

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Реализация библиотеки JWT для управления
доступом и безопасного обмена данными**

Студент

(подпись, дата)

Семичев Сергей Алексеевич

(ФИО полностью)

Руководитель

(подпись, дата)

Козлов А.Ю.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Семичев Сергей Алексеевич гр. 21ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Реализация библиотеки JWT для управления доступом и безопасного обмена данными

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы д.т.н, проф, зав. кафедрой «МО и ПЭВМ» Козлов А.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: программные средства для создания и валидации JWT в виде подключаемой библиотеки.*

2) *Среда выполнения: операционные системы Windows 10 и Ubuntu 22.04 LTS*

Языковые средства разработки: C, C++, UML

Инструментальные средства: MS Visual Studio 2022, VS Code, Draw.io,

Технологии разработки: ООП, практики разработки библиотек.

3) *Основные функции приложения:*

а) добавить в токен утверждения и заголовки;

б) подписать токен;

в) криптографически проверить токен;

г) проверить утверждения и заголовок токена;

д) получить данные из заголовка и утверждений токена.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи на разработку;

2) Анализ требований на разработку программных средств;

3) Проектирование программных средств;

4) Реализация программных средств;

5) Контроль качества программного обеспечения.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 162 листа, 24 рисунка, 6 таблиц и 26 использованных источников.

JWT, JWS, JWE, JWK, JWKS, JWA, БИБЛИОТЕКА, МОДУЛЬНОЕ ТЕСТИРОВАНИЕ, КРИПТОГРАФИЯ, OPENSSL.

Объект исследования – программная библиотека, реализующая набор функций для работы с технологией JWT.

Цель выпускной квалификационной работы – спроектировать и разработать программные средства, позволяющие создавать и валидировать JWT, а также работать с их заголовком и утверждениями.

Технологии разработки – разработка библиотеки, модульных тестов и примеров использования производится в среде разработки Microsoft Visual Studio 2022 для операционной системы Windows и текстовом редакторе Visual Studio Code для операционной системы Linux с использованием языков C и C++. Разработанное программное обеспечение рассчитано на операционные системы Windows и Linux, в частности проверялось на Windows 10 и Ubuntu 22.04 LTS.

Результаты работы – в ходе выполнения выпускной квалификационной работы были спроектированы и разработаны программные средства, позволяющие создавать и валидировать JWT, а также работать с их заголовком и утверждениями.

					ПГУ 09.03.04– 08ВР212.21 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата	Реализация библиотеки JWT для управления доступом и безопасного обмена данными. Пояснительная записка			Лит.	Лист	Листов	
Разраб.		Семичев С.А.									
Пров.		Козлов А.Ю.								4	162
								Группа 21ВР2			
Н. Контр.		Такташкин Д.В									
Утв.		Козлов А.Ю.									

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку.....	8
1.1 Принципы, структура и применение JWT.....	8
1.1.1 Назначение и контекст применения.....	8
1.1.2 Структура токена	8
1.1.3 Криптографические алгоритмы.....	10
1.1.4 Процесс генерации и валидации	11
1.2 Разработка и применение библиотек в программной инженерии	13
1.2.1 Цели и преимущества использования библиотек.....	13
1.2.2 Требования к библиотекам	14
1.2.3 Основные аспекты разработки библиотек	14
1.3 Анализ программного обеспечения аналогичного назначения	15
1.4 Постановка задачи на разработку программных средств.....	21
1.4.1 Цели разработки и назначение программных средств.....	21
1.4.2 Классы пользователей программных средств.....	22
1.4.3 Перечень автоматизируемых функций.....	23
1.5 Планирование разработки и оценка бюджета.....	23
2 Анализ требований на разработку программных средств	27
2.1 Функциональные требования к программному обеспечению	27
2.2 Требования к архитектуре программного обеспечения.....	32
2.3 Требования к интеграции и использованию	32
2.4 Обоснование выбора технологий и средств разработки.....	33
2.5 Требования к аппаратно-программному обеспечению.....	35

3	Проектирование программных средств	36
3.1	Проектирование внешнего API.....	36
3.2	Проектирование внутренней реализации API.....	38
4	Реализация программных средств.....	44
4.1	Разработка функций криптографических подписи и проверки	44
4.2	Разработка функций работы с утверждениями (claims) и заголовком (header).....	44
4.3	Разработка функций расширенной проверки утверждений и заголовка ..	45
4.4	Разработка функций для работы с криптографическими ключами.....	45
4.5	Разработка примеров использования библиотеки	46
5	Контроль качества программного обеспечения.....	48
5.1	Планирование процесса контроля качества программных средств	48
5.2	Тестирование программных средств.....	50
5.2.1	Модульное тестирование	50
5.2.2	Интеграционное тестирование	52
5.3	Оценка качества программного кода	53
5.3.1	Метрика Холстеда.....	53
5.3.2	Метрика Чепина	56
	Заключение	58
	Список источников	59
	Приложение А (обязательное) Листинг исходного кода библиотеки	62
	Приложение Б (рекомендуемое) Спецификации разработанных функций	129
	Приложение В (рекомендуемое) Листинг примеров использования	149

Введение

В последние годы цифровые системы стали не просто инструментами, а сложными экосистемами, взаимодействующими в реальном времени, через сети, протоколы и границы доменов. В таких условиях обеспечение надёжной идентификации и управления доступом стало одной из ключевых задач любой распределённой архитектуры.

С увеличением числа клиентских приложений, мобильных устройств и микросервисов возникла потребность в универсальном, гибком и независимом от состояния механизме обмена аутентификационной информацией. На эту роль естественным образом пришла технология JWT (JSON Web Token), предложившая простой формат передачи утверждений между участниками системы, оформленных в виде компактного и самодостаточного токена.

Популярность JWT обусловлена его простотой, универсальностью и независимостью от платформы. Он легко встраивается в REST API, мобильные приложения, микросервисы, фронтенд-архитектуры и системы авторизации третьих сторон. JWT оказался именно тем недостающим связующим элементом, который позволил системам обмениваться данными об аутентификации без централизованного хранилища состояния и без лишней нагрузки.

Целью данной выпускной квалификационной работы является разработка библиотеки, реализующей технологию JWT и предназначенной для интеграции в сторонние программные решения. Работа включает в себя анализ предметной области, в том числе ознакомление со стандартами RFC, анализ требований на разработку, проектирование и реализацию программных средств, а также контроль качества разработанного программного обеспечения.

1 Анализ предметной области и постановка задачи на разработку

1.1 Принципы, структура и применение JWT

1.1.1 Назначение и контекст применения

JWT (JSON [1] Web Token) [2] – это компактный формат передачи данных между участниками системы в виде подписанных и/или зашифрованных объектов. Основная цель – предоставить безопасный и самодостаточный механизм авторизации и аутентификации, особенно в stateless-приложениях, то есть без сохранения состояния. Токен содержит всю информацию и может проверяться локально, без обращения к хранилищу.

JWT применяется в следующих ситуациях:

- в RESTful API – токен содержит всю сессионную информацию, благодаря этому сервер может её не хранить;
- в микросервисной архитектуре – межсервисная аутентификация;
- в SPA и мобильных клиентах – после входа клиента, сервер возвращает JWT для последующих запросов;
- в OAuth 2.0 [3], OpenID Connect [4].

JWT может быть подписанным JWS (JSON Web Signature) [5] – для проверки подлинности и целостности, зашифрованным JWE (JSON Web Encryption) [6] – для защиты содержимого от несанкционированного доступа, или сочетать оба подхода, например сначала подписанным, а затем зашифрованным.

1.1.2 Структура токена

JWT представляет собой строку из нескольких частей разделенных точками. Каждая из этих частей закодирована в формате Base64URL [7].

В случае JWS строка состоит из трех частей и имеет следующий вид:

BASE64URL(HEADER) || '.' || BASE64URL(PAYLOAD) || '.' || BASE64URL(SIGNATURE),

где HEADER – метаданные о токене;

PAYLOAD – полезная нагрузка;

SIGNATURE – криптографическая подпись.

В случае JWE строка состоит из пяти частей и имеет вид:

BASE64URL(HEADER) || '.' || BASE64URL(ENCRYPTED_KEY) || '.' || BASE64URL(IV)
|| '.' || BASE64URL(CIPHERTEXT) || '.' || BASE64URL(TAG),

где HEADER – метаинформация о токене;

ENCRYPTED_KEY – зашифрованный симметричный ключ;

IV – вектор инициализации;

CIPHERTEXT – зашифрованный payload;

TAG – тег аутентификации (в AEAD режимах).

Метаинформацией о токене выступают пользовательские и зарегистрированные поля. Пользовательские поля определяет использующее приложение для своих служебных задач, зарегистрированные поля объявлены в стандарте и служат конкретным целям. Для JWS и JWE есть как общие поля, так и специфичные для каждого типа. Пример заголовка представлен на листинге 1.

Листинг 1 – Пример заголовка

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Полезная нагрузка (payload) – основная часть токена, представлена набором утверждений (claims). Утверждения делятся на зарезервированные стандартом («sub», «iat», «exp», «iss», «aud», «nbf» и другие) и пользовательские. Пример полезной нагрузки представлен в листинге 2.

Листинг 2 – Пример полезной нагрузки

```
{
  "sub": "1234567890",
  "name": "Alice",
  "admin": true,
  "iat": 1715137200
}
```

Ключи утверждений должны быть уникальны. Поведение при дублировании не определено стандартом, но часто берется последнее значение.

1.1.3 Криптографические алгоритмы

В JWA (JSON Web Algorithms) [8] определены криптографические алгоритмы для JWS и JWE по которым выполняется подпись и шифрование.

Для JWS представлены следующие алгоритмы:

- алгоритмы HMAC, использующие симметричный ключ, в виде HMAC с SHA-256 (HS256), HMAC с SHA-384 (HS384), HMAC с SHA-512 (HS512);
- алгоритмы RSA, использующие асимметричную пару ключей, закрытый и открытый, в виде RSA с SHA-256 (RS256), RSA с SHA-384 (RS384), RSA с SHA-512 (RS512) и RSA-PSS с SHA-256 (PS256), RSA-PSS с SHA-384 (PS384), RSA-PSS с SHA-512 (PS512);
- алгоритмы ECDSA и EdDSA, использующие асимметричную пару ключей на основе эллиптических кривых, в виде ECDSA алгоритмов – определены ECDSA с SHA-256 (ES256), ECDSA с SHA-384 (ES384), ECDSA с SHA-512 (ES512), ECDSA с SHA-256 (ES256K) с использованием кривых P-256, P-384, P-512, secp256k1 соответственно, и в виде EdDSA – более новой схемы подписи, которая используется с кривыми Ed25519 и Ed448 и определена в единственном варианте – EdDSA (без хэш-функции).

В JWE используются два типа алгоритмов:

- 1) алгоритмы управления ключом (alg);
- 2) алгоритмы шифрования данных (enc).

Первые используются для зашифровки и обработки симметричного ключа, который потом применяется для шифрования полезной нагрузки токена. Выделяют следующие группы алгоритмов:

- алгоритмы RSA, к которой относятся RSAES-PKCS1-v1_5 (RSA1_5), RSA-OAEP с SHA-1 (RSA-OAEP), RSAES-OAEP с SHA-256 (RSA-OAEP-256);
- алгоритмы AES Key Wrap, к которой относятся AES Key Wrap с 128-битным ключом (A128KW), AES Key Wrap с 192-битным ключом (A192KW), AES Key Wrap с 256-битным ключом (A256KW);

- передача ключа напрямую, без шифрования, такой алгоритм имеет название dir;
- алгоритмы ECDH-ES – Elliptic Curve Diffie-Hellman, к которой относятся ECDH Ephemeral Static (ECDH-ES), ECDH-ES с AES Key Wrap 128 бит (ECDH-ES+A128KW), ECDH-ES с AES Key Wrap 192 бит (ECDH-ES+A192KW), ECDH-ES с AES Key Wrap 256 бит (ECDH-ES+A256KW);
- алгоритмы Password-Bases Encryption (PBES2), к которой относятся PBES2 с HMAC SHA-256 и AES-128-KW (PBES2-HS256+A128KW), PBES2 с HMAC SHA-384 и AES-192-KW (HS384+A192KW), PBES2 с HMAC SHA-512 и AES-256-KW (HS512+A256KW).

Вторые используются для непосредственного шифрования полезной нагрузки токена. Все они обеспечивают аутентифицированное шифрование, то есть одновременно шифруют и защищают от модификации. Выделяют следующие:

- алгоритм AES_128_CBC с HMAC-SHA-256 (A128CBC-HS256);
- алгоритм AES_192_CBC с HMAC-SHA-384 (A192CBC-HS384);
- алгоритм AES_256_CBC с HMAC-SHA-512 (A256CBC-HS512);
- алгоритм AES in Galois/Counter Mode (GCM), 128 бит (A128GCM);
- алгоритм AES-GCM 192 бит (A192GCM);
- алгоритм AES-GCM 256 бит (A256GCM).

Несмотря на перечисленные выше алгоритмы, стандарт не запрещает добавление новых алгоритмов как для JWS, так и для JWE.

1.1.4 Процесс генерации и валидации

Как говорилось ранее, JWT-токены могут быть созданы двумя основными способами: с подписью (JWS) и с шифрованием (JWE). Далее будут рассмотрены генерация и валидация с компактной сериализацией.

Генерация JWS происходит в несколько этапов:

- 1) формирование заголовка (header);

- 2) формирование полезной нагрузки (payload);
- 3) кодирование заголовка и полезной нагрузки в Base64URL и соединение двух получившихся строк с разделителем в виде точки;
- 4) подпись получившейся строки с помощью выбранного алгоритма;
- 5) кодирование подписи (signature) в Base64URL и её присоединение к строке заголовка и полезной нагрузки с разделителем в виде точки.

Таким образом на выходе получается подписанный токен.

Валидация JWS имеет следующий порядок:

- 1) разбор токена на три части: заголовок, полезная нагрузка и подпись;
- 2) проверка подписи с использованием указанного алгоритма;
- 3) проверка утверждений и заголовков на соответствие ожидаемым значениям.

Генерация JWE происходит сложнее и тоже делится на несколько этапов:

- 1) формирование Protected Header, защищенного заголовка;
- 2) генерация Content Encryption Key (CEK), симметричного ключа, который будет использоваться для шифрования полезной нагрузки;
- 3) сгенерированный CEK шифруется с использованием ключа получателя по алгоритму управления ключом;
- 4) формирование вектора инициализации (Initialization Vector, IV). Он необходим для алгоритма шифрования полезной нагрузки;
- 5) шифрование полезной нагрузки с помощью указанного алгоритма, CEK и IV, на выходе получается шифртекст и тег аутентификации, если присутствует;
- 6) защищенный заголовок, CEK, IV, шифртекст и тег аутентификации кодируются в Base64URL и соединяются друг с другом в указанном порядке с разделителем в виде точки.

Таким образом создается зашифрованный токен.

Расшифровка и валидация JWE происходит следующим образом:

- 1) разбор токена на пять частей;

- 2) дешифровка СЕК с помощью закрытого ключа получателя или общего секретного пароля (shared secret);
- 3) расшифровка полезной нагрузки с помощью СЕК и IV по указанному алгоритму шифрования и проверка тега аутентификации, если этот тэг применим;
- 4) проверка утверждений и заголовков на соответствие ожидаемым значениям.

Проверка утверждений и заголовков не является обязательной, и её выполнение зависит от использующего приложения, однако зачастую одной криптографической проверки подписи бывает недостаточно.

1.2 Разработка и применение библиотек в программной инженерии

1.2.1 Цели и преимущества использования библиотек

Библиотеки – это сборник подпрограмм или объектов, используемых для разработки программного обеспечения. С точки зрения операционной системы и прикладного программного обеспечения, библиотеки разделяются на динамические и статические [9].

Их основными целями являются:

- повторное использование, избавление от дублирования кода и повторной реализации топовой логики;
- инкапсуляция, скрытие деталей реализации и работа только через явно определенный Application Programming Interface (API);
- разделение ответственности, каждая библиотека выполняет четко ограниченную функцию, что упрощает поддержку и тестирование;
- упрощение сопровождения, централизованное обновление библиотеки решает проблему технического долга в разных проектах;
- совместное использование, библиотеки можно публиковать и использовать в других проектах.

Использование библиотек дает множество преимуществ. Благодаря им снижается сложность программных систем, повышается масштабируемость и ускоряется разработка.

1.2.2 Требования к библиотекам

К библиотеке как самостоятельному компоненту предъявляются особые требования:

- архитектурные требования: библиотеки должны иметь явный и стабильный API, то есть интерфейсы не должны часто и резко меняться, а граница между публичным интерфейсом и её реализацией должна быть четкой, сама реализация должна быть инкапсулирована, кроме того, библиотеки должны иметь минимум зависимостей, чтобы их проще было внедрять и сопровождать, а также поддерживать паттерны вроде Plugin, Callback, позволяющие расширять функциональность без модификации кода и последующей перекомпиляции;
- требования к надежности и безопасности: библиотека должна проверять входные данные, не содержать в себе дефектов, в том числе относящихся к утечкам памяти и многопоточной работе, а также соблюдать правила стандартов языка и компилятора;
- практические требования: библиотека должна поддерживать наибольшее количество платформ и сборочных окружений.

Выполнение перечисленных выше требований позволяет создать удобную, универсальную и надежную библиотеку, так как соблюдение каждого из этих критериев повышает качество разработанного программного обеспечения и делает его более привлекательным для использования.

1.2.3 Основные аспекты разработки библиотек

Разработка библиотеки – это целый процесс, включающий в себя не только написание кода, но и тщательное тестирование, удобная дистрибуция и качественная документация.

Библиотека должна тщательно тестироваться. Это включает в себя как интеграционные и unit-тесты (модульные тесты), так и проверки в CI/CD. Чем больше кода покрывают тесты, тем лучше, в идеальном варианте вся логика библиотеки должна регулярно тестироваться. Помимо модульных и интеграционных тестов существуют и другие виды тестирования, все они могут быть полезны.

Дистрибуция библиотеки может происходить в нескольких вариантах, например в пакетах и архивах. Помимо них осуществима поддержка пакетных менеджеров вроде Conan, NuGet и других, а также генерация экспортируемых конфигураций. Очевидно, чем больше способов дистрибуции поддерживает библиотека, тем лучше.

Документация – это крайне важный аспект библиотеки. Она включает в себя как описание функций, классов и структур интерфейса, так и примеры использования библиотеки с пояснениями. Описание функций, классов и структур может быть оформлено как комментариями в коде, так и в виде вэб-документа или документа другого формата. Существуют инструменты позволяющие генерировать документы из комментариев в коде. В целом чем больше документации сделано для библиотеки, и чем более она информативна и доступна, в том числе в разных ее представлениях, тем лучше.

1.3 Анализ программного обеспечения аналогичного назначения

Существует большое количество библиотек, решающих ту же задачу, как в виде открытого, так и в виде проприетарного программного обеспечения. Достаточно полный список можно найти на портале JWT [10]. В списке указаны самые разные реализации с открытым исходным кодом для разных языков программирования и с разными функциональными возможностями. Для каждой реализации указана ссылка. Так как количество реализаций очень большое, было отобрано семь аналогов для сравнения по критериям их популярности и одобрению сообществом: node-jsonwebtoken [11], pyjwt [12], jjwt [13], jose [14],

go-jose [15], php-jwt [16], ruby-jwt [17]. Общая информация о них описана в таблице 1.

Таблица 1 – Общая информация о рассматриваемых аналогах

Название	Общее описание
node-jsonwebtoken	Популярная библиотека для Node.js, предоставляющая простые методы для создания и проверки JWT.
pyjwt	Библиотека для Python, позволяющая кодировать и декодировать JWT с поддержкой различных алгоритмов.
jjwt	Библиотека для Java, предоставляющая полный набор функций для работы с JWT, включая поддержку JWS, JWE и JWK.
jose	Библиотека для JavaScript, реализующая стандарты JOSE, включая JWS, JWE, JWK и JWT.
go-jose	Библиотека для Golang, реализующая стандарты JOSE, включая JWS, JWE, JWK и JWT.
php-jwt	Простая библиотека для PHP от Firebase для кодирования и декодирования JWT.
ruby-jwt	Библиотека для Ruby, реализующая стандарт JWT с поддержкой различных алгоритмов.

Сравнение проходило в первую очередь по функциональным возможностям библиотек, в частности подписи и проверки подписи JWS, шифрования и дешифрования JWE, а также поддержки JWK и JWKS. Также, хотя это и не является основной информацией, был учтен язык программирования для библиотеки или платформа. Условия использования и субъективные критерии использования, вроде удобства и требуемых зависимостей, не учитываются. Результаты данного сравнения представлены в таблице 2.

Таблица 2 – Функциональное сравнение выбранных библиотек

Название	Подпись JWS	Проверка JWS	Шифрование JWE	Дешифрование JWE	Поддержка JWK/JWKS	Платформа
node-jsonwebtoken	Да	Да	Нет	Нет	Нет	Node.js
pyjwt	Да	Да	Нет	Нет	Нет	Python
jjwt	Да	Да	Да	Да	Да	Java
jose	Да	Да	Да	Да	Да	JavaScript
go-jose	Да	Да	Да	Да	Да	Golang
php-jwt	Да	Да	Нет	Нет	Нет	PHP
ruby-jwt	Да	Да	Нет	Нет	Нет	Ruby

Отдельно были рассмотрены поддерживаемые алгоритмы для JWS и JWE, если имеется. Результаты представлены в таблице 3.

