hex,
        plaintext=plaintext,
    )
    decrypted = decrypt_container(
        algorithm="AES-256-GCM",
        key_hex=key_hex,
        header=header,
        payload_hex=payload_hex,
    )
    assert decrypted == plaintext


def test_chacha20_roundtrip():
```

```
    key_hex = "22" * 32
    plaintext = b"vault-roundtrip-chacha"
    payload_hex, header = encrypt_container(
        algorithm="ChaCha20-Poly1305",
        key_hex=key_hex,
        plaintext=plaintext,
    )
    decrypted = decrypt_container(
        algorithm="ChaCha20-Poly1305",
        key_hex=key_hex,
        header=header,
        payload_hex=payload_hex,
    )
    assert decrypted == plaintext
```

Рисунок 46 – tests/test_crypto.py

```
import uuid
from pathlib import Path

from sqlalchemy import inspect, text

from vault.db import VaultRepository


def test_migrates_data_records_auth_header_to_auth_hash():
    db_path = Path(f"test_migration_{uuid.uuid4().hex}.db").resolve()
    db_url = f"sqlite:///{db_path.as_posix()}"
    repo = VaultRepository(db_url)
    try:
        with repo.engine.begin() as conn:
            conn.execute(
                text(
                    """
                    CREATE TABLE data_records (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        name VARCHAR(128) NOT NULL,
                        auth_header VARCHAR(512) NOT NULL,
                        alg_in VARCHAR(64) NOT NULL,
                        alg_out VARCHAR(64) NOT NULL,
                        dek_in VARCHAR(128) NOT NULL,
                        dek_out VARCHAR(128) NOT NULL,
                        header_json TEXT NULL,
                        payload_hex TEXT NOT NULL,
                        created_at DATETIME NOT NULL,
                        updated_at DATETIME NOT NULL,
                        CONSTRAINT uq_data_records_name UNIQUE (name)
                    )
                    """
                )
            )

        repo.init_db()

        with repo.engine.begin() as conn:
            columns = {col["name"] for col in inspect(conn).get_columns("data_records")}
        assert "auth_hash" in columns
```

```
        assert "auth_header" not in columns

        record = repo.upsert_data_record(
            name="migrated",
            auth_header="Bearer 123",
            alg_in="AES-256-GCM",
            alg_out="AES-256-GCM",
            dek_in="src",
            dek_out="dst",
            header_json='{"alg":"AES-256-GCM","nonce":"00","tag":"11"}',
            payload_hex="00",
        )
        assert record.auth_hash.startswith("pbkdf2_sha256$")
    finally:
        repo.close()
        try:
            db_path.unlink()
        except OSError:
            pass
```

Рисунок 47 – tests/test_db_migration.py

```
import json
import os
from pathlib import Path
from uuid import uuid4

import pytest

from vault.cli import main


def test_load_reencrypts_and_stores():
    try:
        from vault.crypto import decrypt_container, encrypt_container
        from sqlalchemy import select
        from vault.db import DataRecord, VaultRepository
    except ModuleNotFoundError as exc:
        pytest.skip(str(exc))

    base_path = Path(".").resolve()
    stem = f"test_load_flow_{uuid4().hex}"
    db_path = (base_path / f"{stem}.db").resolve()
    src_key_file = (base_path / f"{stem}_src_dek.hex").resolve()
    dst_key_file = (base_path / f"{stem}_dst_dek.hex").resolve()
    container_file = (base_path / f"{stem}_container.json").resolve()

    previous_master_kek = os.environ.get("VAULT_MASTER_KEK_HEX")
    os.environ["VAULT_MASTER_KEK_HEX"] = "aa" * 32
    try:
        db_url = f"sqlite:///{db_path.as_posix()}"

        src_key_hex = "11" * 16
        dst_key_hex = "22" * 32

        try:
            src_key_file.write_text(src_key_hex, encoding="utf-8")
```

```
            dst_key_file.write_text(dst_key_hex, encoding="utf-8")
        except PermissionError as exc:
            pytest.skip(f"Filesystem write is restricted in this environment: {exc}")