Таблица 3 – Поддерживаемые алгоритмы JWS и JWE для выбранных библиотек

Название	JWS	JWE
node-jsonwebtoken	HS256, HS384, HS512, RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES384, ES512.	-
pyjwt	HS256, HS384, HS512, RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES256K, ES384, ES512, EdDSA.	-
jjwt	HS256, HS384, HS512, RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES384, ES512, EdDSA.	alg: RSA-OAEP, RSA1_5, A128KW, A256KW, ECDH-ES. enc: A128CBC-HS256, A256CBC-HS512, A128GCM, A256GCM.

Продолжение таблицы 3

Название	JWS	JWE
jose	HS256, HS384, HS512, RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES256K, ES384, ES512, EdDSA.	alg: RSA-OAEP, RSA-OAEP-256, A128KW, A192KW, A256KW, ECDH-ES, ECDH-ES+A128KW, ECDH-ES+A256KW, dir. enc: A128GCM, A192GCM, A256GCM, A128CBC-HS256, A192CBC-HS284, A256CBC-HS512.
go-jose	HS256, HS384, HS512, RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES384, ES512, EdDSA.	alg: RSA-OAEP, RSA-OAEP-256, A128KW, A192KW, A256KW, ECDH-ES, ECDH-ES+A128KW, ECDH-ES+A256KW, dir. enc: A128GCM, A192GCM, A256GCM, A128CBC-HS256, A192CBC-HS284, A256CBC-HS512.
php-jwt	HS256, HS384, HS512, RS256, RS384, RS512.	-
ruby-jwt	HS256, HS384, HS512, RS256, RS384, RS512, PS256, PS384, PS512, ES256, ES256K, ES384, ES512, EdDSA.	-

Далее представлены листинги базовых примеров использования выбранных библиотек.

В листинге 3 представлен код примера использования библиотеки «node-jsonwebtoken».

Листинг 3 – Код примера использования библиотеки «node-jsonwebtoken»

```
const jwt = require('jsonwebtoken');
```

```
const token = jwt.sign({ userId: 123 }, 'secret', { algorithm:
  'HS256' });
console.log('Token:', token);

const decoded = jwt.verify(token, 'secret');
console.log('Decoded:', decoded);
```

В листинге 4 представлен пример использования библиотеки «pyjwt».

Листинг 4 – Код базового примера использования библиотеки «pyjwt»

```
import jwt

token = jwt.encode({'user_id': 123}, 'secret', algorithm='HS256')
print('Token:', token)

decoded = jwt.decode(token, 'secret', algorithms=['HS256'])
print('Decoded:', decoded)
```

В листинге 5 представлен пример использования библиотеки «jjwt».

Листинг 5 – Код базового примера использования библиотеки «jjwt»

```
import io.jsonwebtoken.*;

String token = Jwts.builder()
    .claim("userId", 123)
    .signWith(SignatureAlgorithm.HS256, "secret".getBytes())
    .compact();

System.out.println("Token: " + token);

Claims claims = Jwts.parser()
    .setSigningKey("secret".getBytes())
    .parseClaimsJws(token)
    .getBody();

System.out.println("Decoded userId: " + claims.get("userId"));
```

В листинге 6 представлен пример использования библиотеки «jose».

Листинг 6 – Код базового примера использования библиотеки «jose»

```
import { SignJWT, jwtVerify } from 'jose';

const secret = new TextEncoder().encode('secret');

const token = await new SignJWT({ userId: 123 })
    .setProtectedHeader({ alg: 'HS256' })
    .sign(secret);
```

```
console.log('Token:', token);

const { payload } = await jwtVerify(token, secret);
console.log('Decoded:', payload);
```

В листинге 7 представлен пример использования библиотеки «go-jose».

Листинг 7 – Код базового примера использования библиотеки «go-jose»

```
package main

import (
    "fmt"
    "gopkg.in/square/go-jose.v2/jwt"
    "gopkg.in/square/go-jose.v2"
    "time"
)

func main() {
    key := []byte("secret")
    signer, _ := jose.NewSigner(jose.SigningKey{Algorithm:
jose.HS256, Key: key}, nil)
    claims := jwt.Claims{Subject: "123", IssuedAt:
jwt.NewNumericDate(time.Now())}
    raw, _ :=
jwt.Signed(signer).Claims(claims).CompactSerialize()
    fmt.Println("Token:", raw)

    tok, _ := jwt.ParseSigned(raw)
    out := jwt.Claims{}
    _ = tok.Claims(key, &out)
    fmt.Println("Subject:", out.Subject)
}
```

В листинге 8 представлен пример использования библиотеки «php-jwt».

Листинг 8 – Код базового примера использования библиотеки «php-jwt»

```
<?php
use \Firebase\JWT\JWT;

require 'vendor/autoload.php';

$payload = ["user_id" => 123];
$token = JWT::encode($payload, "secret", 'HS256');
echo "Token: $token\n";

$decoded = JWT::decode($token, new \Firebase\JWT\Key("secret",
'HS256'));
print_r($decoded);
```

В листинге 9 представлен пример использования библиотеки «ruby-jwt».

Листинг 9 – Код базового примера использования библиотеки «ruby-jwt»

```
require 'jwt'

token = JWT.encode({ user_id: 123 }, 'secret', 'HS256')
puts "Token: #{token}"

decoded = JWT.decode(token, 'secret', true, { algorithm: 'HS256'
})
puts "Decoded: #{decoded}"
```

На основе результатов данного сравнения можно сделать вывод, что хотя полная реализация JWS, JWE и JWK/JWKS являются большим плюсом, немалой популярностью пользуются библиотеки, не предоставляющие полной реализации стандартов и не поддерживающие большую долю функционала. Это позволяет понять наиболее частые практические сценарии использования и расставить приоритеты при разработке собственной реализации. Также на основе примеров использования было сформировано представление об общих шаблонах программного интерфейса.

1.4 Постановка задачи на разработку программных средств

1.4.1 Цели разработки и назначение программных средств

Разрабатываемая библиотека предназначена для реализации технологии JSON Web Token (JWT) в программных продуктах, обеспечивая создание и валидацию токена, работу с заголовком, полезной нагрузкой и различными криптографическими алгоритмами, руководствуясь, в первую очередь, RFC, а также предоставлять широкие возможности по использованию ключевой информации в ее разных формах. Допустимо также создание удобных функций для легкого выполнения комплексных задач.

Библиотека может быть использована как в серверных, так и в клиентских приложениях, встраиваясь в решения, где требуются безопасная передача, подтверждение подлинности или защита данных.

Целями разработки являются:

- обеспечение совместимости со стандартами в отношении спецификаций, работы с зарегистрированными алгоритмами, структурами заголовков и утверждениями;
- поддержка гибких сценариев применения, в частности возможность создания и проверки токенов с различными параметрами и политиками;
- обеспечение расширяемости и модульности, чтобы библиотеку можно было легко дополнять новым функционалом в будущем;
- простота встраивания за счет чистого, удобного и минималистичного API, для основных операций и возможности использования в разных окружениях;
- безопасность и корректность реализации криптографических процедур и операций со структурой токена;
- свобода выбора в виде подробного API, не навязывающего собственные правила, особенно в части валидации токена, и дающее пользователю возможность накладывать эти правила по необходимости.

Так как тема JWT очень обширна, невозможно реализовать все детали этой технологии в данной работе. Однако планируется продолжительное сопровождение этой библиотеки и наполнение её дополнительным функционалом. Это дополнительная цель на неопределенное будущее.

1.4.2 Классы пользователей программных средств

Пользователем данной библиотеки является лишь одна категория – разработчики. Разработчики, имеющие отношение к написанию серверных приложений в виде различных REST API, микросервисов и шлюзов, к написанию клиентских приложений, разработчики систем безопасности в рамках специальных компонентов, таких как Identify Provider, Security Gateway, OAuth2.0/OpenID Connect серверов – все они могут быть пользователями создаваемой библиотеки.

1.4.3 Перечень автоматизируемых функций

Библиотека JWT должна обладать следующими автоматизируемыми функциями:

- создание токена;
- криптографическая валидация токена;
- валидация стандартных утверждений внутри токена, например поля «nbf», «iat», «exp»;
- выбор алгоритма создания и валидации токена;
- установка данных в заголовок токена, в том числе пользовательских;
- установка данных в полезную нагрузку токена, в том числе пользовательских;
- вспомогательные функции, такие как функции загрузки и экспорта ключей;
- предоставления подробной информации об ошибках;
- освобождения выделенных ресурсов в памяти и структур.

В процессе разработки библиотеки также будут появляться другие служебные и вспомогательные функции.

1.5 Планирование разработки и оценка бюджета

Для оценки временных затрат всего процесса разработки была разработана диаграмма Ганта с помощью средств Microsoft Office Project. Общий процесс содержит в себе пять вех. Все этапы выполняются поочередно друг за другом. Выделены следующие этапы:

- 1) анализ предметной области и постановка задачи на разработку;
- 2) анализ требований на разработку программных средств;
- 3) проектирование программных средств;
- 4) реализация программных средств;
- 5) контроль качества программного обеспечения.

Для каждого этапа был сформирован перечень задач. Содержание каждого этапа представлено на рисунке 1.

Название задачи
Начало проекта
Анализ предметной области и постановка задачи на разработку
Принципы, структура и применение JWT
Разработка и использование библиотек в программной инженерии
Анализ программного обеспечения аналогичного назначения
Постановка задачи на разработку программных средств
Анализ предметной области завершен
Анализ требований на разработку программных средств
Функциональные требования к программному обеспечению
Требования к архитектуре программного обеспечения
Требования к интеграции и использованию
Обоснование выбора технологий и средств разработки
Требования к аппаратно-программному обеспечению
Анализ требований завершен
Проектирование программных средств
Проектирование внешнего API
Проектирование внутренней реализации API
Проектирование завершенно
Реализация программных средств
Разработка функций криптографических подписи и проверки
Разработка функций работы с утверждениями (claims) и заголовком (header)
Разработка функций для расширенной проверки утверждений и заголовка
Разработка функций для работы с криптографическими ключами
Разработка примеров использования библиотеки
Реализация завершенна
Контроль качества программного обеспечения
Планирование процесса контроля качества программных средств
Тестирование программных средств
Оценка качества программного кода
Контроль качества завершен
Конец проекта

Рисунок 1 – Содержание этапов разработки

Диаграмма Ганта процесса разработки представлена на рисунке 2.

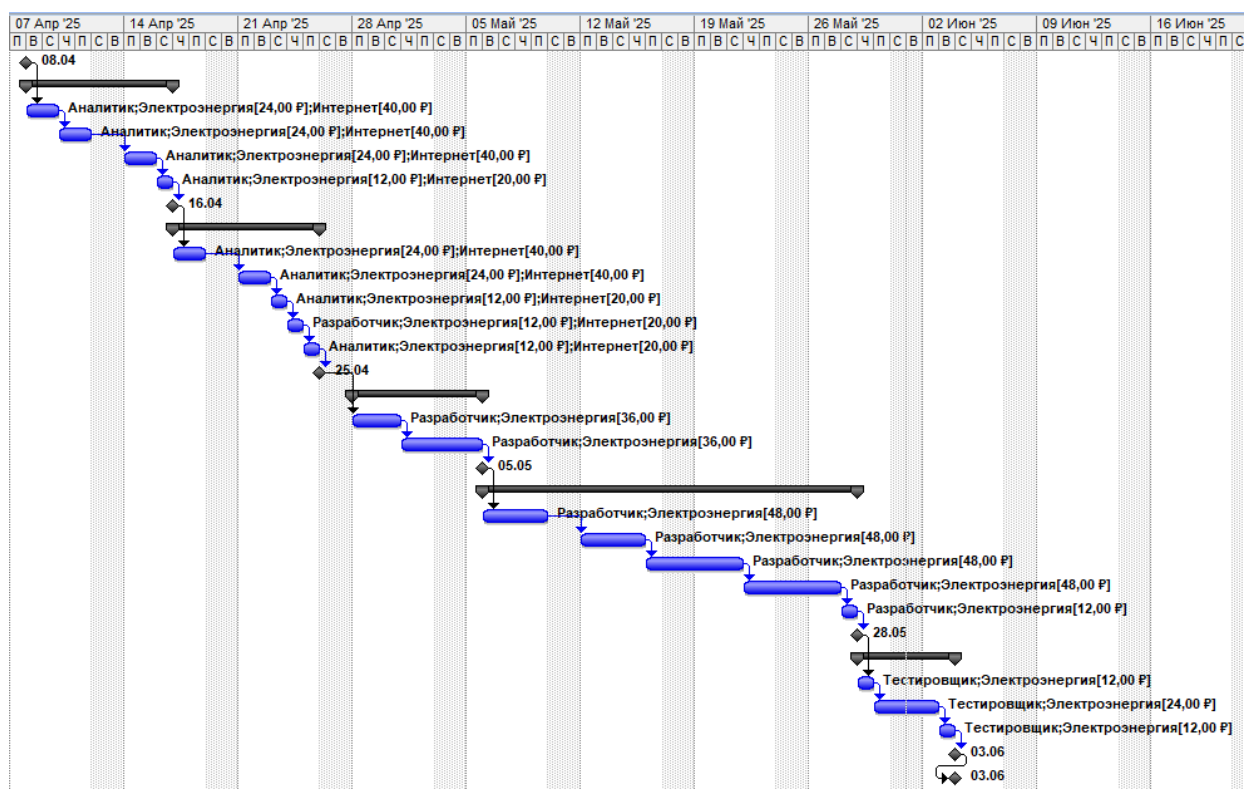


Рисунок 2 – Диаграмма Ганта для процесса разработки

На первом этапе выполняются задачи, направленные на анализ предметной области и программного обеспечения аналогичного назначения, а также постановка задачи на разработку.

Второй этап описывает подготовку различных требований: функциональных, к архитектуре, программно-аппаратному обеспечению, а также определяет используемые технологии и средства.

Третий этап предполагает проектирование программных средств, в частности проектирование внешней части, поставляемой для использования, и внутренней части реализации, основываясь на сформированных требованиях.

Четвертый этап предполагает реализацию программного проекта опираясь на сделанное раннее проектирование.

Пятый, заключительный этап, посвящен тестированию разработанного программного обеспечения и обоснованию используемых методов.

Были выделены следующие ресурсы:

- аналитик;
- разработчик;
- тестировщик;
- электроэнергия;
- сеть Интернет.

Для оценки финансовых задач была назначена стоимость используемым ресурсам проекта. Список ресурсов и их стоимости представлен на рисунке 3.

Название ресурса	Тип	Краткое название	Макс. единиц	Стандартная ставка	Ставка сверхурочных
Аналитик	Трудовой	А	100%	125,00 Р/ч	200,00 Р/ч
Разработчик	Трудовой	Р	100%	200,00 Р/ч	290,00 Р/ч
Тестировщик	Трудовой	Т	100%	140,00 Р/ч	220,00 Р/ч
Электроснабжение	Затраты	Э			
Интернет	Затраты	И			

Рисунок 3 – Список ресурсов и их стоимости

Исходя из стоимости ресурсов, была составлена общая оценка финансовых затрат проекта. Она представлена на рисунке 4.

Отчет о бюджете от Вс 01.06.25
Проект ВКР

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
20	Разработка функций криптографиче	0,00 Р	Пропорциональное	6 448,00 Р	0,00 Р	6 448,00 Р
21	Разработка функций работы с утве	0,00 Р	Пропорциональное	6 448,00 Р	0,00 Р	6 448,00 Р
22	Разработка функций для расширен	0,00 Р	Пропорциональное	6 448,00 Р	0,00 Р	6 448,00 Р
23	Разработка функций для работы с к	0,00 Р	Пропорциональное	6 448,00 Р	0,00 Р	6 448,00 Р
16	Проектирование внешнего API	0,00 Р	Пропорциональное	4 836,00 Р	0,00 Р	4 836,00 Р
17	Проектирование внутренней реализ	0,00 Р	Пропорциональное	4 836,00 Р	0,00 Р	4 836,00 Р
28	Тестирование программных средств	0,00 Р	Пропорциональное	2 264,00 Р	0,00 Р	2 264,00 Р
3	Принципы, структура и применение	0,00 Р	Пропорциональное	2 064,00 Р	0,00 Р	2 064,00 Р
4	Разработка и использование библи	0,00 Р	Пропорциональное	2 064,00 Р	0,00 Р	2 064,00 Р
5	Анализ программного обеспечения	0,00 Р	Пропорциональное	2 064,00 Р	0,00 Р	2 064,00 Р
9	Функциональные требования к прог	0,00 Р	Пропорциональное	2 064,00 Р	0,00 Р	2 064,00 Р
10	Требования к архитектуре программ	0,00 Р	Пропорциональное	2 064,00 Р	0,00 Р	2 064,00 Р
12	Обоснование выбора технологий и	0,00 Р	Пропорциональное	1 632,00 Р	0,00 Р	1 632,00 Р
24	Разработка примеров использования	0,00 Р	Пропорциональное	1 612,00 Р	0,00 Р	1 612,00 Р
27	Планирование процесса контроля к	0,00 Р	Пропорциональное	1 132,00 Р	0,00 Р	1 132,00 Р
29	Оценка качества программного код	0,00 Р	Пропорциональное	1 132,00 Р	0,00 Р	1 132,00 Р
6	Постановка задачи на разработку п	0,00 Р	Пропорциональное	1 032,00 Р	0,00 Р	1 032,00 Р
11	Требования к интеграции и использ	0,00 Р	Пропорциональное	1 032,00 Р	0,00 Р	1 032,00 Р
13	Требования к аппаратно-программн	0,00 Р	Пропорциональное	1 032,00 Р	0,00 Р	1 032,00 Р
1	Начало проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
7	Анализ предметной области завер	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
14	Анализ требований завершен	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
18	Проектирование завершено	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
25	Реализация завершена	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
30	Контроль качества завершен	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
31	Конец проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
		0,00 Р		56 652,00 Р	0,00 Р	56 652,00 Р

Рисунок 4 – Оценка финансовых затрат проекта

Таким образом, видно, что общие затраты на разработку проекта составляют 56 652 рублей, а её общая продолжительность равна 48 дней. Это приемлемая цена и срок выполнения работ.

2 Анализ требований на разработку программных средств

2.1 Функциональные требования к программному обеспечению

В рамках разработки данной библиотеки JWT будет реализовано только через создание JWS. Основной причиной является специфичность использования JWE, он сложный и на него нет большого спроса в индустрии. Из наиболее частых применений – это отправка информации по цепочке из нескольких машин до конкретной точки, в таком случае получить доступ к информации сможет только цель, так как у других участников сети не будет нужного ключа. Однако да данный момент чаще всего передача происходит от машины к машине, без посредников, а значит защищенного сетевого протокола, вроде HTTPS [18], будет вполне достаточно. Также JWE может использоваться в реализации OpenID Connect, но является опциональным, в отличие от обязательного JWS.

Кроме того, понадобится частичная реализация ещё одного связанного с JWT объекта – JWK (JSON Web Key) и JWKS (JSON Web Key Set) [19], а также реализация заголовков «x5c», связанных с сертификатами. Этот функционал не является необходимым, но является дополнительным способом получения открытого ключа для проверки, что может быть применимо в реальных приложениях. Таким образом, проверить подпись можно будет тремя способами: с использованием открытого ключа, ранее загруженного в программную структуру; с использованием открытого ключа из набора JWK по идентификатору ключа «kid»; с использованием открытого ключа из сертификата вложенной цепочки в заголовке «x5c».

Очевидно, что понадобятся вспомогательные функции для создания и проверки токена. В первую очередь это функции загрузки ключа, в частности функции загрузки ключа, как открытого, так и закрытого, из строки в формате PEM [20] [21] и JWK, а также функции загрузки набора ключей JWKS.

Ещё необходимы функции для работы с заголовками и полезной нагрузкой.

Исходя из описанных требований, были составлены диаграммы вариантов использования библиотеки [22]. На рисунке 5 представлена диаграмма вариантов использования библиотеки для создания и криптографических подписи и проверки подписи.

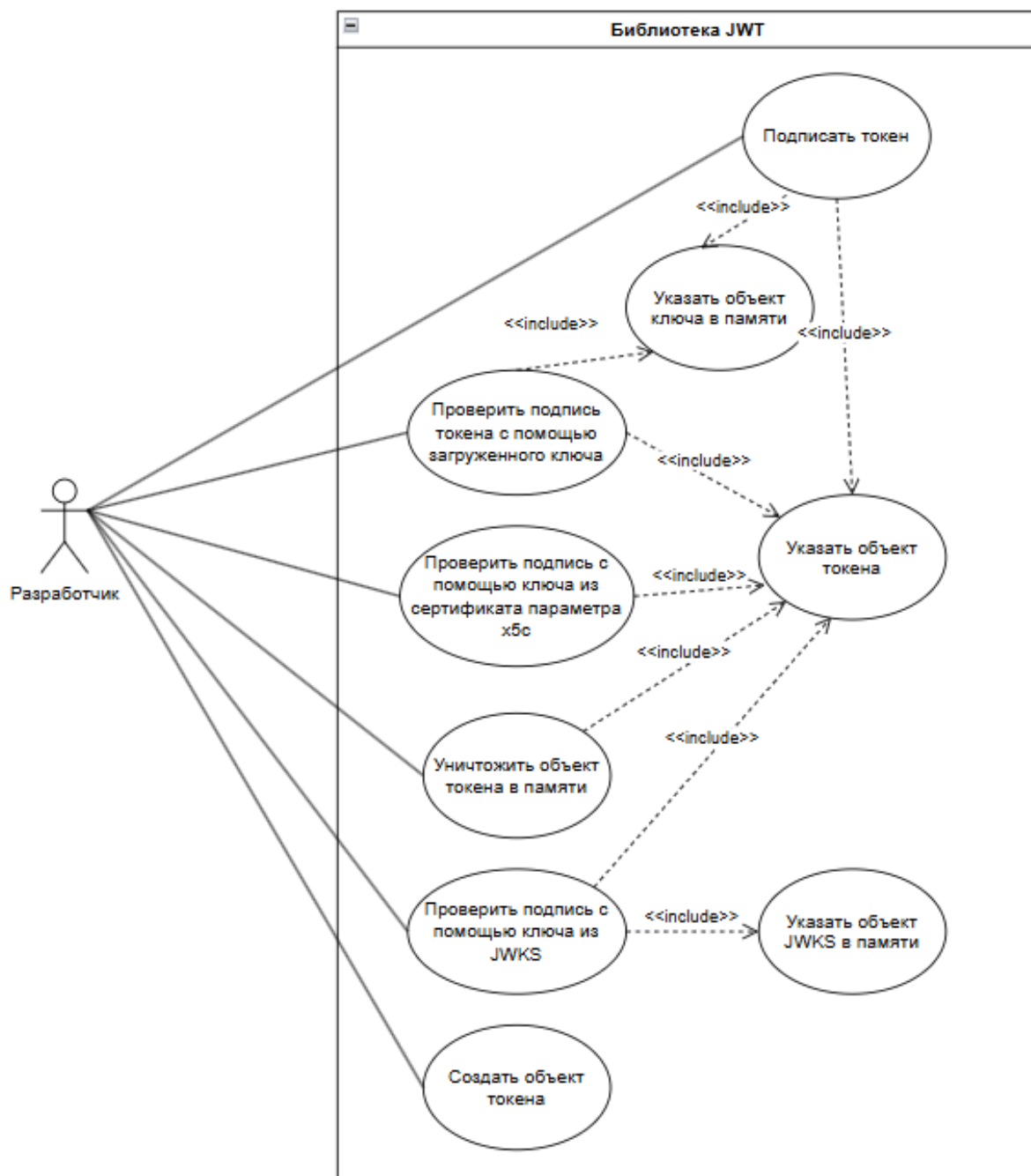


Рисунок 5 – Диаграмма вариантов использования библиотеки для создания и криптографических подписи и проверки JWT

На данном рисунке изображены варианты использования, относящиеся к криптографическим функциям подписи и проверки токена разными способами, а также вариантов создания и уничтожения объекта токена.

На рисунке 6 представлена диаграмма вариантов использования библиотеки для расширенной проверки утверждений и заголовка.

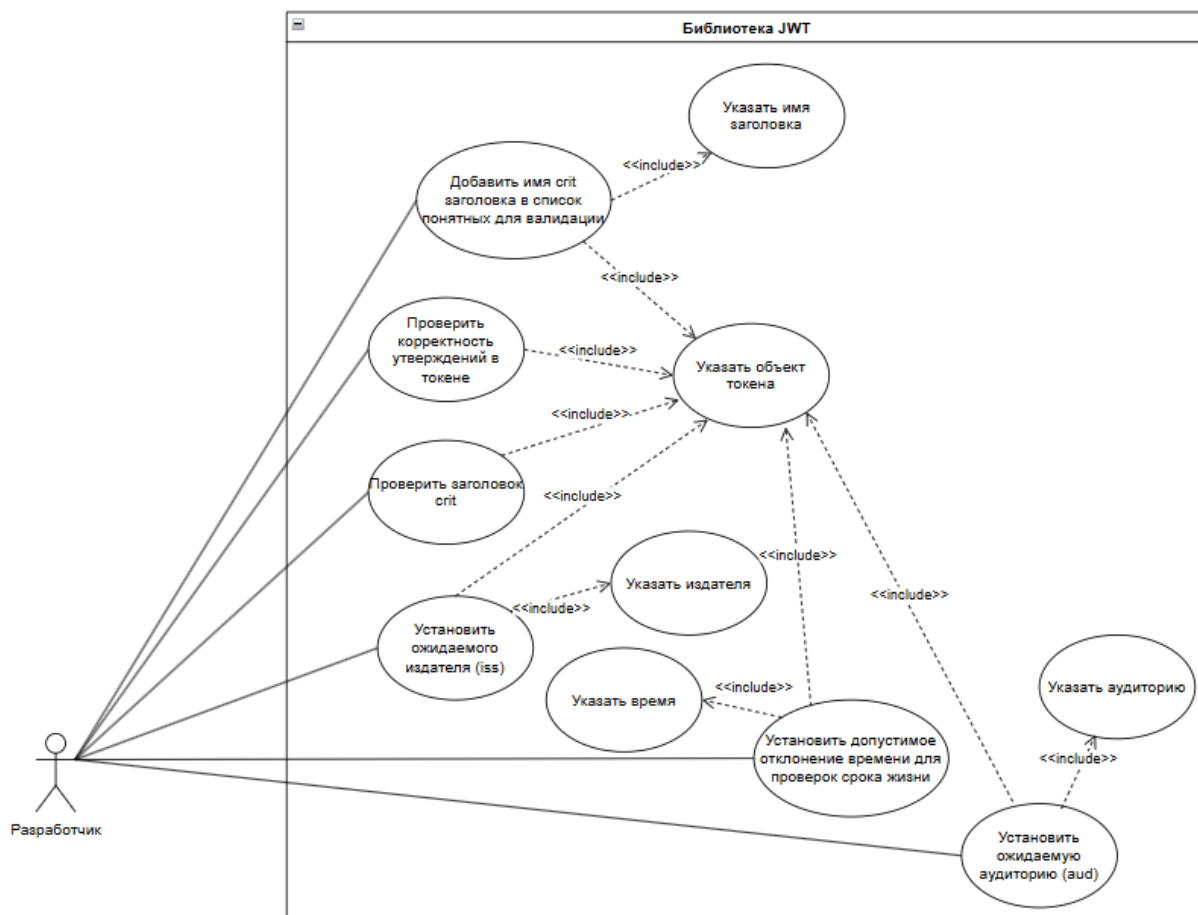


Рисунок 6 – Диаграмма вариантов использования библиотеки для расширенной проверки утверждений и заголовка

На данном рисунке представлены варианты расширенной проверки JWT. В первую очередь это валидация стандартных утверждений «nbf», «iat», «exp», а также «iss» и «aud» при соответствующих настройках, помимо этого определен вариант настройки допустимой задержки, которая учитывается при валидации временных ограничений действия токена. Кроме того, определен вариант проверки критических заголовков, которые также можно указать отдельной функцией.

На рисунке 7 представлена диаграмма вариантов использования библиотеки для работы с утверждениями и заголовком.

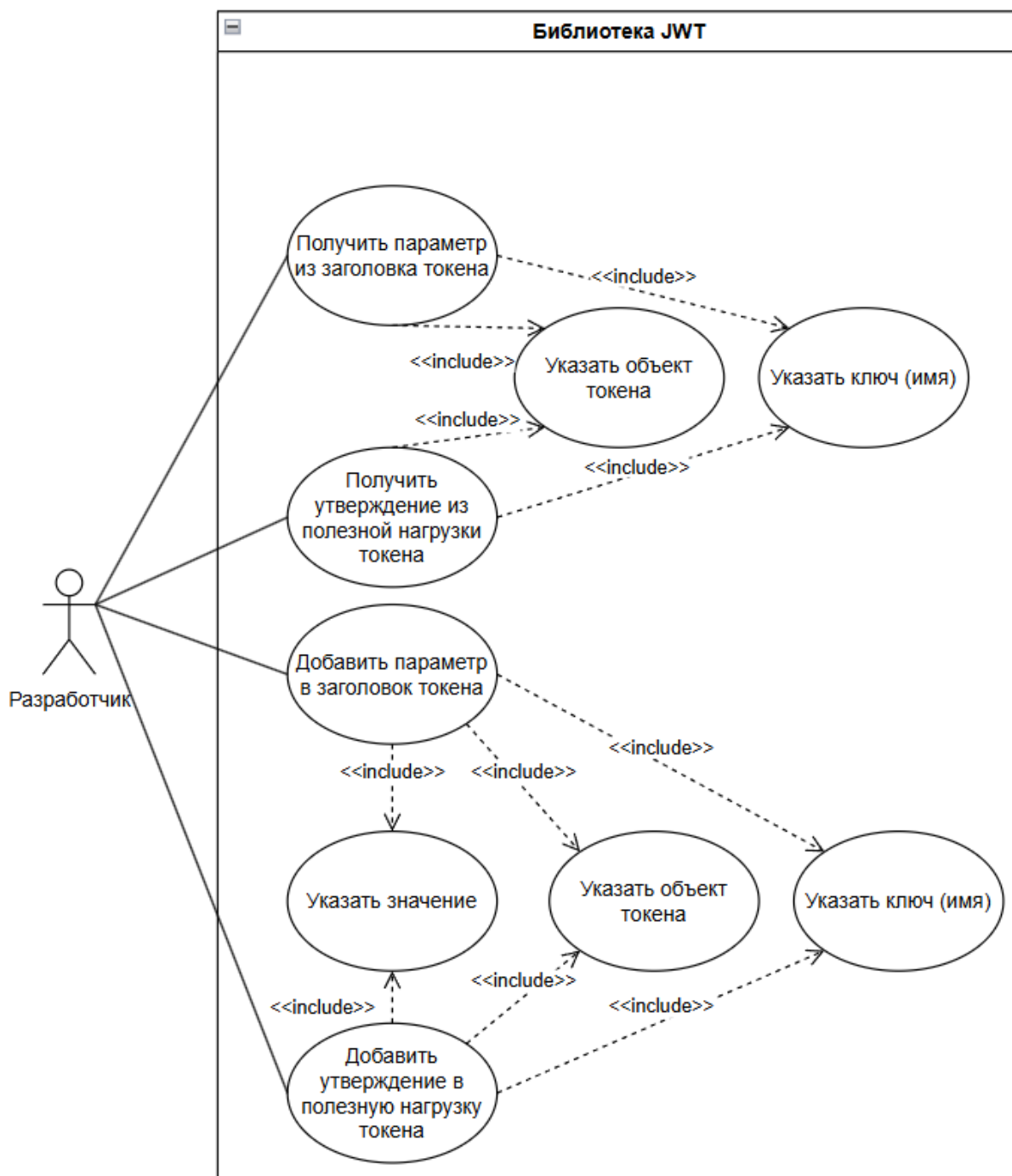


Рисунок 7 – Диаграмма вариантов использования библиотеки для работы с утверждениями и заголовком

На данном рисунке изображены варианты использования для работы с утверждениями и заголовком. В частности, добавления и получения параметров заголовка и утверждений.

На рисунке 8 представлена диаграмма вариантов использования библиотеки для работы с криптографическими ключами.

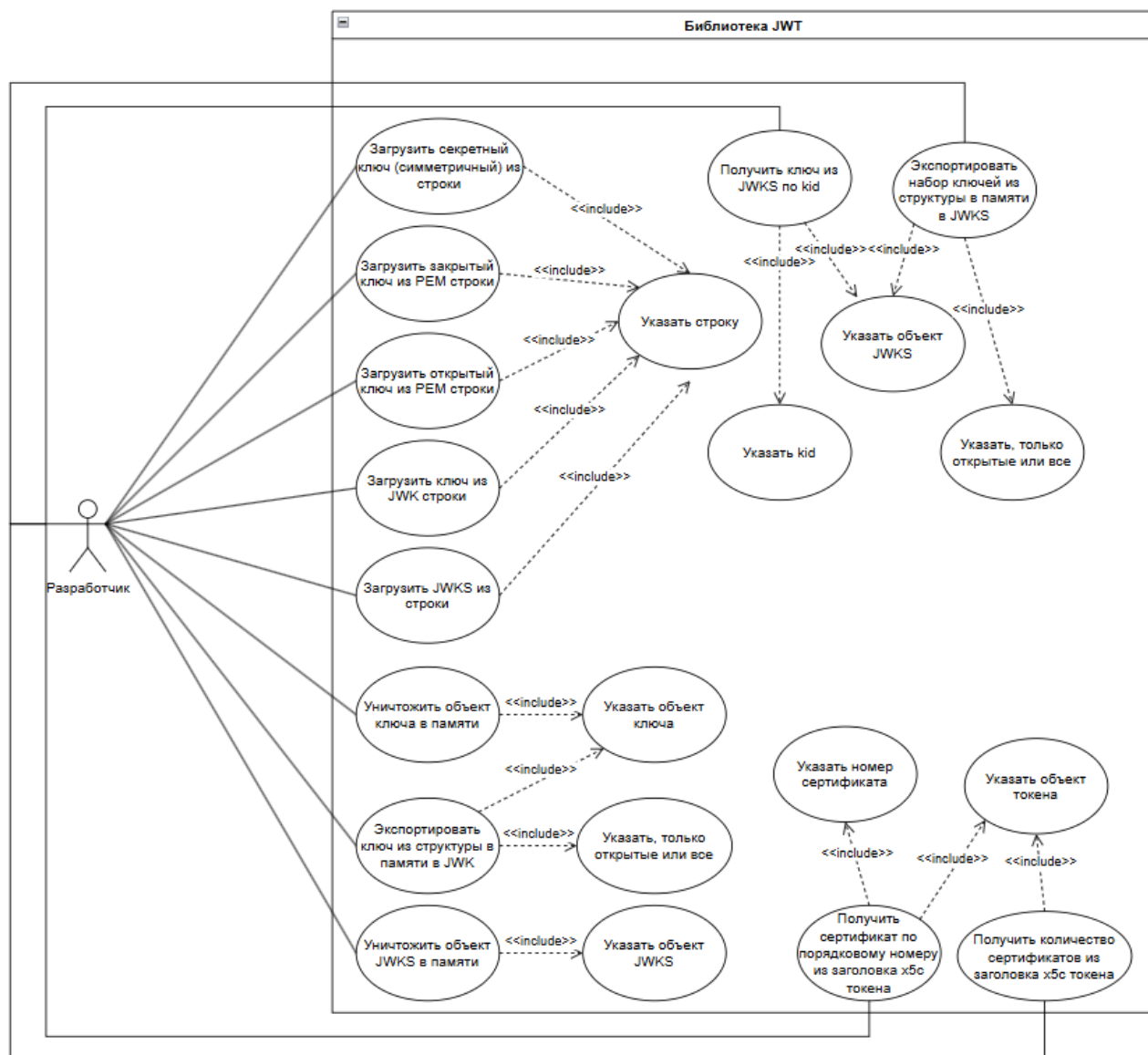


Рисунок 8 – Диаграмма вариантов использования библиотеки для работы с криптографическими ключами

На данном рисунке представлены варианты использования, касающиеся операций с ключевой информацией. Во-первых, это варианты загрузки ключевой информации из разных источников. Во-вторых, это варианты экспорта ключа и набора ключей в JWK и JWKS соответственно. В-третьих, это варианты работы с сертификатами в параметре заголовка «x5c». А также другие вспомогательные варианты использования, например удаления объекта ключа.

В результате анализа функциональных требований были составлены диаграммы вариантов использования библиотеки с их описанием.

2.2 Требования к архитектуре программного обеспечения

По итогам анализа предметной области было выделено несколько требований к архитектуре разрабатываемой библиотеки:

- модульность должна быть высокой, функции библиотеки отвечают только за одно конкретное действие;
- интерфейс должен быть изолированным и независимым от внутренней реализации, таким образом повысится стабильность API;
- должна быть обеспечена расширяемость библиотеки, чтобы добавление нового функционала проходило легко;
- должна быть обеспечена безопасность реализации библиотеки с помощью дополнительных проверок и других мер;
- одним из главных требований является конфигурируемость, пользователь должен иметь возможность управлять поведением, умолчания должны быть безопасными, но возможность тонкой настройки – обязательна;
- обеспечение платформенной независимости тоже является важным аспектом, чем больше платформ будет поддерживать библиотека – тем лучше.

Такие архитектурные требования были выделены в рамках разработки.

2.3 Требования к интеграции и использованию

Одним из самых важных факторов разработки библиотеки является простота её использования. Сложности с развертыванием и последующим использованием отталкивают разработчиков и вынуждают их искать другие реализации, даже если они уступают по возможностям. Поэтому крайне важно обеспечить низкий порог входа, и он обеспечивается несколькими факторами.

Во-первых, API должен быть декларативным. Названия функций, структур и классов должны «говорить за себя», то есть разработчик, глядя на функцию должен примерно понимать, что она делает и как её использовать. Здесь же

весьма уместно смотреть пояснительные комментарии в коде. Реализация этих двух элементов уже делает использование библиотеки намного проще.

Во-вторых, артефакты библиотеки должны соблюдать тот же принцип модульности, что и интерфейс. То есть если библиотека состоит из более чем одного dll/so файла, то разделение должно быть по функциональному признаку и конфигурируемо, а в остальных случаях одного библиотечного бинарного артефакта вполне достаточно. Таким образом можно избежать путаницы и необходимости следить за перечнем библиотечных файлов, которые друг от друга зависят.

В-третьих, как говорилось ранее, библиотека должна иметь минимум зависимостей. С точки зрения использования поиск и подбор библиотек для функционирования другой библиотеки – это всегда неудобно и не привлекательно.

2.4 Обоснование выбора технологий и средств разработки

Исходя из требований, описанных выше, был выбран перечень технологий для разработки.

В качестве языка программирования библиотеки будет использоваться язык C для объявления интерфейса и язык C++ для внутренней реализации интерфейса. C – это язык, который массово используется в разработке низкоуровневого программного обеспечения, а также средств криптографической защиты. Application Binary Interface (ABI) языка C стабилен и поддерживается большинством языков программирования и платформ, то есть библиотеку можно будет портировать, например, на язык Python, и добиться поддержки достаточно специфичных операционных систем, а значит может иметь большой охват аудитории. C++ – мощный язык программирования, он был выбран для внутренней реализации так как совместим с C и имеет множество удобств с точки зрения написания кода. Данный язык представляет удобные структуры данных, алгоритмы, поддерживает ООП и другие современные конструкции, которых нет в чистом C, а значит разработка будет

вестись гораздо быстрее. Такой архитектурный подход весьма распространен, одним из примеров является библиотека ZeroMQ [23], которая пользуется огромной популярностью на разных платформах.

Существует множество систем сборки для языков C и C++, такие как meson или bazel, однако в качестве системы сборки будет использоваться Cmake. Данное программное средство позволяет создать сборочное окружение для разных платформ. Таким образом, с помощью Cmake для операционной системы Windows будет создаваться проект IDE Visual Studio, а для Linux генерироваться файлы make. Данный инструмент позволяет значительно облегчить сборки систем под разные платформы.

Реализация будет иметь несколько зависимостей:

- библиотеку nlohmann_json, для работы с JSON на языке C++ [24];
- библиотеку GTest (Google Test) для тестирования [25];
- библиотеку OpenSSL для работы с криптографией [26].

Первые две будут линковаться статически, то есть, как часть библиотеки, пользователю не нужно будет самостоятельно обеспечивать их наличие. GTest вовсе используется только для внутренних целей тестирования, артефакты которого не публикуются.

С другой стороны, OpenSSL линкуется динамически. Это вызвано несколькими факторами. Во-первых, OpenSSL это активно развивающийся проект, часто получающий обновления. Сделав линковку динамической, мы освобождаем себя от необходимости перекомпилировать наше решение из-за обновлений. Во-вторых, размер исполняемого файла получается намного меньше. OpenSSL – это огромная библиотека, разработка которой началась ещё в 90-ых годах прошлого века, в ней сосредоточено много функционала, в том числе устаревшего, для обратной совместимости. В-третьих, OpenSSL очень популярна, как зависимость. Даже язык программирования Python в качестве одной из своих зависимостей имеет динамическую библиотеку OpenSSL, находящейся прямо в папке языка. Такая практика весьма распространена.

2.5 Требования к аппаратно-программному обеспечению

Для аппаратно-программного обеспечения также были выявлены требования.

Библиотека должна поддерживать операционные системы Windows и Linux в полном составе функционала. Так как C и C++ после сборки имеют разные исполняемые файлы для разных систем, необходимо обеспечить отдельную компиляцию для каждой.

3 Проектирование программных средств

3.1 Проектирование внешнего API

Главная цель проектирования внешнего API является предоставление простого, понятного, конфигурируемого и потенциально расширяемого интерфейса для разработчиков. Однако необходимо не забывать о других целях, таких как безопасность и инкапсуляция.

Основные структуры данных, будут оформлены в виде непрозрачных хэндлов (дескрипторов). С точки зрения разработчика, знание их внутреннего устройства не нужно, особенно если библиотека и так способна ответить на все его вопросы, но с точки зрения инкапсуляции и безопасности это дает множество преимуществ, а именно:

- сокрытие реализации внутренней логики, избыточной для пользователя;
- постоянство интерфейса с пользовательской точки зрения, внутренняя реализация может меняться, но использование будет таким же, как и раньше;
- гибкость и совместимость интерфейса, сигнатура остается такой же, даже при значительных внутренних изменениях;
- явное разграничение возможностей пользователей на допустимое и недопустимое;
- предотвращение несанкционированного доступа и изменения информации.

Всего выделено три таких структуры, первая отвечающая за объект JWT, вторая – за объект криптографического ключа, третья – за объект набора ключей JWKS.

Настройка объектов будет происходить с помощью builder-подобных конструкций. Это такой стиль проектирования, при котором объект поэтапно настраивается через цепочку вызовов, прежде чем он будет использован. В объектно-ориентированном программировании этот паттерн называется Builder.

Например, имеется объект JWT. С помощью такой настройки, сначала будет указан выбранный алгоритм, затем добавлены необходимые утверждения, затем создана подпись, всё в отдельных шагах.

Есть другой способ настройки, через универсальные функции, способные делать множество операций сразу и в качестве параметров принимающих целые структуры с множеством полей настроек.