        assert (
            main(
                [
                    "--db-url",
                    db_url,
                    "add-dek",
                    "--name",
                    "src",
                    "-f",
                    str(src_key_file),
                ]
            )
            == 0
        )
        assert (
            main(
                [
                    "--db-url",
                    db_url,
                    "add-dek",
                    "--name",
                    "dst",
                    "-f",
                    str(dst_key_file),
                ]
            )
            == 0
        )

        plaintext = b"vault-load-flow"
        input_payload_hex, input_header = encrypt_container(
            algorithm="AES-128-GCM",
            key_hex=src_key_hex,
            plaintext=plaintext,
        )
        try:
            container_file.write_text(
                json.dumps({"header": input_header, "data": input_payload_hex}, ensure_ascii=
    False),
                encoding="utf-8",
            )
        except PermissionError as exc:
            pytest.skip(f"Filesystem write is restricted in this environment: {exc}")

        assert (
            main(
                [
                    "--db-url",
                    db_url,
                    "load",
                    "--name",
                    "record-1",
                    "-f",
                    str(container_file),
                    "--auth",
```

```
                    "Bearer 123",
                    "--alg-in",
                    "AES-128-GCM",
                    "--alg-out",
                    "ChaCha20-Poly1305",
                    "--dek-in",
                    "src",
                    "--dek-out",
                    "dst",
                ]
            )
            == 0
        )

        repo = VaultRepository(db_url)
        try:
            record = repo.get_key(name="dst", key_type="dek")
            assert record is not None

            with repo.session_factory() as session:
                data_record = session.execute(select(DataRecord).where(DataRecord.name == "
    record-1")).scalar_one()

            output_header = json.loads(data_record.header_json)
            decrypted = decrypt_container(
                algorithm="ChaCha20-Poly1305",
                key_hex=dst_key_hex,
                header=output_header,
                payload_hex=data_record.payload_hex,
            )
            assert decrypted == plaintext
        finally:
            repo.close()
    finally:
        if previous_master_kek is None:
            os.environ.pop("VAULT_MASTER_KEK_HEX", None)
        else:
            os.environ["VAULT_MASTER_KEK_HEX"] = previous_master_kek
        for path in (container_file, src_key_file, dst_key_file, db_path):
            try:
                path.unlink(missing_ok=True)
            except OSError:
                pass
```

Рисунок 48 – tests/test_load_flow.py

```
from pathlib import Path

import pytest
from pydantic import ValidationError

from vault.schemas import HeaderModel, LoadCommand, StorePlaintextCommand


def test_header_validation():
    header = HeaderModel.model_validate(
        {
```

```
            "alg": "AES-256-GCM",
            "nonce": "00" * 12,
            "tag": "11" * 16,
        }
    )
    assert header.alg == "AES-256-GCM"


def test_load_requires_header_for_data_hex():
    with pytest.raises(ValidationError):
        LoadCommand.model_validate(
            {
                "db_url": "sqlite:///x.db",
                "name": "record",
                "file": str(Path(__file__).resolve()),
                "auth": "Bearer 123",
                "alg_in": "AES-128-GCM",
                "alg_out": "AES-256-GCM",
                "dek_in": "src",
                "dek_out": "dst",
                "data_hex": "00ff",
            }
        )


def test_store_plaintext_validates_auth_and_file():
    command = StorePlaintextCommand.model_validate(
        {
            "db_url": "sqlite:///x.db",
            "name": "record",
            "file": str(Path(__file__).resolve()),
            "auth": "Bearer 123",
            "alg": "AES-256-GCM",
            "dek": "main",
        }
    )
    assert command.auth == "Bearer 123"
```

Рисунок 49 – tests/test_schemas.py

```
from __future__ import annotations

import datetime as dt
import json
import os
import shutil
import ssl
import subprocess
import sys
import time
import urllib.error
import urllib.request
import uuid
from dataclasses import dataclass
from pathlib import Path

ROOT = Path(__file__).resolve().parents[1]
```

```
PYTHON = sys.executable
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))