Данный способ имеет свои преимущества. Во-первых, простота вызова – одна функция на решение всех проблем, без нужды пользоваться целым перечнем функций. Во-вторых, компактность. В-третьих, удобство использования – не нужно знать, что и в каком порядке вызывать, так как внутренняя реализация позаботится обо всё сама.

Однако у этого стиля есть и минусы. Во-первых, плохая читаемость – чем больше настраиваемых параметров, тем проще запутаться или что-то не учесть. Во-вторых, проблемы с расширяемостью – в случае добавления нового параметра или поля в структуру пропадает обратная совместимость, крайне неприятная проблема. В-третьих, низкая гибкость – далеко не всегда есть необходимость использовать все параметры, а значит приходится добавлять пустые значения и «заглушки». В-четвертых, наносится серьезный урон по конфигурируемости, так как внутренняя реализация функции может не удовлетворять требованиям по процессу операций, например, может иметь избыточные проверки, которые не нужны в некоторых сценариях. Кроме того, может усложниться вызов из других языков программирования и платформ. Все эти минусы являются критическими и полностью противоречат целям, поставленным ранее.

Информацию об успехе проведения операции пользователь будет получать с помощью специальных кодов, возвращаемых из функций. Код представляет числовое значение, расшифровку которого можно получить с помощью специальной функции. Количество кодов должно быть большим, чтобы можно было давать наиболее точную причину ошибки. Помимо кодов ошибок будет создан код успешного завершения в виде числовой единицы или нуля.

3.2 Проектирование внутренней реализации API

Внутренняя реализация будет сделана на языке C++, соответственно в разработке открывается много возможностей. Во-первых, доступно полноценное объектно-ориентированное программирование, а значит можно эффективнее переиспользовать код и визуально делить функционал. Однако C++ – это мультипарадигменный язык, не обязывающий составлять код только из классов и его объектов, как, например, Java. Соответственно, можно комбинировать стили в зависимости от ситуации.

В проектировании внешнего API упоминалось, что объекты представлены дескрипторами, то есть указателями на внутренние структуры, скрытые от пользователей. Внутренняя реализация этих структур представлена на диаграмме классов на рисунке 9.

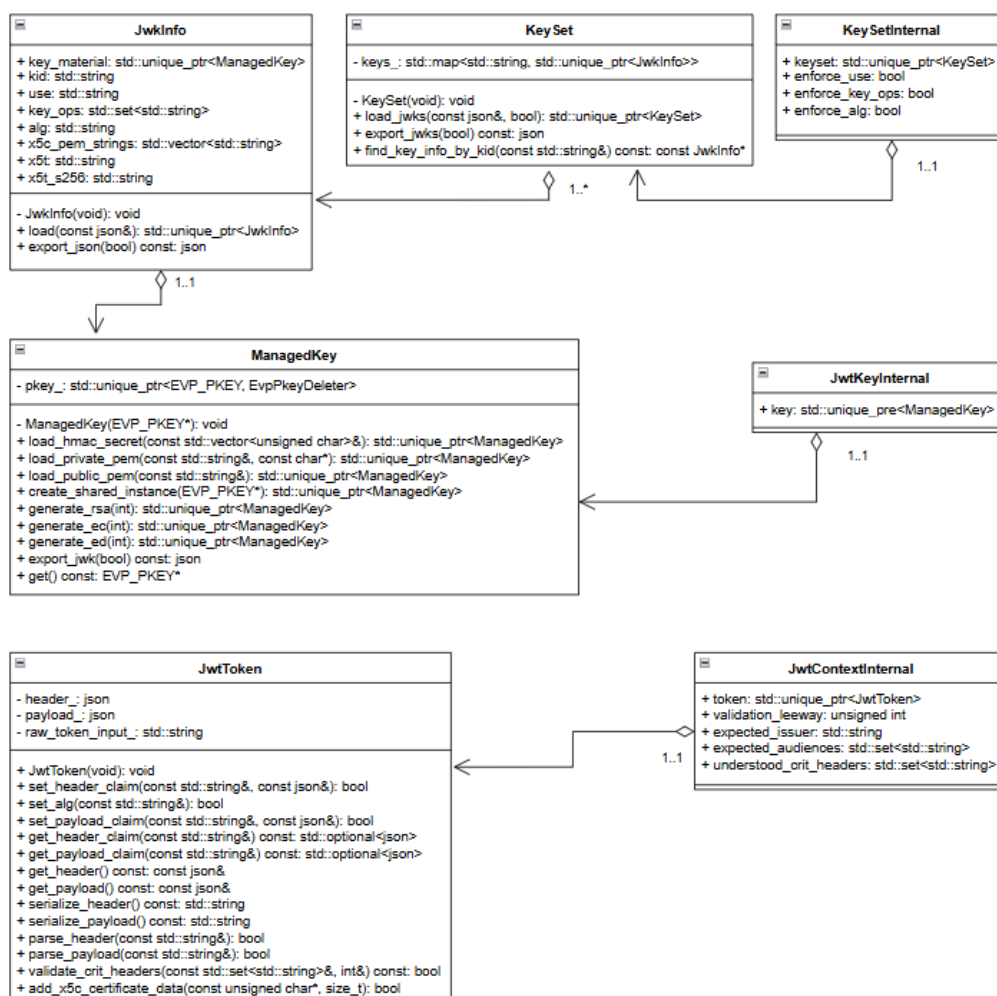


Рисунок 9 – Диаграмма классов структур

На диаграмме классов «JwtContextInternal» – это структура JWT в оперативной памяти компьютера, «JwtKeyInternal» – это структура криптографического ключа, которая может представлять как открытый, так и закрытый ключ в оперативной памяти. «KeySetInternal» – это структура набора ключей JWKS, загруженная в оперативную память. Эти структуры в свою очередь включают в себя объекты других классов, описанных далее в соответствующих спецификациях.

Ниже, на рисунках 10 – 12 представлены спецификации структур «JwtContextInternal», «JwtKeyInternal», «KeySetInternal», их наименование, назначение, поля класса.

Наименование класса: JwtContextInternal
Назначение класса: реализация структуры JWT, дескриптором которого владеет пользователь
Поля класса: <ul style="list-style-type: none"> – token: std::unique_ptr<JwtToken> – объект JwtToken; – validation_leeway: unsigned int – допустимая временная задержка при валидации стандартных утверждений токена; – expected_issuer: std::string – ожидаемый издатель токена; – expected_audiences: std::set<std::string> – ожидаемая аудитория токена; – understood_crit_headers: std::set<std::string> – критические заголовки, понятные приложению.

Рисунок 10 – Спецификация структуры «JwtContextInternal»

Наименование класса: JwtKeyInternal
Назначение класса: реализация структуры криптографического ключа, дескриптором которого владеет пользователь
Поля класса: <ul style="list-style-type: none"> – key: std::unique_ptr<ManagedKey> – объект ManagedKey.

Рисунок 11 – Спецификация структуры «JwtKeyInternal»

Наименование класса: KeySetInternal
Назначение класса: реализация структуры набора ключей JWKS, дескриптором которого владеет пользователь
Поля класса: <ul style="list-style-type: none"> – keyset: std::unique_ptr<KeySet> – объект KeySet; – enforce_use: bool – опция обязательного присутствия поля «use» у JWK; – enforce_key_ops: bool – опция обязательного присутствия поля «key_ops» у JWK; – enforce_alg: bool – опция обязательного присутствия поля «alg».

Рисунок 12 – Спецификация структуры «KeySetInternal»

На рисунках 13 – 15 представлены спецификации классов «JwtToken», «KeySet», «ManagedKey» объекты которых представлены в структурах.

Наименование класса: JwtToken
Назначение класса: управление JSON составляющей токена
Поля: <ul style="list-style-type: none"> – header_: json – JSON объект заголовка токена; – payload_: json – JSON объект полезной нагрузки токена; – raw_token_input_: std::string – исходная строка токена.
Методы: <ul style="list-style-type: none"> – set_header_claim(const std::string&, const json&): bool – установка утверждения в заголовок; – set_alg(const std::string&): bool – установка утверждения «alg»; – set_payload_claim(const std::string&, const json&): bool – установка утверждения в полезную нагрузку; – get_header_claim(const std::string&) const: std::optional<json> – получение утверждения заголовка; – get_payload_claim(const std::string&) const: std::optional<json> – получение утверждения полезной нагрузки; – get_header() const: const json& – получение структуры JSON заголовка; – get_payload() const: const json& – получение структуры JSON полезной нагрузки; – get_alg() const: std::optional<std::string> – получение значения поля «alg»; – serialize_header() const: std::string – сериализация заголовка; – serialize_payload() const: std::string – сериализация полезной нагрузки; – parse_header(const std::string&): bool – парсинг заголовка; – parse_payload(const std::string&): bool – парсинг полезной нагрузки; – validate_crit_headers(const std::set<std::string>&, int&): bool – проверка заголовка «crit»; – add_x5c_certificate_data(const unsigned char*, size_t): bool – добавление сертификата X.509 в заголовок «x5c».

Рисунок 13 – Спецификация класса «JwtToken»

Наименование класса: KeySet
Назначение класса: управление набором ключей JWKS
Поля: <ul style="list-style-type: none"> – keys_: std::map<std::string, std::unique_ptr<JwkInfo>> – хранилище ключей JWK по их kid.
Методы: <ul style="list-style-type: none"> – load_jwks(const json&, bool): std::unique_ptr<KeySet> – загрузка набора ключей из JWKS; – export_jwks(bool) const: json – экспорт ключей в формат JWKS; – find_key_info_by_kid(const std::string&) const: const JwkInfo* – поиск ключа в наборе по kid.

Рисунок 14 – Спецификация класса «KeySet»

Наименование класса: ManagedKey
Назначение класса: управление криптографическим ключом
Поля: <ul style="list-style-type: none"> – pkey_: EVP_PKEY* – структура ключа OpenSSL;
Методы: <ul style="list-style-type: none"> – load_hmac_secret(const std::vector<unsigned char>&): std::unique_ptr<ManagedKey> – создание ключа для HMAC из секрета; – load_private_pem(const std::string&, const char*): std::unique_ptr<ManagedKey> – загрузка закрытого ключа из PEM строки; – load_public_pem(const std::string&): std::unique_ptr<ManagedKey> – загрузка открытого ключа из PEM строки; – create_shared_instance(EVP_PKEY*): std::unique_ptr<ManagedKey> – создание нового экземпляра ключа из существующего; – generate_rsa(int): std::unique_ptr<ManagedKey> – генерация пары ключей RSA; – generate_ec(int): std::unique_ptr<ManagedKey> – генерация пары ключей ECDSA; – generate_ed(int): std::unique_ptr<ManagedKey> – генерация пары ключей EdDSA; – export_jwk(bool) const: json – экспорт ключа в формат JWK; – get() const: EVP_PKEY* – получение ключа.

Рисунок 15 – Спецификация класса «ManagedKey»

На рисунке 16 представлена спецификация класса «JwkInfo».

Наименование класса: JwkInfo
Назначение класса: управление JWK
Поля: <ul style="list-style-type: none"> – key_material: std::unique_ptr<ManagedKey> – криптографический ключ; – kid: std::string – Key ID; – use: std::string – Public Key Use; – key_ops: std::set<std::string> – Key Operations; – alg: std::string – Algorithm; – x5c_pem_strings: std::vector<std::string> – цепь сертификатов из заголовка «x5c»; – x5t: std::string – отпечаток сертификата SHA-1; – x5t_s256: std::string – отпечаток сертификата SHA-256.
Методы: <ul style="list-style-type: none"> – load(const json&): std::unique_ptr<JwkInfo> – загружает ключ и его метаданные из JWK; – export_json(bool) const: json – экспорт ключа и его метаданных в формат JWK.

Рисунок 16 – Спецификация класса «JwkInfo»

С функциями всё намного проще. Объявления в API – это прототипы, имеющие реализацию в объектных файлах. И, учитывая, что реализация функций сделана на C++, можно пользоваться высокоуровневыми конструкциями, например исключениями и другими возможностями богатой библиотеки языка, и дополненным функционалом используемой JSON библиотеки. В рамках реализации функций API, будет разработано множество вспомогательных функций, которые останутся только внутри, например функции кодирования и декодирования в Base64URL, функции подписи и шифрования для конкретных алгоритмов, функции работы с сертификатами и другие.

Таким образом была спроектирована внутренняя реализация программного интерфейса. Такой план является приемлемым для начальных этапов реализации, в будущем количество классов может увеличиться или

текущие могут дополниться. В любом случае имеется понятная структура, которую не сложно поддерживать и дополнять.

4 Реализация программных средств

4.1 Разработка функций криптографических подписи и проверки

Согласно соответствующей диаграмме вариантов использования, библиотека должна предоставлять возможности создать объект токена, уничтожить объект токена, подписать токен и криптографически проверить токен. Последнее может быть выполнено помощью открытого ключа в трех его формах:

- в форме PEM строки;
- в форме JWK из набора ключей JWKS по его «kid»;
- в форме его извлечения из последнего сертификата X.509 цепочки в параметре заголовка «x5c».

Все функции были разработаны. Требования к ним, как функциональные, так и архитектурные были выполнены. Подробные спецификации данных функций представлены в приложении Б.

4.2 Разработка функций работы с утверждениями (claims) и заголовком (header)

Согласно соответствующей диаграмме вариантов использования, библиотека должна предоставлять возможности добавлять и получать параметры из заголовка, а также добавлять и получать утверждения из полезной нагрузки токена.

Было решено сделать для каждой из этих четырех групп набор из пяти функций, каждая из которых посвящена добавлению или получению значения определенного типа данных. Всего было определено пять типов данных:

- строковый тип данных;
- целое знаковое число;
- знаковое число с плавающей точкой;
- булево значение;
- строка, содержащая в себе валидный JSON объект.

Кроме того, были сделаны более специфичные функции, а именно функция установки алгоритма подписи «alg», функции получения отпечатка сертификата SHA-1 и SHA-256 из параметров «x5t» и «x5t#S256» соответственно.

Все функции были разработаны. Функциональные и архитектурные требования к ним были выполнены. Подробные спецификации данных функций представлены в приложении Б.

4.3 Разработка функций расширенной проверки утверждений и заголовка

Согласно соответствующей диаграмме вариантов использования библиотека должна предоставлять возможности установки ожидаемой аудитории токена «aud», установки допустимого времени отклонения времени для проверок срока жизни токена, установки ожидаемого издателя «iss» и выполнению проверки корректности утверждений в проверяемом токене. Кроме того, должны быть возможности добавить имя критического заголовка в список понятных для валидации, а также выполнить проверку заголовка «crit».

Все эти функции были реализованы. Функциональные и архитектурные требования к ним также были выполнены. Подробные спецификации данных функций представлены в приложении Б.

4.4 Разработка функций для работы с криптографическими ключами

Согласно соответствующей диаграмме вариантов использования библиотека должна предоставлять следующие возможности в плане загрузки ключевой информации:

- загрузка секретного ключа из строки;
- загрузка закрытого ключа из PEM строки;
- загрузка открытого ключа из PEM строки;
- загрузка ключа из строки, представляющей JWK;
- загрузка набора ключей JWKS из строки.

Каждая из этих функций принимает на вход ключ в том или ином виде и создает объект ключа в памяти, который позже выдается для использования.

Кроме этого, были выделены следующие функции экспорта ключей:

- экспорт набора ключей в JWKS строку;
- экспорт ключа в строку JWK.

Обе этих функции принимают на вход существующий объект в памяти и создают строку указанной структуры. Данные функции могут быть полезны, однако функционал работы с JWK и JWKS в целом может быть доработан.

Помимо функций выше библиотека должна предоставлять функции удаления объектов ключей, функцию поиска ключа из JWKS по «kid» и функции получения количества сертификатов и сертификата по индексу из параметра «x5c».

Все функции были разработаны с соблюдением функциональных и архитектурных требований. Также были разработаны ещё несколько функций для JWKS, в частности функции получения параметров «alg», «use», «key_ops» и их принудительной проверки существования в JWKS. Подробные спецификации данных функций представлены в приложении Б.

4.5 Разработка примеров использования библиотеки

Примеры являются важным дополнением к любой библиотеке. Они дают самый наглядный пример использования на практике и зачастую, при наличии таковых, разработчик в первую очередь посмотрит на примеры, нежели в документацию. Поэтому необходимо добавить примеры использования и для этой библиотеки.

Всего было разработано два примера – один на использование функций по работе с параметрами заголовка и утверждениями в полезной нагрузке, а также для демонстрации функций валидации утверждений и заголовка; второй – предназначен преимущественно для демонстрации криптографических возможностей – это большой пример с возможностями использования разных криптографических алгоритмов, разных способов передачи ключевой

информации и разных способов проверки подписи токена. Примеры намеренно были написаны просто и прямолинейно, с наличием дополнительных комментариев. Исходные коды обоих примеров представлены в приложении В.

5 Контроль качества программного обеспечения

5.1 Планирование процесса контроля качества программных средств

Одним из главных требований к библиотеке является надежность. Достигнуть её, или хотя бы приблизиться к этому, можно путем обильного тестирования разработанного решения. Тестирование может быть нескольких видов, но в случае библиотек наиболее популярными являются следующие:

- модульное тестирование;
- интеграционное тестирование;
- тестирование совместимости (регрессионное);
- фаззинг (Fuzz Testing);
- тестирование производительности.

Модульное тестирование заключается в проверке отдельных функций и компонентов библиотеки в изоляции от остальной системы. Оно позволяет убедиться, что каждая функция корректно обрабатывает входные данные, граничные случаи, ошибки и возвращает ожидаемый результат.

Интеграционное тестирование проверяет работу связанного набора компонентов библиотеки как единого целого. Данный вид тестирования позволяет убедиться, что компоненты взаимодействуют корректно, а полные сценарии работают от начала и до конца.

Тестирование совместимости направлено на проверку того, что нововведения в программном коде не влияют на поведение ранее разработанных функций и не нарушают совместимость программного интерфейса. Сохранение обратной совместимости и поведения между версиями её главная цель.

Фаззинг – это автоматическое тестирование с помощью генерации случайных или маловероятных входных данных. Целью данного тестирования является выявление нестандартных ошибок, вроде утечек памяти, падений программы, выходы за границы буфера при определенных условиях и другие.

Тестирование производительности посвящено измерению времени выполнения, использованию памяти, пропускной способности функций. С помощью такого тестирования можно определить узкие места для последующей оптимизации при необходимости.

Был сделан вывод, что самыми необходимыми тестами являются модульные и интеграционные. Тестирование совместимости не совсем актуально на данный момент, так как разработанная библиотека пока не получала обновлений. Фаззинг является весьма полезным, особенно в контексте данной библиотеки, так как её потребителями с большой вероятностью будут серверные приложения, создающие и верифицирующие JWT круглые сутки, однако сам процесс подготовки и проведения фаззинг-тестирования является очень трудоемким. То же самое касается тестирования производительности – потенциально полезное тестирование, однако трудоемкое для организации и на данном этапе не особо нужное.

Также будет осуществлена оценка качества программного кода. Она включает в себя различные метрики, позволяющие количественно измерить характеристики исходного кода: его сложность, читаемость, сопровождаемость, потенциальную надежность и даже стоимость поддержки. Это особенно важно при разработке программного обеспечения, где качество и надежность критичны.

Для оценки качества разработанного программного кода были выбраны наиболее популярные метрики кода, такие как метрика Холстеда и метрика Чепина.

Метрика Холстеда – это количественный способ оценки сложности исходного кода, основанный на анализе количества уникальных и общих операторов и операндов в программе. Она позволяет вычислить такие показатели, как объем, сложность и предполагаемое усилие для понимания или реализации программного фрагмента.

Метрика Чепина – это способ оценки сложности программы на основе анализа ролей переменных, классифицируемых как входные, управляющие,

изменяемые и временные. Она позволяет судить о структурной сложности кода с учетом его логической организации и семантики.

5.2 Тестирование программных средств

5.2.1 Модульное тестирование

В результате разработки программы были разработаны модульные тесты. Большая часть из них направлены на создание и валидацию JWT и может быть поделена на следующие группы:

- создание и валидация JWT с использованием закрытого и открытого ключей из PEM строк дополненные работой с утверждениями и заголовком – «SignVerifyTest»;
- создание и валидация JWT с использованием закрытого ключа из PEM строки и открытого ключа, загруженного из строки JWK – «SignVerifyPubToJWK»;
- создание и валидация JWT с использованием закрытого ключа из PEM строки и открытого ключа, полученного из соответствующего сертификата X.509 – «SignVerify_x5cCert»;
- создание и валидация JWT с использованием закрытого ключа из PEM строки и открытого ключа, полученного из набора ключей JWK (JWK Set) – «SignPEM_VerifyJWKS»;
- проверка JWT, созданных сторонними сервисами, с помощью открытых ключей из PEM строк, полученных от того же сервиса – ExternalVerify.

Каждая из групп тестирует сценарии со всеми поддерживаемыми алгоритмами (HS256/384/512, RSA256/384/512, ES256/256K/384/512, EdDSA), кроме последней группы в которой отсутствуют алгоритмы ES256K и EdDSA. Это связано с тем, что выбранный сервис <https://jwt.io> не поддерживает данные алгоритмы.

Помимо выделенных групп тестов был создан дополнительный тест, проверяющий загрузку и экспорт открытых и закрытых ключей в JWK –

«JWK_LoadAndExport». Результаты всех созданных тестов представлены на рисунках 17 – 22.

```

-----] 15 tests from SignVerifyTest
RUN    ] SignVerifyTest.HS256
      OK ] SignVerifyTest.HS256 (61 ms)
RUN    ] SignVerifyTest.HS384
      OK ] SignVerifyTest.HS384 (6 ms)
RUN    ] SignVerifyTest.HS512
      OK ] SignVerifyTest.HS512 (4 ms)
RUN    ] SignVerifyTest.RS256
      OK ] SignVerifyTest.RS256 (14 ms)
RUN    ] SignVerifyTest.RS384
      OK ] SignVerifyTest.RS384 (9 ms)
RUN    ] SignVerifyTest.RS512
      OK ] SignVerifyTest.RS512 (6 ms)
RUN    ] SignVerifyTest.PS256
      OK ] SignVerifyTest.PS256 (5 ms)
RUN    ] SignVerifyTest.PS384
      OK ] SignVerifyTest.PS384 (8 ms)
RUN    ] SignVerifyTest.PS512
      OK ] SignVerifyTest.PS512 (9 ms)
RUN    ] SignVerifyTest.ES256
      OK ] SignVerifyTest.ES256 (10 ms)
RUN    ] SignVerifyTest.ES384
      OK ] SignVerifyTest.ES384 (7 ms)
RUN    ] SignVerifyTest.ES512
      OK ] SignVerifyTest.ES512 (12 ms)
RUN    ] SignVerifyTest.EdDSA_25519
      OK ] SignVerifyTest.EdDSA_25519 (6 ms)
RUN    ] SignVerifyTest.EdDSA_448
      OK ] SignVerifyTest.EdDSA_448 (9 ms)
RUN    ] SignVerifyTest.ES256K
      OK ] SignVerifyTest.ES256K (9 ms)
-----] 15 tests from SignVerifyTest (197 ms total)

```

Рисунок 17 – Результаты выполнения группы тестов «SignVerifyTest»

```

-----] 12 tests from SignPEM_VerifyJWKS
RUN    ] SignPEM_VerifyJWKS.RS256
      OK ] SignPEM_VerifyJWKS.RS256 (10 ms)
RUN    ] SignPEM_VerifyJWKS.RS384
      OK ] SignPEM_VerifyJWKS.RS384 (10 ms)
RUN    ] SignPEM_VerifyJWKS.RS512
      OK ] SignPEM_VerifyJWKS.RS512 (10 ms)
RUN    ] SignPEM_VerifyJWKS.PS256
      OK ] SignPEM_VerifyJWKS.PS256 (10 ms)
RUN    ] SignPEM_VerifyJWKS.PS384
      OK ] SignPEM_VerifyJWKS.PS384 (10 ms)
RUN    ] SignPEM_VerifyJWKS.PS512
      OK ] SignPEM_VerifyJWKS.PS512 (11 ms)
RUN    ] SignPEM_VerifyJWKS.ES256
      OK ] SignPEM_VerifyJWKS.ES256 (7 ms)
RUN    ] SignPEM_VerifyJWKS.ES384
      OK ] SignPEM_VerifyJWKS.ES384 (13 ms)
RUN    ] SignPEM_VerifyJWKS.ES512
      OK ] SignPEM_VerifyJWKS.ES512 (20 ms)
RUN    ] SignPEM_VerifyJWKS.EdDSA_25519
      OK ] SignPEM_VerifyJWKS.EdDSA_25519 (8 ms)
RUN    ] SignPEM_VerifyJWKS.EdDSA_448
      OK ] SignPEM_VerifyJWKS.EdDSA_448 (10 ms)
RUN    ] SignPEM_VerifyJWKS.ES256K
      OK ] SignPEM_VerifyJWKS.ES256K (11 ms)
-----] 12 tests from SignPEM_VerifyJWKS (166 ms total)

```

Рисунок 18 – Результаты выполнения группы тестов «SignPEM_VerifyJWKS»

```

-----] 12 tests from SignVerify_x5cCert
RUN    ] SignVerify_x5cCert.RS256
      OK ] SignVerify_x5cCert.RS256 (19 ms)
RUN    ] SignVerify_x5cCert.RS384
      OK ] SignVerify_x5cCert.RS384 (11 ms)
RUN    ] SignVerify_x5cCert.RS512
      OK ] SignVerify_x5cCert.RS512 (11 ms)
RUN    ] SignVerify_x5cCert.PS256
      OK ] SignVerify_x5cCert.PS256 (12 ms)
RUN    ] SignVerify_x5cCert.PS384
      OK ] SignVerify_x5cCert.PS384 (11 ms)
RUN    ] SignVerify_x5cCert.PS512
      OK ] SignVerify_x5cCert.PS512 (12 ms)
RUN    ] SignVerify_x5cCert.ES256
      OK ] SignVerify_x5cCert.ES256 (6 ms)
RUN    ] SignVerify_x5cCert.ES384
      OK ] SignVerify_x5cCert.ES384 (12 ms)
RUN    ] SignVerify_x5cCert.ES512
      OK ] SignVerify_x5cCert.ES512 (18 ms)
RUN    ] SignVerify_x5cCert.EdDSA_25519
      OK ] SignVerify_x5cCert.EdDSA_25519 (6 ms)
RUN    ] SignVerify_x5cCert.EdDSA_448
      OK ] SignVerify_x5cCert.EdDSA_448 (8 ms)
RUN    ] SignVerify_x5cCert.ES256K
      OK ] SignVerify_x5cCert.ES256K (6 ms)
-----] 12 tests from SignVerify_x5cCert (169 ms total)

```

Рисунок 19 – Результаты выполнения группы тестов «SignVerify_x5cCert»

```

-----] 12 tests from SignVerifyPubToJWK
[ RUN      ] SignVerifyPubToJWK.RS256
[ OK       ] SignVerifyPubToJWK.RS256 (11 ms)
[ RUN      ] SignVerifyPubToJWK.RS384
[ OK       ] SignVerifyPubToJWK.RS384 (14 ms)
[ RUN      ] SignVerifyPubToJWK.RS512
[ OK       ] SignVerifyPubToJWK.RS512 (15 ms)
[ RUN      ] SignVerifyPubToJWK.PS256
[ OK       ] SignVerifyPubToJWK.PS256 (14 ms)
[ RUN      ] SignVerifyPubToJWK.PS384
[ OK       ] SignVerifyPubToJWK.PS384 (12 ms)
[ RUN      ] SignVerifyPubToJWK.PS512
[ OK       ] SignVerifyPubToJWK.PS512 (15 ms)
[ RUN      ] SignVerifyPubToJWK.ES256
[ OK       ] SignVerifyPubToJWK.ES256 (12 ms)
[ RUN      ] SignVerifyPubToJWK.ES384
[ OK       ] SignVerifyPubToJWK.ES384 (10 ms)
[ RUN      ] SignVerifyPubToJWK.ES512
[ OK       ] SignVerifyPubToJWK.ES512 (23 ms)
[ RUN      ] SignVerifyPubToJWK.EdDSA_25519
[ OK       ] SignVerifyPubToJWK.EdDSA_25519 (9 ms)
[ RUN      ] SignVerifyPubToJWK.EdDSA_448
[ OK       ] SignVerifyPubToJWK.EdDSA_448 (14 ms)
[ RUN      ] SignVerifyPubToJWK.ES256K
[ OK       ] SignVerifyPubToJWK.ES256K (15 ms)
-----] 12 tests from SignVerifyPubToJWK (196 ms total)

```

Рисунок 20 – Результаты выполнения группы тестов «SignVerifyPubToJWK»

```

-----] 12 tests from ExternalVerify
[ RUN      ] ExternalVerify.HS256
[ OK       ] ExternalVerify.HS256 (2 ms)
[ RUN      ] ExternalVerify.HS384
[ OK       ] ExternalVerify.HS384 (2 ms)
[ RUN      ] ExternalVerify.HS512
[ OK       ] ExternalVerify.HS512 (2 ms)
[ RUN      ] ExternalVerify.RS256
[ OK       ] ExternalVerify.RS256 (3 ms)
[ RUN      ] ExternalVerify.RS384
[ OK       ] ExternalVerify.RS384 (3 ms)
[ RUN      ] ExternalVerify.RS512
[ OK       ] ExternalVerify.RS512 (3 ms)
[ RUN      ] ExternalVerify.PS256
[ OK       ] ExternalVerify.PS256 (3 ms)
[ RUN      ] ExternalVerify.PS384
[ OK       ] ExternalVerify.PS384 (3 ms)
[ RUN      ] ExternalVerify.PS512
[ OK       ] ExternalVerify.PS512 (3 ms)
[ RUN      ] ExternalVerify.ES256
[ OK       ] ExternalVerify.ES256 (3 ms)
[ RUN      ] ExternalVerify.ES384
[ OK       ] ExternalVerify.ES384 (5 ms)
[ RUN      ] ExternalVerify.ES512
[ OK       ] ExternalVerify.ES512 (8 ms)
-----] 12 tests from ExternalVerify (75 ms total)

```

Рисунок 21 – Результаты выполнения группы тестов «ExternalVerify»

```

-----] 1 test from JWK
[ RUN      ] JWK.LoadAndExport
[ OK       ] JWK.LoadAndExport (19 ms)
-----] 1 test from JWK (22 ms total)

```

Рисунок 22 – Результаты выполнения теста «JWK_LoadAndExport»

Все созданные тесты были выполнены и подтвердили корректность работы функций библиотеки.

5.2.2 Интеграционное тестирование

Было осуществлено интеграционное тестирование разработанного программного обеспечения. Для данной библиотеки оно заключается в том, как разработанные функции работают друг с другом. Хорошим вариантом для таких

Таблица 4 – Операторы функции «sign_ecdsa»

Операторы	Число вхождений
!	4
()	23
.get()	4
.data()	3
.size()	2
.resize()	1
	1
!=	2
=	8
==	4
&&	1
<=	5
if	13
return	10
else	3
*	1
&	3
n1 = 17	N1 = 88

Операнды функции «sign_ecdsa» и число их вхождений представлены в таблице 5.

Таблица 5 – Операнды функции «sign_ecdsa»

Операнды	Число вхождений
algorithm	4
md_type	2
key	1
data_to_sign	2
std::nullopt	9
pkey	6
EVP_PKEY_EC	1
expected_nid	7

Продолжение таблицы 5

Операнды	Число вхождений
NID_undef	2
NID_X9_62_prime256v1	1
NID_secp256k1	1
NID_secp384r1	1
NID_secp521r1	1
«ES256»	1
«ES256K»	1
«ES384»	1
«ES512»	1
rs_len	3
0	6
ctx	6
nullptr	3
der_sig_len	5
der_signature	5
n2 = 23	N2 = 70

Данные, полученные в ходе обработки программного кода функции «sign_ecdsa» были использованы для расчета характеристик программы по формулам из метрики Холстеда. Результаты вычислений представлены в таблице 6.

Таблица 6 – Результаты вычислений по метрике Холстеда

Значение	Результат
Длина реализации программы	158
Словарь программы	40
Объем программы	840,8602
Сложность программы	25,8695

Все полученные показатели программы находятся в пределах нормы и удовлетворяют требованиям по метрикам программного кода.

5.3.2 Метрика Чепина

Расчет метрики Чепина будет производиться над той же функцией «sign_ecdsa», исходный код которой представлен в листинге 3 приложения А.

Для расчета метрики Чепина берется все множество переменных, составляющих список ввода-вывода, а затем разбивается на четыре функциональные группы:

- вводимые переменные для расчетов и для обеспечения вывода (Р);
- модифицируемые внутри программы переменные (М);
- переменные, участвующие в управлении работой программного модуля (С);
- не используемые в программе переменные (Т).

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе. Метрика Чепина:

$Q = a1 * P + a2 * M + a3 * C + a4 * T$, где $a1, a2, a3, a4$ – весовые коэффициенты.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы. По мнению автора метрики, наибольший вес, равный 3, имеет функциональная группа С, так как она влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: $a1=1, a2=2, a3=3, a4=0.5$. Весовой коэффициент группы Т не равен 0, поскольку «паразитные» переменные не увеличивают сложность потока данных программы, но иногда затрудняют ее понимание.

В выбранной функции было выделено 4 вводимых переменных, 4 модифицируемых внутри, 2 управляющих переменных и не оказалось «паразитной» переменной. Таким образом, формула подсчета метрики Чепина для данного модуля:

$$Q = 4 + 2 * 4 + 3 * 2 + 0.5 * 0 = 18.$$

Данный результат вычисления метрики Чепина указывает на то, что рассматриваемый компонент имеет относительно небольшой размер и невысокую сложность исходного кода.

Заключение

Результатом выполнения выпускной квалификационной работы стала библиотека, позволяющая создавать и валидировать JWT, а также предоставляющая функционал для работы с заголовком и утверждениями. Была проведена оценка финансовых и временных задач на разработку.

На стадии анализа требований была изучена технология JWT, получены знания о создании и использовании программных библиотек, изучены аналогичные разработки.

Для проекта было осуществлен анализ функциональных требований, требований к архитектуре и требований к интеграции и использованию.

Далее на стадии проектирования была описана задумка разработки внешнего программного интерфейса и его внутренней реализации.

После этого библиотека была разработана, а все разработанные функции описаны. Также была упомянута разработка примеров использования библиотеки.

И наконец в заключительной части было проведено тестирование программного проекта, которая показала его работоспособность и пригодность к использованию.

Таким образом, поставленная задача, а также все промежуточные цели, были выполнены в полном объеме.

Список источников

1. RFC 7519 (JWT, JSON Web Token) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата обращения: 13.05.2025).
2. RFC 8259 (The JavaScript Object Notation (JSON) Data Interchange Format) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc8259> (дата обращения: 26.05.2025).
3. RFC 6749 (The OAuth 2.0 Authorization Framework) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc6749> (дата обращения: 13.05.2025).
4. OpenID Connect Core 1.0 // OpenID Foundation [Электронный ресурс]. URL: https://openid.net/specs/openid-connect-core-1_0.html (дата обращения: 13.05.2025).
5. RFC 7515 (JWS, JSON Web Signature) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7515> (дата обращения: 13.05.2025).
6. RFC 7516 (JWE, JSON Web Encryption) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7516> (дата обращения: 13.05.2025).
7. RFC 4648 (The Base16, Base32, and Base64 Data Encodings) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc4648> (дата обращения: 13.05.2025).
8. RFC 7518 (JWA, JSON Web Algorithms) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7518> (дата обращения: 13.05.2025).
9. Библиотека (программирование) // Wikipedia [Электронный ресурс]. [https://ru.wikipedia.org/wiki/Библиотека_\(программирование\)](https://ru.wikipedia.org/wiki/Библиотека_(программирование)) (дата обращения: 13.05.2025).

10. Libraries for Token Signing/Verification // JWT IO [Электронный ресурс]. URL: <https://jwt.io/libraries> (дата обращения: 26.05.2025).
11. Node jsonwebtoken // Github [Электронный ресурс]. URL: <https://github.com/auth0/node-jwebtoken> (дата обращения: 26.05.2025).
12. PyJWT // Github [Электронный ресурс]. URL: <https://github.com/jpadilla/pyjwt> (дата обращения: 26.05.2025).
13. Java JWT: JSON Web Token for Java and Android // Github [Электронный ресурс]. URL: <https://github.com/jwtkt/jjwt> (дата обращения: 26.05.2025).
14. Jose // Github [Электронный ресурс]. URL: <https://github.com/panva/jose> (дата обращения: 26.05.2025).
15. Go JOSE // Github [Электронный ресурс]. URL: <https://github.com/go-jose/go-jose> (дата обращения: 26.05.2025).
16. PHP-JWT // Github [Электронный ресурс]. URL: <https://github.com/firebase/php-jwt> (дата обращения: 26.05.2025).
17. Ruby-JWT // Github [Электронный ресурс]. URL: <https://github.com/jwt/ruby-jwt> (дата обращения: 26.05.2025).
18. RFC 2818 (HTTP Over TLS) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc2818> (дата обращения: 14.05.2025).
19. RFC 7517 (JWK, JSON Web Key) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7517> (дата обращения: 13.05.2025).
20. RFC 1421 (Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc1421> (дата обращения: 14.05.2025).
21. RFC 7468 (Textual Encodings of PKIX, PKCS, and CMS Structures) // Internet Engineering Task Force (IETF) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc7468> (дата обращения: 14.05.2025).

22. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624с.

23. The ZeroMQ Project // Github [Электронный ресурс]. URL: <https://github.com/zeromq> (дата обращения: 13.05.2025).

24. nlohmann/json: JSON for modern C++ // Github [Электронный ресурс]. URL: <https://github.com/nlohmann/json> (дата обращения: 13.05.2025).

25. GoogleTest – Google Testing and Mocking Framework // Github [Электронный ресурс]. URL: <https://github.com/google/googletest> (дата обращения: 13.05.2025).

26. openssl/openssl: TLS/SSL and crypto library // Github [Электронный ресурс]. URL: <https://github.com/openssl/openssl> (дата обращения: 13.05.2025).

Приложение А

(обязательное)

Листинг исходного кода библиотеки

Листинг A.1 – Файл abc_jwt.h

```

#ifndef ABC_JWT_H
#define ABC_JWT_H

#include <stddef.h> // для size_t
#include "abc_jwt_export.h"

#ifdef __cplusplus
extern "C" {
#endif

// --- Непрозрачные типы (указатели на внутренние структуры) ---
/**
 * @brief Непрозрачный хэндл для контекста создания/проверки JWT.
 */
typedef struct JwtContextInternal* JwtHandle;

/**
 * @brief Непрозрачный хэндл для криптографического ключа.
 */
typedef struct JwtKeyInternal* JwtKeyHandle;

/**
 * @brief Непрозрачный хэндл для набора ключей
 */
typedef struct KeySetInternal* KeySetHandle;

// --- Коды ошибок ---
typedef enum {
    JWT_SUCCESS = 0, // Операция успешна
    JWT_ERROR_ALLOC = -1, // Ошибка выделения памяти
    JWT_ERROR_INVALID_ARG = -2, // Неверный аргумент функции
    JWT_ERROR_OPENSSL = -3, // Внутренняя ошибка OpenSSL
    JWT_ERROR_JSON = -4, // Ошибка парсинга или сериализации JSON
    JWT_ERROR_INVALID_SIGNATURE = -5, // Неверная
    криптографическая подпись
    JWT_ERROR_ALG_NOT_SUPPORTED = -6, // Алгоритм не
    поддерживается или несовместим с ключом
    JWT_ERROR_KEY_LOAD = -7, // Ошибка загрузки/формата ключа
    JWT_ERROR_CLAIM_NOT_FOUND = -8, // Запрошенный claim не найден
    JWT_ERROR_INVALID_TOKEN_FORMAT = -9, // Неверный формат токена
    JWT
    JWT_ERROR_INTERNAL = -10, // Другая внутренняя ошибка
    JWT_ERROR_CLAIM_TYPE_MISMATCH = -11, // Запрошенный claim
    найден, но имеет неверный тип
    JWT_ERROR_VALIDATION_FAILED = -20, // Общая ошибка валидации
    JWT_ERROR_VALIDATION_EXP_EXPIRED = -21, // Токен истек (exp)
    JWT_ERROR_VALIDATION_NBF_NOT_YET_VALID = -22, // Токен еще не
    действителен (nbf)
    JWT_ERROR_VALIDATION_IAT_INVALID = -23, // Невалидное время
    создания (iat) (опционально)

```

```

    JWT_ERROR_VALIDATION_ISS_MISMATCH = -24, // Неверный издатель
    (iss)
    JWT_ERROR_VALIDATION_AUD_MISMATCH = -25, // Неверная аудитория
    (aud)
    JWT_ERROR_VALIDATION_CLAIM_MISSING = -26, // Обязательный для
    валидации claim отсутствует
    JWT_ERROR_KEY_CURVE_MISMATCH = -27, // Кривая ключа не
    соответствует алгоритму
    JWT_ERROR_KEY_NOT_FOUND_FOR_KID = -28, // Ключ для kid не
    найден в наборе
    JWT_ERROR_KID_MISSING_OR_INVALID = -29, // Отсутствует или
    невалидный 'kid' в заголовке JWT
    JWT_ERROR_KEY_USE_MISMATCH = -30, // Неверное значение "use" в
    JWK для операции
    JWT_ERROR_KEY_OP_NOT_ALLOWED = -31, // Операция не разрешена в
    "key_ops" JWK
    JWT_ERROR_KEY_ALG_MISMATCH = -32, // Алгоритм JWT не совпадает
    с "alg" в JWK
    JWT_ERROR_KEY_EXPORT = -33, // Ошибка экспорта ключа
    JWT_ERROR_X5C_INVALID_FORMAT = -34, // Неверный формат
    заголовка x5c (не массив строк, ошибка Base64/DER)
    JWT_ERROR_X5C_KEY_EXTRACTION = -35, // Не удалось извлечь ключ
    из сертификата в x5c
    JWT_ERROR_HEADER_CLAIM_INVALID_TYPE = -36, // Заголовок или
    claim имеет неверный тип (уточнение для get_header_*)
    JWT_ERROR_KEY_GENERATION = -37, // Ошибка генерации ключа
    JWT_ERROR_CRIT_INVALID_FORMAT = -38, // Неверный формат crit
    (не массив строк, пустой)
    JWT_ERROR_CRIT_UNKNOWN_HEADER = -39, // Неизвестный или
    необрабатываемый критический заголовок
    JWT_ERROR_CRIT_HEADER_MISSING = -40, // Критический заголовок
    отсутствует в JWT
    JWT_ERROR_CRIT_STANDARD_HEADER_LISTED = -41 // Стандартный
    заголовок JOSE указан как критический
} JwtError;

// --- Функции управления ресурсами (пока только объявления) ---

/**
 * @brief Создает контекст для формирования нового JWT.
 * @param handle_out Указатель, куда будет записан созданный
хэнгл.
 * @return JWT_SUCCESS в случае успеха, иначе код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_create(JwtHandle* handle_out);

/**
 * @brief Уничтожает контекст JWT и освобождает связанные ресурсы.
 * @param handle Хэнгл для уничтожения. Если NULL, ничего не
делает.
 */
ABC_JWT_EXPORT void jwt_destroy(JwtHandle handle);