@dataclass
class CheckResult:
    name: str
    ok: bool
    details: str


class IntegrationRunner:
    def __init__(self) -> None:
        self.results: list[CheckResult] = []
        self.temp_dir = ROOT / f"integration_tmp_{uuid.uuid4().hex}"
        self.temp_dir.mkdir(parents=True, exist_ok=True)
        self.api_process: subprocess.Popen[str] | None = None
        self.master_kek_hex = "aa" * 32
        self.common_env = os.environ.copy()
        self.common_env["VAULT_MASTER_KEK_HEX"] = self.master_kek_hex

    def add_result(self, name: str, ok: bool, details: str = "") -> None:
        self.results.append(CheckResult(name=name, ok=ok, details=details))
        status = "PASS" if ok else "FAIL"
        print(f"[{status}] {name}")
        if details and not ok:
            print(details)

    def run_cmd(self, name: str, args: list[str], expect_code: int = 0, contains: str | None =
    None) -> None:
        proc = subprocess.run(
            [PYTHON, *args],
            cwd=ROOT,
            env=self.common_env,
            capture_output=True,
            text=True,
        )
        output = (proc.stdout + "\n" + proc.stderr).strip()
        ok = proc.returncode == expect_code
        if contains is not None:
            ok = ok and (contains in output)
        self.add_result(name, ok, details=output[:1200])

    def run_unit_tests(self) -> None:
        self.run_cmd(
            name="pytest",
            args=["-m", "pytest", "-q", "tests"],
            expect_code=0,
            contains="passed",
        )

    def run_cli_checks(self) -> None:
        db_url = f"sqlite:///{(self.temp_dir / 'manual.db').as_posix()}"
        src_key_file = self.temp_dir / "src.hex"
        dst_key_file = self.temp_dir / "dst.hex"
        bad_key_file = self.temp_dir / "bad.hex"
        payload_file = self.temp_dir / "payload.hex"
        container_file = self.temp_dir / "container.json"
```

```
76
77          src_key_file.write_text("11" * 16 + "\n", encoding="ascii")
78          dst_key_file.write_text("22" * 32 + "\n", encoding="ascii")
79          bad_key_file.write_text("xyz\n", encoding="ascii")
80
81          self.run_cmd(
82              name="cli add-dek src",
83              args=["main.py", "--db-url", db_url, "add-dek", "--name", "src", "-f", str(
    src_key_file)],
84              expect_code=0,
85              contains='"status": "ok"',
86          )
87          self.run_cmd(
88              name="cli add-dek dst",
89              args=["main.py", "--db-url", db_url, "add-dek", "--name", "dst", "-f", str(
    dst_key_file)],
90              expect_code=0,
91              contains='"status": "ok"',
92          )
93
94          from vault.crypto import encrypt_container
95
96          payload, header = encrypt_container(
97              algorithm="AES-128-GCM",
98              key_hex="11" * 16,
99              plaintext=b"integration-data",
100         )
101         container_file.write_text(json.dumps({"header": header, "data": payload}), encoding="utf
    -8")
102
103         self.run_cmd(
104             name="cli load success",
105             args=[
106                 "main.py",
107                 "--db-url",
108                 db_url,
109                 "load",
110                 "--name",
111                 "rec1",
112                 "-f",
113                 str(container_file),
114                 "--auth",
115                 "Bearer 123",
116                 "--alg-in",
117                 "AES-128-GCM",
118                 "--alg-out",
119                 "ChaCha20-Poly1305",
120                 "--dek-in",
121                 "src",
122                 "--dek-out",
123                 "dst",
124             ],
125             expect_code=0,
126             contains='"status": "ok"',
127         )
128
129         self.run_cmd(
130             name="cli invalid algorithm",
131             args=["main.py", "--db-url", db_url, "generate-dek", "--name", "bad", "--alg", "
```

```
    INVALID"],
132             expect_code=1,
133             contains="Input should be",
134         )
135         self.run_cmd(
136             name="cli bad hex key",
137             args=["main.py", "--db-url", db_url, "add-dek", "--name", "badhex", "-f", str(
    bad_key_file)],
138             expect_code=1,
139             contains="Hex value",
140         )
141         self.run_cmd(
142             name="cli missing dek",
143             args=[
144                 "main.py",
145                 "--db-url",
146                 db_url,
147                 "load",
148                 "--name",
149                 "rec2",
150                 "-f",
151                 str(container_file),
152                 "--auth",
153                 "Bearer 123",
154                 "--alg-in",
155                 "AES-128-GCM",
156                 "--alg-out",
157                 "ChaCha20-Poly1305",
158                 "--dek-in",
159                 "missing",
160                 "--dek-out",
161                 "dst",
162             ],
163             expect_code=1,
164             contains="DEK not found",
165         )
166         self.run_cmd(
167             name="cli data-hex without header",
168             args=[
169                 "main.py",
170                 "--db-url",
171                 db_url,
172                 "load",
173                 "--name",
174                 "rec3",
175                 "-f",
176                 str(container_file),
177                 "--auth",
178                 "Bearer 123",
179                 "--alg-in",
180                 "AES-128-GCM",
181                 "--alg-out",
182                 "ChaCha20-Poly1305",
183                 "--dek-in",
184                 "src",
185                 "--dek-out",
186                 "dst",
187                 "--data-hex",
188                 "aa",
```

```
            ],
            expect_code=1,
            contains="header-json is required",
        )

        payload_file.write_text("0011\n", encoding="ascii")
        bad_header = json.dumps(
            {
                "alg": "AES-256-GCM",
                "nonce": "00112233445566778899aabb",
                "tag": "00112233445566778899aabbccddeeff",
            }
        )
        self.run_cmd(
            name="cli header alg mismatch",
            args=[
                "main.py",
                "--db-url",
                db_url,
                "load",
                "--name",
                "rec4",
                "-f",
                str(payload_file),
                "--header-json",
                bad_header,
                "--auth",
                "Bearer 123",
                "--alg-in",
                "AES-128-GCM",
                "--alg-out",
                "ChaCha20-Poly1305",
                "--dek-in",
                "src",
                "--dek-out",
                "dst",
            ],
            expect_code=1,
            contains="Header alg mismatch",
        )

        self.run_api_e2e(db_url=db_url)

    def _make_self_signed_cert(self, key_path: Path, cert_path: Path) -> None:
        from cryptography import x509
        from cryptography.hazmat.primitives import hashes, serialization
        from cryptography.hazmat.primitives.asymmetric import rsa
        from cryptography.x509.oid import NameOID
        import ipaddress

        key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
        subject = issuer = x509.Name(
            [
                x509.NameAttribute(NameOID.COUNTRY_NAME, "US"),
                x509.NameAttribute(NameOID.ORGANIZATION_NAME, "Vault Integration"),
                x509.NameAttribute(NameOID.COMMON_NAME, "127.0.0.1"),
            ]
        )
        cert = (
```

```
            x509.CertificateBuilder()
            .subject_name(subject)
            .issuer_name(issuer)
            .public_key(key.public_key())
            .serial_number(x509.random_serial_number())
            .not_valid_before(dt.datetime.now(dt.timezone.utc) - dt.timedelta(minutes=1))
            .not_valid_after(dt.datetime.now(dt.timezone.utc) + dt.timedelta(days=1))
            .add_extension(
                x509.SubjectAlternativeName(
                    [
                        x509.DNSName("localhost"),
                        x509.IPAddress(ipaddress.ip_address("127.0.0.1")),
                    ]
                ),
                critical=False,
            )
            .sign(key, hashes.SHA256())
        )

        key_path.write_bytes(
            key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.TraditionalOpenSSL,
                encryption_algorithm=serialization.NoEncryption(),
            )
        )
        cert_path.write_bytes(cert.public_bytes(serialization.Encoding.PEM))

    def run_api_e2e(self, *, db_url: str) -> None:
        key_path = self.temp_dir / "cert_key.pem"
        cert_path = self.temp_dir / "cert.pem"
        self._make_self_signed_cert(key_path=key_path, cert_path=cert_path)

        env = self.common_env.copy()
        env["VAULT_DB_URL"] = db_url
        self.api_process = subprocess.Popen(
            [
                PYTHON,
                "-m",
                "uvicorn",
                "vault.api:app",
                "--host",
                "127.0.0.1",
                "--port",
                "9666",
                "--ssl-keyfile",
                str(key_path),
                "--ssl-certfile",
                str(cert_path),
            ],
            cwd=ROOT,
            env=env,
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT,
            text=True,
        )