```

```
// --- Функции управления ресурсами KeySet ---

/**
 * @brief Загружает набор ключей из строки JSON Web Key Set
 * (JWKS).
 * Ожидает JSON объект с массивом "keys", где каждый элемент - JWK
 * с "kid".
 * @param jwks_json_str Нуль-терминированная строка с JWKS в
 * формате JSON.
 * @param ignore_load_errors Если не 0, ошибки загрузки отдельных
 * ключей в наборе
 * будут проигнорированы. Если 0, любая ошибка ключа
 * приведет к провалу загрузки всего набора.
 * @param keyset_handle_out Указатель, куда будет записан хэндл
 * набора ключей.
 * Этот хэндл нужно будет уничтожить с помощью
 * jwt_keyset_destroy().
 * @return JWT_SUCCESS в случае успеха, иначе код ошибки
 * (JWT_ERROR_JSON, JWT_ERROR_KEY_LOAD).
 */
ABC_JWT_EXPORT JwtError jwt_keyset_load_jwks(const char*
jwks_json_str, int ignore_load_errors, KeySetHandle*
keyset_handle_out);

/**
 * @brief Экспортирует ВСЕ набор ключей из KeySetHandle в строку
 * формата JWK Set JSON.
 * @param ks_handle Хэндл набора ключей для экспорта.
 * @param include_private Не 0, чтобы включить приватные
 * компоненты для ВСЕХ ключей, 0 - только публичные.
 * @param jwks_json_out Указатель, куда будет записана строка с
 * JWK Set JSON (`{"keys": [...]}`).
 * Память выделяется библиотекой, освободить через
 * jwt_free_string().
 * @return JWT_SUCCESS или код ошибки (JWT_ERROR_KEY_EXPORT,
 * JWT_ERROR_ALLOC).
 */
ABC_JWT_EXPORT JwtError jwt_keyset_export_jwks(KeySetHandle
ks_handle, int include_private, char** jwks_json_out);

/**
 * @brief Ищет ключ в наборе по его Key ID ('kid').
 * @param keyset_handle Хэндл набора ключей.
 * @param kid Идентификатор ключа для поиска.
 * @param key_handle_out Указатель, куда будет записан хэндл
 * найденного ключа.
 * Этот хэндл является "копией" (увеличивает счетчик ссылок
 * внутреннего ключа OpenSSL) и должен быть уничтожен
 * отдельно с помощью jwt_key_destroy(). Он действителен
 * даже после уничтожения keyset_handle.
 */

```

```

    * @return JWT_SUCCESS, если ключ найден, или код ошибки
    (JWT_ERROR_KEY_NOT_FOUND_FOR_KID).
    */
ABC_JWT_EXPORT JwtError jwt_keyset_find_key_by_kid(KeySetHandle
keyset_handle, const char* kid, JwtKeyHandle* key_handle_out);

/**
 * @brief Включает/выключает принудительную проверку параметра
 "use" ключа JWK
 * при поиске ключа для верификации через jwt_load_verify_jwks.
 * Если включено, параметр "use" найденного ключа (если он есть)
 должен быть "sig".
 * По умолчанию выключено (0).
 * @param keyset_handle Хэндл набора ключей.
 * @param enforce Не 0 для включения проверки, 0 для выключения.
 * @return JWT_SUCCESS или JWT_ERROR_INVALID_ARG.
 */
ABC_JWT_EXPORT JwtError jwt_keyset_set_enforce_use(KeySetHandle
keyset_handle, int enforce);

/**
 * @brief Включает/выключает принудительную проверку параметра
 "key_ops" ключа JWK
 * при поиске ключа для верификации через jwt_load_verify_jwks.
 * Если включено, параметр "key_ops" найденного ключа (если он
 есть) должен содержать "verify".
 * По умолчанию выключено (0).
 * @param keyset_handle Хэндл набора ключей.
 * @param enforce Не 0 для включения проверки, 0 для выключения.
 * @return JWT_SUCCESS или JWT_ERROR_INVALID_ARG.
 */
ABC_JWT_EXPORT JwtError
jwt_keyset_set_enforce_key_ops(KeySetHandle keyset_handle, int
enforce);

/**
 * @brief Включает/выключает принудительную проверку параметра
 "alg" ключа JWK
 * при поиске ключа для верификации через jwt_load_verify_jwks.
 * Если включено, параметр "alg" найденного ключа (если он есть)
 должен совпадать
 * с параметром "alg" из заголовка проверяемого JWT.
 * По умолчанию выключено (0).
 * @param keyset_handle Хэндл набора ключей.
 * @param enforce Не 0 для включения проверки, 0 для выключения.
 * @return JWT_SUCCESS или JWT_ERROR_INVALID_ARG.
 */
ABC_JWT_EXPORT JwtError jwt_keyset_set_enforce_alg(KeySetHandle
keyset_handle, int enforce);

/**

```

```

    * @brief Уничтожает набор ключей и освобождает все связанные
ресурсы (включая ключи).
    * @param keyset_handle Хэндл набора ключей для уничтожения. Если
NULL, ничего не делает.
    */
ABC_JWT_EXPORT void jwt_keyset_destroy(KeySetHandle
keyset_handle);

/**
    * @brief Получает параметр "use" (Public Key Use) ключа из набора
JWKS.
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @return Указатель на строку с значением "use" ("sig", "enc",
...) или NULL,
    * если ключ не найден или параметр "use" отсутствует у ключа.
    * ВНИМАНИЕ: Возвращаемый указатель указывает на внутренние данные
KeySet.
    * Он действителен только пока действителен keyset_handle и не
должен освобождаться.
    */
ABC_JWT_EXPORT const char* jwt_keyset_get_key_use(KeySetHandle
keyset_handle, const char* kid);

/**
    * @brief Получает параметр "alg" (Algorithm) ключа из набора
JWKS.
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @return Указатель на строку с значением "alg" (например,
"RS256") или NULL,
    * если ключ не найден или параметр "alg" отсутствует у ключа.
    * ВНИМАНИЕ: Возвращаемый указатель указывает на внутренние
данные KeySet.
    * Он действителен только пока действителен keyset_handle и не
должен освобождаться.
    */
ABC_JWT_EXPORT const char* jwt_keyset_get_key_alg(KeySetHandle
keyset_handle, const char* kid);

/**
    * @brief Получает количество операций в параметре "key_ops" (Key
Operations) ключа из набора JWKS.
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @return Количество операций (>= 0) или -1, если ключ не найден
или параметр "key_ops" отсутствует/не массив.
    */
ABC_JWT_EXPORT int jwt_keyset_get_key_ops_count(KeySetHandle
keyset_handle, const char* kid);

/**

```

```

    * @brief Получает строку операции из "key_ops" по индексу для
ключа из набора JWKS.
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @param index Индекс операции (от 0 до count-1).
    * @return Указатель на строку с операцией (например, "sign",
"verify") или NULL,
    * если ключ не найден, индекс вне диапазона или "key_ops" не
является массивом строк.
    * ВНИМАНИЕ: Возвращаемый указатель указывает на внутренние
данные KeySet.
    * Он действителен только пока действителен keyset_handle и не
должен освобождаться.
    */
ABC_JWT_EXPORT const char*
jwt_keyset_get_key_op_at_index(KeySetHandle keyset_handle, const
char* kid, int index);

/**
    * @brief Получает количество сертификатов в параметре "x5c" ключа
из набора JWKS.
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @return Количество сертификатов (>= 0) или -1, если
ключ/параметр "x5c" отсутствует или не массив.
    */
ABC_JWT_EXPORT int jwt_keyset_get_key_x5c_count(KeySetHandle
keyset_handle, const char* kid);

/**
    * @brief Получает сертификат из параметра "x5c" ключа по индексу
в формате PEM.
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @param index Индекс сертификата в массиве 'x5c' (начиная с 0).
    * @return Указатель на строку с PEM-сертификатом или NULL при
ошибке.
    * ВНИМАНИЕ: Указатель указывает на внутренние данные.
Действителен пока жив keyset_handle. Не освобождать!
    */
ABC_JWT_EXPORT const char*
jwt_keyset_get_key_x5c_entry_pem(KeySetHandle keyset_handle, const
char* kid, int index);

/**
    * @brief Получает отпечаток сертификата SHA-1 из параметра "x5t"
ключа (Base64URL).
    * @param keyset_handle Хэндл набора ключей.
    * @param kid Идентификатор ключа ('kid').
    * @return Указатель на строку с отпечатком или NULL, если
ключ/параметр "x5t" отсутствует.

```

```

    * ВНИМАНИЕ: Указатель указывает на внутренние данные.
    Действителен пока жив keyset_handle. Не освобождать!
    */
ABC_JWT_EXPORT const char* jwt_keyset_get_key_x5t(KeySetHandle
keyset_handle, const char* kid);

/**
 * @brief Получает отпечаток сертификата SHA-256 из параметра
"x5t#S256" ключа (Base64URL).
 * @param keyset_handle Хэндл набора ключей.
 * @param kid Идентификатор ключа ('kid').
 * @return Указатель на строку с отпечатком или NULL, если
ключ/параметр "x5t#S256" отсутствует.
 * ВНИМАНИЕ: Указатель указывает на внутренние данные.
    Действителен пока жив keyset_handle. Не освобождать!
    */
ABC_JWT_EXPORT const char*
jwt_keyset_get_key_x5t_s256(KeySetHandle keyset_handle, const
char* kid);

/**
 * @brief Загружает секретный ключ (например, для HMAC).
 * @param secret Указатель на байты секрета.
 * @param secret_len Длина секрета в байтах.
 * @param key_handle_out Указатель, куда будет записан хэндл
ключа.
 * @return JWT_SUCCESS в случае успеха, иначе код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_key_load_secret(const unsigned char*
secret, size_t secret_len, JwtKeyHandle* key_handle_out);

/**
 * @brief Загружает приватный ключ из строки PEM.
 * @param pem_data Нуль-терминированная строка с ключом в формате
PEM.
 * @param password Пароль для расшифровки (NULL или пустая строка,
если не зашифрован).
 * @param key_handle_out Указатель, куда будет записан хэндл
ключа.
 * @return JWT_SUCCESS в случае успеха, иначе код ошибки
(JWT_ERROR_KEY_LOAD).
    */
ABC_JWT_EXPORT JwtError jwt_key_load_private_pem(const char*
pem_data, const char* password, JwtKeyHandle* key_handle_out);

/**
 * @brief Загружает публичный ключ из строки PEM.
 * @param pem_data Нуль-терминированная строка с ключом в формате
PEM.
 * @param key_handle_out Указатель, куда будет записан хэндл
ключа.

```

```

    * @return JWT_SUCCESS в случае успеха, иначе код ошибки
    (JWT_ERROR_KEY_LOAD).
    */
ABC_JWT_EXPORT JwtError jwt_key_load_public_pem(const char*
pem_data, JwtKeyHandle* key_handle_out);

/**
    * @brief Загружает ключ из строки JSON Web Key (JWK).
    * Поддерживаемые 'kty': "RSA", "EC", "oct" (для HMAC), "OKP"
    (Ed25519/Ed448).
    * @param jwk_json_str Нуль-терминированная строка с JWK в формате
    JSON.
    * @param key_handle_out Указатель, куда будет записан хэнгл
    ключа.
    * @return JWT_SUCCESS в случае успеха, иначе код ошибки
    (JWT_ERROR_JSON, JWT_ERROR_KEY_LOAD).
    */
ABC_JWT_EXPORT JwtError jwt_key_load_jwk(const char* jwk_json_str,
JwtKeyHandle* key_handle_out);

/**
    * @brief Экспортирует ключ в формат JSON Web Key (JWK) в виде
    строки.
    * Экспортирует только материал ключа. Метаданные (use, key_ops,
    alg, kid) не включаются.
    * Не поддерживает экспорт симметричных ключей ("kty": "oct").
    * @param key_handle Хэнгл ключа для экспорта.
    * @param include_private Не 0, чтобы включить приватные
    компоненты, 0 - только публичные.
    * @param jwk_json_out Указатель, куда будет записана строка с JWK
    JSON.
    * Память для строки выделяется библиотекой, ее нужно освободить
    * с помощью jwt_free_string().
    * @return JWT_SUCCESS или код ошибки (JWT_ERROR_KEY_EXPORT,
    JWT_ERROR_ALLOC).
    */
ABC_JWT_EXPORT JwtError jwt_key_export_jwk(JwtKeyHandle
key_handle, int include_private, char** jwk_json_out);

/**
    * @brief Экспортирует ключ из набора JWK Set в формат JWK JSON
    строки, включая метаданные.
    * @param keyset_handle Хэнгл набора ключей.
    * @param kid Идентификатор ключа ('kid') для экспорта.
    * @param include_private Не 0, чтобы включить приватные
    компоненты, 0 - только публичные.
    * @param jwk_json_out Указатель, куда будет записана строка с
    полным JWK JSON.
    * Память для строки выделяется библиотекой, ее нужно освободить с
    помощью jwt_free_string().

```

```

    * @return JWT_SUCCESS или код ошибки
    (JWT_ERROR_KEY_NOT_FOUND_FOR_KID, JWT_ERROR_KEY_EXPORT,
    JWT_ERROR_ALLOC).
    */
ABC_JWT_EXPORT JwtError jwt_keyset_export_jwk(KeySetHandle
keyset_handle, const char* kid, int include_private, char**
jwk_json_out);

/**
 * @brief Генерирует новую пару ключей RSA.
 * @param bits Размер ключа в битах (например, 2048, 3072, 4096).
Рекомендуется >= 2048.
 * @param key_handle_out Указатель, куда будет записан хэнгл
нового ключа.
 * @return JWT_SUCCESS или код ошибки (JWT_ERROR_KEY_GENERATION,
JWT_ERROR_ALLOC).
 */
ABC_JWT_EXPORT JwtError jwt_key_generate_rsa(int bits,
JwtKeyHandle* key_handle_out);

/**
 * @brief Генерирует новую пару ключей EC для указанной кривой.
 * @param curve_name Имя кривой (например, "P-256", "secp256k1",
"P-384", "P-521").
 * @param key_handle_out Указатель, куда будет записан хэнгл
нового ключа.
 * @return JWT_SUCCESS или код ошибки (JWT_ERROR_INVALID_ARG,
JWT_ERROR_KEY_GENERATION, JWT_ERROR_ALLOC).
 */
ABC_JWT_EXPORT JwtError jwt_key_generate_ec(const char*
curve_name, JwtKeyHandle* key_handle_out);

/**
 * @brief Генерирует новую пару ключей EdDSA (Ed25519 или Ed448).
(Требуется OpenSSL 1.1.1+)
 * @param curve_name Имя кривой ("Ed25519" или "Ed448").
 * @param key_handle_out Указатель, куда будет записан хэнгл
нового ключа.
 * @return JWT_SUCCESS или код ошибки (JWT_ERROR_INVALID_ARG,
JWT_ERROR_KEY_GENERATION, JWT_ERROR_ALLOC).
 */
ABC_JWT_EXPORT JwtError jwt_key_generate_ed(const char*
curve_name, JwtKeyHandle* key_handle_out);

/**
 * @brief Уничтожает ключ и освобождает связанные ресурсы.
 * @param key_handle Хэнгл ключа для уничтожения. Если NULL,
ничего не делает.
 */
ABC_JWT_EXPORT void jwt_key_destroy(JwtKeyHandle key_handle);

// --- Вспомогательные функции ---

```

```

/**
 * @brief Возвращает строковое представление для кода ошибки JWT.
 * @param error_code Код ошибки JwtError.
 * @return Статическая строка с описанием ошибки. Не освобождать!
 */
ABC_JWT_EXPORT const char* jwt_error_str(JwtError error_code);

/**
 * @brief Освобождает память, выделенную библиотекой для строки.
 * Используется для строк, возвращаемых, например, jwt_sign.
 * @param str Указатель на строку для освобождения.
 */
ABC_JWT_EXPORT void jwt_free_string(char* str);

// --- Функции для работы с JWT ---

/**
 * @brief Устанавливает алгоритм подписи ('alg') в заголовке JWT.
 * @param handle Контекст JWT.
 * @param alg Строка с названием алгоритма (например, "HS256").
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_set_alg(JwtHandle handle, const char*
alg);

/**
 * @brief Добавляет строковый claim в полезную нагрузку JWT.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) claim.
 * @param value Значение claim.
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_add_claim_string(JwtHandle handle,
const char* key, const char* value);

/**
 * @brief Добавляет целочисленный claim (long long) в полезную
нагрузку JWT.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) claim.
 * @param value Значение claim (64-битное целое).
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_add_claim_int(JwtHandle handle, const
char* key, long long value);

/**
 * @brief Добавляет claim с плавающей точкой (double) в полезную
нагрузку JWT.
 * @param handle Контекст JWT.

```

```

    * @param key Ключ (имя) claim.
    * @param value Значение claim.
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_add_claim_double(JwtHandle handle,
const char* key, double value);

/**
 * @brief Добавляет булев claim в полезную нагрузку JWT.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) claim.
 * @param value Значение claim (0 для false, не 0 для true).
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_add_claim_bool(JwtHandle handle, const
char* key, int value);

/**
 * @brief Добавляет claim из строки, содержащей валидный JSON
(для массивов/объектов).
 * @param handle Контекст JWT.
 * @param key Ключ (имя) claim.
 * @param json_value_str Строка с валидным JSON значением.
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_add_claim_json(JwtHandle handle, const
char* key, const char* json_value_str);

// --- Функции для добавления параметров заголовка ---

/**
 * @brief Добавляет строковый параметр в заголовок.
 * НЕ ИСПОЛЬЗУЙТЕ для установки 'alg' - для этого есть
jwt_set_alg.
 * @param handle Контекст JWT (созданный через jwt_create).
 * @param key Ключ (имя) параметра заголовка (например, "typ",
"cty", "kid").
 * @param value Строковое значение параметра.
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_add_header_string(JwtHandle handle,
const char* key, const char* value);

/**
 * @brief Добавляет целочисленный параметр (long long) в
заголовок.
 * НЕ ИСПОЛЬЗУЙТЕ для установки 'alg'.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) параметра заголовка.
 * @param value Целочисленное значение параметра.
 * @return JWT_SUCCESS или код ошибки.
 */

```

```
ABC_JWT_EXPORT JwtError jwt_add_header_int(JwtHandle handle, const
char* key, long long value);
```

```
/**
 * @brief Добавляет параметр с плавающей точкой (double) в
заголовок.
 * НЕ ИСПОЛЬЗУЙТЕ для установки 'alg'.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) параметра заголовка.
 * @param value Вещественное значение параметра.
 * @return JWT_SUCCESS или код ошибки.
 */
```

```
ABC_JWT_EXPORT JwtError jwt_add_header_double(JwtHandle handle,
const char* key, double value);
```

```
/**
 * @brief Добавляет булев параметр в заголовок.
 * НЕ ИСПОЛЬЗУЙТЕ для установки 'alg'.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) параметра заголовка.
 * @param value Булево значение (0 = false, не 0 = true).
 * @return JWT_SUCCESS или код ошибки.
 */
```

```
ABC_JWT_EXPORT JwtError jwt_add_header_bool(JwtHandle handle,
const char* key, int value);
```

```
/**
 * @brief Добавляет параметр из строки, содержащей валидный JSON,
в заголовок.
 * Полезно для добавления массивов или вложенных объектов в
заголовок.
 * НЕ ИСПОЛЬЗУЙТЕ для установки 'alg'.
 * @param handle Контекст JWT.
 * @param key Ключ (имя) параметра заголовка.
 * @param json_value_str Строка с валидным JSON значением.
 * @return JWT_SUCCESS или код ошибки.
 */
```

```
ABC_JWT_EXPORT JwtError jwt_add_header_json(JwtHandle handle,
const char* key, const char* json_value_str);
```

```
/**
 * @brief Подписывает текущий JWT заданным ключом.
 * @param handle Контекст JWT с установленным алгоритмом и
claims.
 * @param key_handle Хэндл ключа для подписи.
 * @param token_out Указатель, куда будет записана строка с
подписанным JWT.
 * Память для строки выделяется библиотекой, ее нужно освободить
 * с помощью jwt_free_string().
 * @return JWT_SUCCESS или код ошибки.
 */
```

```

ABC_JWT_EXPORT JwtError jwt_sign(JwtHandle handle, JwtKeyHandle
key_handle, char** token_out);

/**
 * @brief Загружает JWT из строки, проверяет его подпись и
парсит.
 * НЕ ВЫПОЛНЯЕТ ВАЛИДАЦИЮ CLAIMS (exp, nbf, iss, aud и т.д.).
 * Для валидации claims используйте jwt_validate_claims() после
этой функции.
 * @param token_str Строка с JWT.
 * @param key_handle Хэндл ключа для проверки подписи.
 * @param handle_out Указатель, куда будет записан хэндл
контекста JWT с
 * распарсенными данными в случае успешной проверки подписи.
 * Этот хэндл нужно будет позже уничтожить с помощью
jwt_destroy().
 * @return JWT_SUCCESS или код ошибки (включая
JWT_ERROR_INVALID_SIGNATURE).
 */
ABC_JWT_EXPORT JwtError jwt_load_verify(const char* token_str,
JwtKeyHandle key_handle, JwtHandle* handle_out);

/**
 * @brief Загружает JWT, ищет ключ для проверки в JWK Set по 'kid'
из заголовка,
 * проверяет подпись найденным ключом и парсит токен.
 * НЕ ВЫПОЛНЯЕТ ВАЛИДАЦИЮ CLAIMS (exp, nbf, iss, aud и т.д.).
 * Для валидации claims используйте jwt_validate_claims() после
этой функции.
 * @param token_str Строка с JWT (должен содержать 'kid' в
заголовке).
 * @param keyset_handle Хэндл набора ключей JWKS для поиска ключа
проверки.
 * @param handle_out Указатель, куда будет записан хэндл контекста
JWT с
 * распарсенными данными в случае успешной проверки подписи.
 * Этот хэндл нужно будет позже уничтожить с помощью
jwt_destroy().
 * @return JWT_SUCCESS или код ошибки (включая
JWT_ERROR_INVALID_SIGNATURE,
 * JWT_ERROR_KID_MISSING_OR_INVALID,
JWT_ERROR_KEY_NOT_FOUND_FOR_KID).
 */
ABC_JWT_EXPORT JwtError jwt_load_verify_jwks(const char*
token_str, KeySetHandle keyset_handle, JwtHandle* handle_out);

/**
 * @brief Загружает JWT, извлекает публичный ключ из ПЕРВОГО
сертификата в заголовке 'x5c',
 * проверяет подпись этим ключом и парсит токен.
 * ВНИМАНИЕ: НЕ ВЫПОЛНЯЕТ ВАЛИДАЦИЮ СЕРТИФИКАТА ИЛИ ЦЕПОЧКИ
ДОВЕРИЯ!

```

```

* Это должно быть сделано вызывающим приложением согласно RFC
7515 / RFC 5280.
* НЕ ВЫПОЛНЯЕТ ВАЛИДАЦИЮ CLAIMS (exp, nbf, iss, aud и т.д.).
* @param token_str Строка с JWT (должен содержать 'x5c' в
заголовке).
* @param handle_out Указатель, куда будет записан хэндл контекста
JWT с
* распарсенными данными в случае успешной проверки подписи.
* @return JWT_SUCCESS или код ошибки (включая
JWT_ERROR_INVALID_SIGNATURE,
* JWT_ERROR_X5C_INVALID_FORMAT, JWT_ERROR_X5C_KEY_EXTRACTION).
*/
ABC_JWT_EXPORT JwtError jwt_load_verify_x5c(const char* token_str,
JwtHandle* handle_out);

/**
* @brief Добавляет сертификат X.509 в заголовок 'x5c' JWS.
* Функция автоматически определяет формат входных данных (PEM или
DER),
* конвертирует сертификат в DER (если необходимо), кодирует его в
стандартный Base64
* и добавляет в JSON-массив 'x5c' в заголовке.
* Может быть вызвана несколько раз для добавления цепочки
сертификатов.
* Порядок добавления сохраняется.
* @param handle Контекст JWT (созданный через jwt_create).
* @param cert_data Указатель на данные сертификата.
* @param cert_len Длина данных сертификата в байтах.
* @return JWT_SUCCESS или код ошибки
(JWT_ERROR_X5C_INVALID_FORMAT, JWT_ERROR_ALLOC,
JWT_ERROR_INTERNAL).
*/
ABC_JWT_EXPORT JwtError
jwt_add_header_x5c_certificate_data(JwtHandle handle, const
unsigned char* cert_data, size_t cert_len);

// --- Функции настройки валидации (вызывать перед
jwt_validate_claims) ---

/**
* @brief Устанавливает допустимое отклонение времени (секунды)
для проверок exp, nbf, iat.
* Используется для компенсации рассинхронизации часов. По
умолчанию 0.
* @param handle Контекст JWT (полученный из jwt_load_verify).
* @param leeway_seconds Допустимое отклонение в секундах
(например, 60).
* @return JWT_SUCCESS или код ошибки.
*/
ABC_JWT_EXPORT JwtError jwt_set_validation_leeway(JwtHandle
handle, unsigned int leeway_seconds);

```

```

/**
 * @brief Устанавливает ожидаемого издателя (iss) для валидации.
 * Если установлено, claim 'iss' в токене должен присутствовать и
 совпадать.
 * @param handle Контекст JWT.
 * @param issuer Ожидаемая строка издателя (может быть NULL для
 отмены проверки).
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_set_expected_issuer(JwtHandle handle,
const char* issuer);

/**
 * @brief Добавляет ожидаемую аудиторию (aud) в список для
 валидации.
 * Если список не пуст, claim 'aud' в токене должен
 присутствовать, и хотя бы
 * одно значение из claim 'aud' (если это массив) или само
 значение (если строка)
 * должно совпадать с одним из значений в этом списке.
 * Можно вызывать несколько раз для добавления нескольких
 допустимых аудиторий.
 * @param handle Контекст JWT.
 * @param audience Ожидаемая строка аудитории.
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_add_expected_audience(JwtHandle
handle, const char* audience);

// --- Функция валидации Claims ---

/**
 * @brief Выполняет валидацию стандартных claims (exp, nbf, iat,
 iss, aud)
 * для загруженного и проверенного (по подписи) токена.
 * Использует параметры валидации, установленные через
 jwt_set_validation_leeway,
 * jwt_set_expected_issuer, jwt_add_expected_audience.
 * Валидация выполняется на основе текущего системного времени.
 * @param handle Контекст JWT, полученный после успешного
 jwt_load_verify.
 * @return JWT_SUCCESS, если все применимые проверки пройдены,
 иначе код ошибки
 * (JWT_ERROR_VALIDATION_EXP_EXPIRED,
 JWT_ERROR_VALIDATION_NBF_NOT_YET_VALID,
 * JWT_ERROR_VALIDATION_ISS_MISMATCH,
 JWT_ERROR_VALIDATION_AUD_MISMATCH,
 * JWT_ERROR_VALIDATION_CLAIM_MISSING,
 JWT_ERROR_CLAIM_TYPE_MISMATCH).
 */
ABC_JWT_EXPORT JwtError jwt_validate_claims(JwtHandle handle);

```

```
// --- Функции настройки и выполнения валидации `crit` ---

/**
 * @brief Добавляет имя заголовка в список понятных приложению для
 валидации 'crit'.
 * Вызывать после jwt_load_verify* и перед
 jwt_validate_crit_headers.
 * @param handle Контекст JWT.
 * @param header_name Имя заголовка, который приложение понимает и
 обрабатывает.
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwError jwt_add_understood_crit_header(JwtHandle
handle, const char* header_name);

/**
 * @brief Выполняет валидацию заголовка 'crit' согласно RFC 7515.
 * Проверяет, что все заголовки, перечисленные в 'crit',
 присутствуют,
 * не являются стандартными и были добавлены в список понятных
 через
 * jwt_add_understood_crit_header.
 * Вызывать после jwt_load_verify*.
 * @param handle Контекст JWT.
 * @return JWT_SUCCESS, если валидация 'crit' прошла успешно (или
 'crit' отсутствует),
 * иначе соответствующий код ошибки JWT_ERROR_CRIT_*.
 */
ABC_JWT_EXPORT JwError jwt_validate_crit_headers(JwtHandle
handle);

// --- Функции получения Claims (после jwt_load_verify) ---

/**
 * @brief Получает строковый claim из полезной нагрузки
 проверенного JWT.
 * @param handle Контекст JWT, полученный из jwt_load_verify.
 * @param key Ключ (имя) claim.
 * @param value_out Указатель, куда будет записана строка со
 значением claim.
 * Память для строки выделяется библиотекой, ее нужно освободить
 * с помощью jwt_free_string().
 * @return JWT_SUCCESS или код ошибки (JWT_ERROR_CLAIM_NOT_FOUND,
 JWT_ERROR_CLAIM_TYPE_MISMATCH).
 */
ABC_JWT_EXPORT JwError jwt_get_payload_claim_string(JwtHandle
handle, const char* key, char** value_out);

/**
 * @brief Получает целочисленный claim (long long) из полезной
 нагрузки проверенного JWT.
 * @param handle Контекст JWT.
```

```

    * @param key Ключ (имя) claim.
    * @param value_out Указатель, куда будет записано значение
claim.
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_payload_claim_int(JwtHandle
handle, const char* key, long long* value_out);

/**
    * @brief Получает claim с плавающей точкой (double) из полезной
нагрузки проверенного JWT.
    * @param handle Контекст JWT.
    * @param key Ключ (имя) claim.
    * @param value_out Указатель, куда будет записано значение
claim.
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_payload_claim_double(JwtHandle
handle, const char* key, double* value_out);

/**
    * @brief Получает булев claim из полезной нагрузки проверенного
JWT.
    * @param handle Контекст JWT.
    * @param key Ключ (имя) claim.
    * @param value_out Указатель, куда будет записано значение claim
(0 или 1).
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_payload_claim_bool(JwtHandle
handle, const char* key, int* value_out);

/**
    * @brief Получает claim как строку с JSON из полезной нагрузки
проверенного JWT.
    * @param handle Контекст JWT.
    * @param key Ключ (имя) claim.
    * @param json_value_out Указатель, куда будет записана строка с
JSON значением claim.
    * Память для строки выделяется библиотекой, ее нужно освободить
    * с помощью jwt_free_string().
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_payload_claim_json(JwtHandle
handle, const char* key, char** json_value_out);

// --- Функции получения параметров из Заголовка ---

/**
    * @brief Получает строковый параметр из заголовка проверенного
JWT.

```

```

    * @param handle Контекст JWT/JWE, полученный после успешного
    парсинга/проверки.
    * @param key Ключ (имя) параметра заголовка (например, "typ",
    "cty", "kid").
    * @param value_out Указатель, куда будет записана строка со
    значением параметра.
    * Память выделяется библиотекой, освободить через
    jwt_free_string().
    * @return JWT_SUCCESS или код ошибки (JWT_ERROR_CLAIM_NOT_FOUND,
    JWT_ERROR_HEADER_CLAIM_INVALID_TYPE).
    */
ABC_JWT_EXPORT JwtError jwt_get_header_claim_string(JwtHandle
handle, const char* key, char** value_out);

/**
    * @brief Получает целочисленный параметр (long long) из
    заголовка проверенного JWT.
    * @param handle Контекст JWT/JWE.
    * @param key Ключ (имя) параметра заголовка.
    * @param value_out Указатель, куда будет записано значение
    параметра.
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_header_claim_int(JwtHandle handle,
const char* key, long long* value_out);

/**
    * @brief Получает параметр с плавающей точкой (double) из
    заголовка проверенного JWT.
    * @param handle Контекст JWT/JWE.
    * @param key Ключ (имя) параметра заголовка.
    * @param value_out Указатель, куда будет записано значение
    параметра.
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_header_claim_double(JwtHandle
handle, const char* key, double* value_out);

/**
    * @brief Получает булев параметр из заголовка проверенного JWT.
    * @param handle Контекст JWT/JWE.
    * @param key Ключ (имя) параметра заголовка.
    * @param value_out Указатель, куда будет записано значение
    параметра (0 или 1).
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_header_claim_bool(JwtHandle
handle, const char* key, int* value_out);

/**
    * @brief Получает параметр как строку с JSON из заголовка
    проверенного JWT.

```

```

    * @param handle Контекст JWT/JWE.
    * @param key Ключ (имя) параметра заголовка.
    * @param json_value_out Указатель, куда будет записана строка с
JSON значением параметра.
    * Память выделяется библиотекой, освободить через
jwt_free_string().
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_header_claim_json(JwtHandle
handle, const char* key, char** json_value_out);

// --- Функции получения заголовков X.509 ---

/**
 * @brief Получает количество сертификатов в заголовке 'x5c'.
 * @param handle Контекст JWT, полученный после успешного парсинга
(любой функцией jwt_load_*).
 * @return Количество сертификатов (>= 0) или -1, если заголовок
'x5c' отсутствует или не является массивом.
 */
ABC_JWT_EXPORT int jwt_get_header_x5c_count(JwtHandle handle);

/**
 * @brief Получает сертификат из заголовка 'x5c' по индексу в
формате PEM.
 * @param handle Контекст JWT.
 * @param index Индекс сертификата в массиве 'x5c' (начиная с 0).
 * @param pem_cert_out Указатель, куда будет записана строка с
PEM-сертификатом.
 * Память для строки выделяется библиотекой, ее нужно освободить
с помощью jwt_free_string().
 * @return JWT_SUCCESS или код ошибки (JWT_ERROR_CLAIM_NOT_FOUND,
JWT_ERROR_INVALID_ARG,
 * JWT_ERROR_X5C_INVALID_FORMAT, JWT_ERROR_ALLOC).
 */
ABC_JWT_EXPORT JwtError jwt_get_header_x5c_entry_pem(JwtHandle
handle, int index, char** pem_cert_out);

/**
 * @brief Получает отпечаток сертификата SHA-1 из заголовка 'x5t'
(Base64URL).
 * @param handle Контекст JWT.
 * @param thumbprint_out Указатель, куда будет записана строка с
отпечатком.
 * Память для строки выделяется библиотекой, ее нужно освободить
с помощью jwt_free_string().
 * @return JWT_SUCCESS или код ошибки.
 */
ABC_JWT_EXPORT JwtError jwt_get_header_x5t(JwtHandle handle,
char** thumbprint_out);

/**

```

```

    * @brief Получает отпечаток сертификата SHA-256 из заголовка
    'x5t#S256' (Base64URL).
    * @param handle Контекст JWT.
    * @param thumbprint_out Указатель, куда будет записана строка с
    отпечатком.
    * Память для строки выделяется библиотекой, ее нужно освободить
    с помощью jwt_free_string().
    * @return JWT_SUCCESS или код ошибки.
    */
ABC_JWT_EXPORT JwtError jwt_get_header_x5t_s256(JwtHandle handle,
char** thumbprint_out);

#ifdef __cplusplus
} // extern "C"
#endif

#endif // ABC_JWT_H

```

Листинг А.2 – Файл abc_jwt.cpp

```

#include "abc_jwt.h"
#include "jwt_core.h"
#include "key_manager.h"
#include "jws_crypto.h"
#include "utils.h"

#include <vector>
#include <cstring>
#include <memory>
#include <string>
#include <sstream>
#include <limits>
#include <chrono>
#include <set>
#include <nlohmann/json.hpp>
#include <openssl/evp.h>

struct JwtContextInternal {
    std::unique_ptr<jwt_cpp::JwtToken> token;
    unsigned int validation_leeway = 0;
    std::string expected_issuer;
    std::set<std::string> expected_audiences;
    std::set<std::string> understood_crit_headers;
};

struct JwtKeyInternal {
    std::unique_ptr<jwt_cpp::ManagedKey> key;
};

struct KeySetInternal {
    std::unique_ptr<jwt_cpp::KeySet> keyset;
    bool enforce_use = false;
    bool enforce_key_ops = false;
};

```

```

    bool enforce_alg = false;
};

extern "C" {

ABC_JWT_EXPORT JwtError jwt_create(JwtHandle* handle_out) {
    if (!handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        auto internal_handle = new JwtContextInternal();
        internal_handle->token =
std::make_unique<jwt_cpp::JwtToken>();
        *handle_out = internal_handle;
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT void jwt_destroy(JwtHandle handle) {
    if (handle) {
        delete handle;
    }
}

ABC_JWT_EXPORT JwtError jwt_keyset_load_jwks(const char*
jwks_json_str, int ignore_load_errors, KeySetHandle*
keyset_handle_out) {
    if (!jwks_json_str || !keyset_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *keyset_handle_out = nullptr;

    try {
        nlohmann::json jwks_json =
nlohmann::json::parse(jwks_json_str, nullptr, false, true);
        if (jwks_json.is_discarded()) {
            return JWT_ERROR_JSON;
        }

        auto key_set = jwt_cpp::KeySet::load_jwks(jwks_json,
(ignore_load_errors != 0));

        if (!key_set) {
            return JWT_ERROR_KEY_LOAD;
        }

        auto internal_keyset_handle = new KeySetInternal();
        internal_keyset_handle->keyset = std::move(key_set);
    }
}

```

```

        *keyset_handle_out = internal_keyset_handle;
        return JWT_SUCCESS;

    }
    catch (const nlohmann::json::parse_error& e) {
        return JWT_ERROR_JSON;
    }
    catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
    catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_keyset_export_jwks(KeySetHandle
ks_handle, int include_private, char** jwks_json_out) {
    if (!ks_handle || !ks_handle->keyset || !jwks_json_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *jwks_json_out = nullptr;

    try {
        nlohmann::json jwks_obj = ks_handle->keyset-
>export_jwks(include_private != 0);

        if (jwks_obj.is_null() || !jwks_obj.contains("keys")) {
            return JWT_ERROR_KEY_EXPORT;
        }

        std::string jwks_str = jwks_obj.dump(2);

        *jwks_json_out = strdup(jwks_str.c_str());
        if (!*jwks_json_out) {
            return JWT_ERROR_ALLOC;
        }

        return JWT_SUCCESS;

    } catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_keyset_find_key_by_kid(KeySetHandle
keyset_handle, const char* kid, JwtKeyHandle* key_handle_out) {
    if (!keyset_handle || !keyset_handle->keyset || !kid ||
!key_handle_out) {

```

```

        return JWT_ERROR_INVALID_ARG;
    }
    *key_handle_out = nullptr;

    try {
        const jwt_cpp::JwkInfo* found_jwk_info = keyset_handle-
>keyset->find_key_info_by_kid(kid);

        if (!found_jwk_info || !found_jwk_info->key_material) {
            return JWT_ERROR_KEY_NOT_FOUND_FOR_KID;
        }

        EVP_PKEY* original_pkey = found_jwk_info->key_material-
>get();
        if (!original_pkey) {
            return JWT_ERROR_INTERNAL;
        }

        if (EVP_PKEY_up_ref(original_pkey) != 1) {
            return JWT_ERROR_OPENSSL;
        }

        auto new_managed_key =
jwt_cpp::ManagedKey::create_shared_instance(original_pkey);
        if (!new_managed_key) {
            EVP_PKEY_free(original_pkey);
            return JWT_ERROR_ALLOC;
        }

        auto internal_key_handle = new JwkKeyInternal();
        internal_key_handle->key = std::move(new_managed_key);
        *key_handle_out = internal_key_handle;

        return JWT_SUCCESS;
    }
    catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
    catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_keyset_set_enforce_use(KeySetHandle
handle, int enforce) {
    if (!handle) return JWT_ERROR_INVALID_ARG;
    handle->enforce_use = (enforce != 0);
    return JWT_SUCCESS;
}

ABC_JWT_EXPORT JwtError
jwt_keyset_set_enforce_key_ops(KeySetHandle handle, int enforce) {

```

```

        if (!handle) return JWT_ERROR_INVALID_ARG;
        handle->enforce_key_ops = (enforce != 0);
        return JWT_SUCCESS;
    }

ABC_JWT_EXPORT JwtError jwt_keyset_set_enforce_alg(KeySetHandle
handle, int enforce) {
    if (!handle) return JWT_ERROR_INVALID_ARG;
    handle->enforce_alg = (enforce != 0);
    return JWT_SUCCESS;
}

ABC_JWT_EXPORT void jwt_keyset_destroy(KeySetHandle keyset_handle)
{
    if (keyset_handle) {
        delete keyset_handle;
    }
}

ABC_JWT_EXPORT const char* jwt_keyset_get_key_use(KeySetHandle
keyset_handle, const char* kid) {
    if (!keyset_handle || !keyset_handle->keyset || !kid) {
        return NULL;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info && !info->use.empty()) {
        return info->use.c_str();
    }
    return NULL;
}

ABC_JWT_EXPORT const char* jwt_keyset_get_key_alg(KeySetHandle
keyset_handle, const char* kid) {
    if (!keyset_handle || !keyset_handle->keyset || !kid) {
        return NULL;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info && !info->alg.empty()) {
        return info->alg.c_str();
    }
    return NULL;
}

ABC_JWT_EXPORT int jwt_keyset_get_key_ops_count(KeySetHandle
keyset_handle, const char* kid) {
    if (!keyset_handle || !keyset_handle->keyset || !kid) {
        return -1;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);

```

```

        if (info) {
            return static_cast<int>(info->key_ops.size());
        }
        return -1;
    }

ABC_JWT_EXPORT const char*
jwt_keyset_get_key_op_at_index(KeySetHandle keyset_handle, const
char* kid, int index) {
    if (!keyset_handle || !keyset_handle->keyset || !kid || index
< 0) {
        return NULL;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info && index < static_cast<int>(info->key_ops.size())) {
        auto it = info->key_ops.begin();
        std::advance(it, index);
        if (it != info->key_ops.end()) {
            return it->c_str();
        }
    }
    return NULL;
}

ABC_JWT_EXPORT int jwt_keyset_get_key_x5c_count(KeySetHandle
keyset_handle, const char* kid) {
    if (!keyset_handle || !keyset_handle->keyset || !kid) {
        return -1;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info) {
        return static_cast<int>(info->x5c_pem_strings.size());
    }
    return -1;
}

ABC_JWT_EXPORT const char*
jwt_keyset_get_key_x5c_entry_pem(KeySetHandle keyset_handle, const
char* kid, int index) {
    if (!keyset_handle || !keyset_handle->keyset || !kid || index
< 0) {
        return NULL;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info && index < static_cast<int>(info-
>x5c_pem_strings.size())) {
        return info->x5c_pem_strings[index].c_str();
    }
    return NULL;
}