        ctx = ssl.create_default_context()
        ctx.check_hostname = False
```

```
        ctx.verify_mode = ssl.CERT_NONE

        def http_get(path: str, auth: str | None = None) -> tuple[int, str]:
            req = urllib.request.Request(f"https://127.0.0.1:9666{path}")
            if auth:
                req.add_header("Authorization", auth)
            with urllib.request.urlopen(req, context=ctx, timeout=5) as resp:
                return resp.status, resp.read().decode("utf-8")

        ready = False
        for _ in range(40):
            if self.api_process.poll() is not None:
                break
            try:
                status_code, body = http_get("/health")
                if status_code == 200 and "ok" in body:
                    ready = True
                    break
            except Exception:
                time.sleep(0.2)

        self.add_result("api health", ready, "Server did not become healthy.")
        if not ready:
            return

        try:
            status_code, body = http_get("/data/rec1", auth="Bearer 123")
            self.add_result(
                "api data authorized",
                status_code == 200 and "integration-data" in body,
                body,
            )
        except Exception as exc:
            self.add_result("api data authorized", False, str(exc))

        try:
            http_get("/data/rec1", auth="Bearer bad")
            self.add_result("api data unauthorized", False, "Expected 401 response.")
        except urllib.error.HTTPError as exc:
            self.add_result("api data unauthorized", exc.code == 401, f"HTTP {exc.code}")

        try:
            http_get("/data/missing", auth="Bearer 123")
            self.add_result("api data missing", False, "Expected 404 response.")
        except urllib.error.HTTPError as exc:
            self.add_result("api data missing", exc.code == 404, f"HTTP {exc.code}")

    def cleanup(self) -> None:
        if self.api_process is not None:
            self.api_process.terminate()
            try:
                self.api_process.wait(timeout=5)
            except subprocess.TimeoutExpired:
                self.api_process.kill()
        shutil.rmtree(self.temp_dir, ignore_errors=True)
        for db_file in ROOT.glob("test_api*.db"):
            try:
                db_file.unlink()
            except OSError:
```

```
                pass

    def summary(self) -> int:
        passed = sum(1 for item in self.results if item.ok)
        total = len(self.results)
        print(f"\nSummary: {passed}/{total} checks passed")
        if passed != total:
            print("Failed checks:")
            for item in self.results:
                if not item.ok:
                    print(f"- {item.name}")
            return 2
        return 0


def main() -> int:
    runner = IntegrationRunner()
    try:
        runner.run_unit_tests()
        runner.run_cli_checks()
        return runner.summary()
    finally:
        runner.cleanup()


if __name__ == "__main__":
    raise SystemExit(main())
```

Рисунок 50 – scripts/check_all.py

ПРИЛОЖЕНИЕ Б

```python
from __future__ import annotations

import argparse
import json
import sys
from pathlib import Path

ROOT = Path(__file__).resolve().parents[1]
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

from vault.crypto import encrypt_container


def build_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(
        description="Create an encrypted container from plaintext using Vault's Python API.",
    )
    parser.add_argument("-f", "--file", required=True, help="Path to plaintext input file.")
    parser.add_argument("--alg", required=True, help="Encryption algorithm.")
    parser.add_argument("--key-hex", required=True, help="Plaintext DEK in hex format.")
    parser.add_argument(
        "-o",
        "--output",
        help="Optional path to write JSON container. Prints to stdout when omitted.",
    )
    parser.add_argument(
        "--flat",
        action="store_true",
        help="Output flat JSON with alg/nonce/tag/data instead of nested header/data.",
    )
    return parser


def main(argv: list[str] | None = None) -> int:
    parser = build_parser()
    args = parser.parse_args(argv)

    plaintext = Path(args.file).read_bytes()
    if not plaintext:
        raise SystemExit("Plaintext file is empty.")

    payload_hex, header = encrypt_container(
        algorithm=args.alg,
        key_hex=args.key_hex,
        plaintext=plaintext,
    )
    if args.flat:
        container = {**header, "data": payload_hex}
    else:
        container = {"header": header, "data": payload_hex}

    rendered = json.dumps(container, ensure_ascii=False, indent=2)
    if args.output:
        Path(args.output).write_text(rendered + "\n", encoding="utf-8")
        print(args.output)
    else:
        print(rendered)
    return 0

```

```
61
62 if __name__ == "__main__":
63     raise SystemExit(main())
```

Рисунок 51 – Пример использования программного API для создания зашифрованного контейнера