```

```

}

ABC_JWT_EXPORT const char* jwt_keyset_get_key_x5t(KeySetHandle
keyset_handle, const char* kid) {
    if (!keyset_handle || !keyset_handle->keyset || !kid) {
        return NULL;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info && !info->x5t.empty()) {
        return info->x5t.c_str();
    }
    return NULL;
}

ABC_JWT_EXPORT const char*
jwt_keyset_get_key_x5t_s256(KeySetHandle keyset_handle, const
char* kid) {
    if (!keyset_handle || !keyset_handle->keyset || !kid) {
        return NULL;
    }
    const jwt_cpp::JwkInfo* info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (info && !info->x5t_s256.empty()) {
        return info->x5t_s256.c_str();
    }
    return NULL;
}

ABC_JWT_EXPORT JwtError jwt_key_load_secret(const unsigned char*
secret, size_t secret_len, JwtKeyHandle* key_handle_out) {
    if (!secret || secret_len == 0 || !key_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        std::vector<unsigned char> secret_vec(secret, secret +
secret_len);
        auto managed_key =
jwt_cpp::ManagedKey::load_hmac_secret(secret_vec);

        if (!managed_key) {
            return JWT_ERROR_OPENSSL;
        }

        auto internal_key_handle = new JwtKeyInternal();
        internal_key_handle->key = std::move(managed_key);
        *key_handle_out = internal_key_handle;
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
}

```

```

    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_key_load_private_pem(const char*
pem_data, const char* password, JwtKeyHandle* key_handle_out) {
    if (!pem_data || !key_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        std::string pem_str(pem_data);
        auto managed_key =
jwt_cpp::ManagedKey::load_private_pem(pem_str, password);

        if (!managed_key) {
            return JWT_ERROR_KEY_LOAD;
        }

        auto internal_key_handle = new JwtKeyInternal();
        internal_key_handle->key = std::move(managed_key);
        *key_handle_out = internal_key_handle;
        return JWT_SUCCESS;

    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_key_load_public_pem(const char*
pem_data, JwtKeyHandle* key_handle_out) {
    if (!pem_data || !key_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        std::string pem_str(pem_data);
        auto managed_key =
jwt_cpp::ManagedKey::load_public_pem(pem_str);

        if (!managed_key) {
            return JWT_ERROR_KEY_LOAD;
        }

        auto internal_key_handle = new JwtKeyInternal();
        internal_key_handle->key = std::move(managed_key);
        *key_handle_out = internal_key_handle;
        return JWT_SUCCESS;

    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
}

```

```

    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_key_load_jwk(const char* jwk_json_str,
JwtKeyHandle* key_handle_out) {
    if (!jwk_json_str || !key_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *key_handle_out = nullptr;

    try {
        nlohmann::json jwk = nlohmann::json::parse(jwk_json_str,
        nullptr, false, true);

        if (jwk.is_discarded()) {
            return JWT_ERROR_JSON;
        }

        auto jwk_info = jwt_cpp::JwkInfo::load(jwk);

        if (!jwk_info || !jwk_info->key_material) {
            return JWT_ERROR_KEY_LOAD;
        }

        auto internal_key_handle = new JwtKeyInternal();
        internal_key_handle->key = std::move(jwk_info->
        key_material);
        *key_handle_out = internal_key_handle;
        return JWT_SUCCESS;

    }
    catch (const nlohmann::json::parse_error& e) {
        return JWT_ERROR_JSON;
    }
    catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
    catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_key_export_jwk(JwtKeyHandle
key_handle, int include_private, char** jwk_json_out) {
    if (!key_handle || !key_handle->key || !jwk_json_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *jwk_json_out = nullptr;

    try {

```

```

        nlohmann::json jwk = key_handle->key-
>export_jwk(include_private != 0);

        if (jwk.is_null() || jwk.empty()) {
            return JWT_ERROR_KEY_EXPORT;
        }

        std::string jwk_str = jwk.dump(-1);

        *jwk_json_out = strdup(jwk_str.c_str());
        if (!*jwk_json_out) {
            return JWT_ERROR_ALLOC;
        }

        return JWT_SUCCESS;
    }
    catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
    catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    }
    catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_keyset_export_jwk(KeySetHandle
keyset_handle, const char* kid, int include_private, char**
jwk_json_out) {
    if (!keyset_handle || !keyset_handle->keyset || !kid ||
!jwk_json_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *jwk_json_out = nullptr;

    try {
        const jwt_cpp::JwkInfo* jwk_info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
        if (!jwk_info) {
            return JWT_ERROR_KEY_NOT_FOUND_FOR_KID;
        }

        nlohmann::json jwk = jwk_info->export_json(include_private
!= 0);

        if (jwk.is_null() || jwk.empty()) {
            return JWT_ERROR_KEY_EXPORT;
        }

        std::string jwk_str = jwk.dump(-1);

```

```

        *jwk_json_out = strdup(jwk_str.c_str());
        if (!*jwk_json_out) return JWT_ERROR_ALLOC;

        return JWT_SUCCESS;
    }
    catch (const std::bad_alloc& e) { return JWT_ERROR_ALLOC; }
    catch (const nlohmann::json::exception& e) { return
JWT_ERROR_JSON; }
    catch (...) { return JWT_ERROR_INTERNAL; }
}

ABC_JWT_EXPORT JwtError jwt_key_generate_rsa(int bits,
JwtKeyHandle* key_handle_out) {
    if (!key_handle_out || bits <= 0) {
        return JWT_ERROR_INVALID_ARG;
    }
    *key_handle_out = nullptr;
    try {
        auto managed_key =
jwt_cpp::ManagedKey::generate_rsa(bits);
        if (!managed_key) {
            return JWT_ERROR_KEY_GENERATION;
        }
        auto internal_handle = new JwtKeyInternal();
        internal_handle->key = std::move(managed_key);
        *key_handle_out = internal_handle;
        return JWT_SUCCESS;
    }
    catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
    catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_key_generate_ec(const char*
curve_name, JwtKeyHandle* key_handle_out) {
    if (!curve_name || !key_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *key_handle_out = nullptr;
    int nid = jwt_cpp::jwa_curve_name_to_nid(curve_name);
    if (nid == NID_undef || nid == EVP_PKEY_ED25519 || nid ==
EVP_PKEY_ED448) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto managed_key = jwt_cpp::ManagedKey::generate_ec(nid);
        if (!managed_key) {

```

```

        return JWT_ERROR_KEY_GENERATION;
    }
    auto internal_handle = new JwtKeyInternal();
    internal_handle->key = std::move(managed_key);
    *key_handle_out = internal_handle;
    return JWT_SUCCESS;
}
catch (const std::bad_alloc& e) { return JWT_ERROR_ALLOC; }
catch (...) { return JWT_ERROR_INTERNAL; }
}

ABC_JWT_EXPORT JwtError jwt_key_generate_ed(const char*
curve_name, JwtKeyHandle* key_handle_out) {
    if (!curve_name || !key_handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *key_handle_out = nullptr;
    int pkey_id = jwt_cpp::jwa_curve_name_to_nid(curve_name);
    if (pkey_id != EVP_PKEY_ED25519 && pkey_id != EVP_PKEY_ED448)
    {
        return JWT_ERROR_INVALID_ARG;
    }

#ifdef OPENSSSL_VERSION_NUMBER < 0x10101000L
    return JWT_ERROR_ALG_NOT_SUPPORTED;
#else
    try {
        auto managed_key =
jwt_cpp::ManagedKey::generate_ed(pkey_id);
        if (!managed_key) {
            return JWT_ERROR_KEY_GENERATION;
        }
        auto internal_handle = new JwtKeyInternal();
        internal_handle->key = std::move(managed_key);
        *key_handle_out = internal_handle;
        return JWT_SUCCESS;
    }
    catch (const std::bad_alloc& e) { return JWT_ERROR_ALLOC; }
    catch (...) { return JWT_ERROR_INTERNAL; }
#endif
}

ABC_JWT_EXPORT void jwt_key_destroy(JwtKeyHandle key_handle) {
    if (key_handle) {
        delete key_handle;
    }
}

ABC_JWT_EXPORT const char* jwt_error_str(JwtError error_code) {
    switch (error_code) {
        case JWT_SUCCESS: return "Success";
        case JWT_ERROR_ALLOC: return "Memory allocation failed";
    }
}

```

```

        case JWT_ERROR_INVALID_ARG: return "Invalid argument
provided to function";
        case JWT_ERROR_OPENSSL: return "Internal OpenSSL error";
        case JWT_ERROR_JSON: return "JSON parsing or serialization
error";
        case JWT_ERROR_INVALID_SIGNATURE: return "Invalid
signature";
        case JWT_ERROR_ALG_NOT_SUPPORTED: return "Algorithm not
supported or incompatible with key";
        case JWT_ERROR_KEY_LOAD: return "Key loading error or
invalid key format";
        case JWT_ERROR_CLAIM_NOT_FOUND: return "Requested claim
not found";
        case JWT_ERROR_INVALID_TOKEN_FORMAT: return "Invalid JWT
token format";
        case JWT_ERROR_INTERNAL: return "Internal library error";
        case JWT_ERROR_CLAIM_TYPE_MISMATCH: return "Requested
claim type mismatch";
        case JWT_ERROR_VALIDATION_FAILED: return "Claim validation
failed (general)";
        case JWT_ERROR_VALIDATION_EXP_EXPIRED: return "Token has
expired (exp claim)";
        case JWT_ERROR_VALIDATION_NBF_NOT_YET_VALID: return "Token
not yet valid (nbf claim)";
        case JWT_ERROR_VALIDATION_IAT_INVALID: return "Token issue
time is invalid (iat claim)";
        case JWT_ERROR_VALIDATION_ISS_MISMATCH: return "Token
issuer mismatch (iss claim)";
        case JWT_ERROR_VALIDATION_AUD_MISMATCH: return "Token
audience mismatch (aud claim)";
        case JWT_ERROR_VALIDATION_CLAIM_MISSING: return "Required
validation claim is missing";
        case JWT_ERROR_KEY_CURVE_MISMATCH: return "Key curve is
incompatible with the specified algorithm";
        case JWT_ERROR_KEY_NOT_FOUND_FOR_KID: return "Key for
specified kid not found in keyset";
        case JWT_ERROR_KID_MISSING_OR_INVALID: return "kid missing
or invalid in JWT header";
        case JWT_ERROR_KEY_USE_MISMATCH: return "JWK use parameter
mismatch for verification";
        case JWT_ERROR_KEY_OP_NOT_ALLOWED: return "JWK key_ops
does not allow verification";
        case JWT_ERROR_KEY_ALG_MISMATCH: return "JWT alg does not
match JWK alg parameter";
        case JWT_ERROR_KEY_EXPORT: return "Key export failed or
not supported";
        case JWT_ERROR_X5C_INVALID_FORMAT: return "Invalid x5c
header format (not array, base64/der error)";
        case JWT_ERROR_X5C_KEY_EXTRACTION: return "Failed to
extract public key from x5c certificate";
        case JWT_ERROR_HEADER_CLAIM_INVALID_TYPE: return "Header
claim has invalid type";

```

```

        case JWT_ERROR_KEY_GENERATION: return "Key generation
failed";
        case JWT_ERROR_CRIT_INVALID_FORMAT: return "crit header is
invalid (not array/string, empty)";
        case JWT_ERROR_CRIT_UNKNOWN_HEADER: return "crit header
lists unknown or unprocessed header";
        case JWT_ERROR_CRIT_HEADER_MISSING: return "crit header
lists header not present in JWT";
        case JWT_ERROR_CRIT_STANDARD_HEADER_LISTED: return "crit
header lists standard JOSE header";
        default: return "Unknown error code";
    }
}

ABC_JWT_EXPORT void jwt_free_string(char* str) {
    if (str) {
        free(str);
    }
}

ABC_JWT_EXPORT JwtError jwt_set_alg(JwtHandle handle, const char*
alg) {
    if (!handle || !handle->token || !alg) {
        return JWT_ERROR_INVALID_ARG;
    }
    if (!handle->token->set_alg(alg)) {
        return JWT_ERROR_JSON;
    }
    return JWT_SUCCESS;
}

ABC_JWT_EXPORT JwtError
jwt_add_header_x5c_certificate_data(JwtHandle handle, const
unsigned char* cert_data, size_t cert_len) {
    if (!handle || !handle->token || !cert_data || cert_len == 0)
    {
        return JWT_ERROR_INVALID_ARG;
    }

    if (handle->token->add_x5c_certificate_data(cert_data,
cert_len)) {
        return JWT_SUCCESS;
    } else {
        return JWT_ERROR_X5C_INVALID_FORMAT;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_claim_string(JwtHandle handle,
const char* key, const char* value) {
    if (!handle || !handle->token || !key || !value) {
        return JWT_ERROR_INVALID_ARG;
    }
}

```

```

    try {
        if (!handle->token->set_payload_claim(key, value)) {
            return JWT_ERROR_JSON;
        }
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_claim_int(JwtHandle handle, const
char* key, long long value) {
    if (!handle || !handle->token || !key) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        if (!handle->token->set_payload_claim(key, value)) {
            return JWT_ERROR_JSON;
        }
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_claim_double(JwtHandle handle,
const char* key, double value) {
    if (!handle || !handle->token || !key) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        if (!handle->token->set_payload_claim(key, value)) {
            return JWT_ERROR_JSON;
        }
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

```

```

ABC_JWT_EXPORT JwtError jwt_add_claim_bool(JwtHandle handle, const
char* key, int value) {
    if (!handle || !handle->token || !key) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        if (!handle->token->set_payload_claim(key, bool(value !=
0))) {
            return JWT_ERROR_JSON;
        }
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

```

```

ABC_JWT_EXPORT JwtError jwt_add_claim_json(JwtHandle handle, const
char* key, const char* json_value_str) {
    if (!handle || !handle->token || !key || !json_value_str) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        nlohmann::json json_value =
nlohmann::json::parse(json_value_str, nullptr, true, true);
        if (!handle->token->set_payload_claim(key, json_value)) {
            return JWT_ERROR_JSON;
        }
        return JWT_SUCCESS;
    } catch (const nlohmann::json::parse_error& e) {
        return JWT_ERROR_INVALID_ARG;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

```

```

ABC_JWT_EXPORT JwtError jwt_add_header_string(JwtHandle handle,
const char* key, const char* value) {
    if (!handle || !handle->token || !key || !value) {
        return JWT_ERROR_INVALID_ARG;
    }
    if (is_alg_key(key)) {
        return JWT_ERROR_INVALID_ARG;
    }
}

```

```

    try {
        if (handle->token->set_header_claim(key, value)) {
            return JWT_SUCCESS;
        } else {
            return JWT_ERROR_INTERNAL;
        }
    } catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_header_int(JwtHandle handle, const
char* key, long long value) {
    if (!handle || !handle->token || !key) {
        return JWT_ERROR_INVALID_ARG;
    }
    if (is_alg_key(key)) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        if (handle->token->set_header_claim(key, value)) {
            return JWT_SUCCESS;
        } else {
            return JWT_ERROR_INTERNAL;
        }
    } catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_header_double(JwtHandle handle,
const char* key, double value) {
    if (!handle || !handle->token || !key) {
        return JWT_ERROR_INVALID_ARG;
    }
    if (is_alg_key(key)) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        if (handle->token->set_header_claim(key, value)) {
            return JWT_SUCCESS;
        } else {

```

```

        return JWT_ERROR_INTERNAL;
    }
} catch (const std::bad_alloc&) {
    return JWT_ERROR_ALLOC;
} catch (const nlohmann::json::exception& e) {
    return JWT_ERROR_JSON;
} catch (...) {
    return JWT_ERROR_INTERNAL;
}
}

ABC_JWT_EXPORT JwtError jwt_add_header_bool(JwtHandle handle,
const char* key, int value) {
    if (!handle || !handle->token || !key) {
        return JWT_ERROR_INVALID_ARG;
    }
    if (is_alg_key(key)) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        if (handle->token->set_header_claim(key, bool(value !=
0))) {
            return JWT_SUCCESS;
        } else {
            return JWT_ERROR_INTERNAL;
        }
    } catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_header_json(JwtHandle handle,
const char* key, const char* json_value_str) {
    if (!handle || !handle->token || !key || !json_value_str) {
        return JWT_ERROR_INVALID_ARG;
    }
    if (is_alg_key(key)) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        nlohmann::json json_value =
nlohmann::json::parse(json_value_str, nullptr, true, true);
        if (!handle->token->set_header_claim(key, json_value)) {
            return JWT_ERROR_INTERNAL;
        }
        return JWT_SUCCESS;
    }
}

```

```

    } catch (const nlohmann::json::parse_error& e) {
        return JWT_ERROR_JSON;
    } catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_sign(JwtHandle handle, JwtKeyHandle
key_handle, char** token_out) {
    if (!handle || !handle->token || !key_handle || !key_handle-
>key || !token_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *token_out = nullptr;

    try {
        auto alg_opt = handle->token->get_alg();
        if (!alg_opt) {
            return JWT_ERROR_INVALID_ARG;
        }
        const std::string& algorithm = *alg_opt;

        int pkey_type = EVP_PKEY_base_id(key_handle->key->get());
        bool key_ok = false;
        if (algorithm.rfind("HS", 0) == 0) {
            key_ok = true;
        } else if ((algorithm.rfind("RS", 0) == 0 ||
algorithm.rfind("PS", 0) == 0) && (pkey_type == EVP_PKEY_RSA ||
pkey_type == EVP_PKEY_RSA_PSS)) {
            key_ok = true;
        } else if (algorithm.rfind("ES", 0) == 0 && pkey_type ==
EVP_PKEY_EC) {
            key_ok = true;
        } else if (algorithm == "EdDSA" && (pkey_type ==
EVP_PKEY_ED25519 || pkey_type == EVP_PKEY_ED448)) {
            key_ok = true;
        }

        if (!key_ok) {
            return JWT_ERROR_ALG_NOT_SUPPORTED;
        }

        std::string header_b64 = handle->token-
>serialize_header();
        std::string payload_b64 = handle->token-
>serialize_payload();
        if (header_b64.empty() || payload_b64.empty()) {
            return JWT_ERROR_JSON;
        }
    }
}

```

```

    }

    std::string signing_input = header_b64 + "." +
payload_b64;

    auto signature_opt = jwt_crypto::sign(algorithm,
*key_handle->key, signing_input);

    if (!signature_opt) {
        return JWT_ERROR_OPENSSL;
    }

    std::string signature_b64 =
jwt_utils::base64url_encode(*signature_opt);

    std::string final_token = signing_input + "." +
signature_b64;

    *token_out = strdup(final_token.c_str());
    if (!*token_out) {
        return JWT_ERROR_ALLOC;
    }

    return JWT_SUCCESS;

} catch (const std::bad_alloc& e) {
    return JWT_ERROR_ALLOC;
} catch (...) {
    return JWT_ERROR_INTERNAL;
}
}

ABC_JWT_EXPORT JwtError jwt_load_verify(const char* token_str,
JwtKeyHandle key_handle, JwtHandle* handle_out) {
    if (!token_str || !key_handle || !key_handle->key ||
!handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *handle_out = nullptr;

    try {
        std::string token_cpp_str(token_str);
        auto parts = split_string(token_cpp_str, '.');
        if (parts.size() != 3) {
            return JWT_ERROR_INVALID_TOKEN_FORMAT;
        }
        const std::string& header_b64 = parts[0];
        const std::string& payload_b64 = parts[1];
        const std::string& signature_b64 = parts[2];

        auto internal_handle = new JwtContextInternal();

```

```

        internal_handle->token =
std::make_unique<jwt_cpp::JwtToken>();

        if (!internal_handle->token->parse_header(header_b64)) {
            delete internal_handle;
            return JWT_ERROR_JSON;
        }

        auto alg_opt = internal_handle->token->get_alg();
        if (!alg_opt) {
            delete internal_handle;
            return JWT_ERROR_INVALID_TOKEN_FORMAT;
        }
        const std::string& algorithm = *alg_opt;

        int pkey_type = EVP_PKEY_base_id(key_handle->key->get());
        bool key_ok = false;
        if (algorithm.rfind("HS", 0) == 0) {
            key_ok = true;
        } else if ((algorithm.rfind("RS", 0) == 0 ||
algorithm.rfind("PS", 0) == 0) && (pkey_type == EVP_PKEY_RSA ||
pkey_type == EVP_PKEY_RSA_PSS)) {
            key_ok = true;
        } else if (algorithm.rfind("ES", 0) == 0 && pkey_type ==
EVP_PKEY_EC) {
            key_ok = true;
        } else if (algorithm == "EdDSA" && (pkey_type ==
EVP_PKEY_ED25519 || pkey_type == EVP_PKEY_ED448)) {
            key_ok = true;
        }

        if (!key_ok) {
            delete internal_handle;
            return JWT_ERROR_ALG_NOT_SUPPORTED;
        }

        if (!internal_handle->token->parse_payload(payload_b64))
        {
            delete internal_handle;
            return JWT_ERROR_JSON;
        }

        auto signature_opt =
jwt_utils::base64url_decode(signature_b64);
        if (!signature_opt) {
            delete internal_handle;
            return JWT_ERROR_INVALID_TOKEN_FORMAT;
        }

        std::string data_to_verify = header_b64 + "." +
payload_b64;

```

```

        bool verified = jwt_crypto::verify(algorithm,
*key_handle->key, data_to_verify, *signature_opt);

        if (!verified) {
            delete internal_handle;
            return JWT_ERROR_INVALID_SIGNATURE;
        }

        *handle_out = internal_handle;

        return JWT_SUCCESS;

    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_load_verify_jwks(const char*
token_str, KeySetHandle keyset_handle, JwtHandle* handle_out) {
    if (!token_str || !keyset_handle || !keyset_handle->keyset ||
!handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *handle_out = nullptr;
    JwtKeyHandle verification_key_handle = nullptr;
    JwtError final_error = JWT_ERROR_INTERNAL;
    const jwt_cpp::JwkInfo* found_jwk_info = nullptr;

    try {
        std::string token_cpp_str(token_str);
        auto parts = split_string(token_cpp_str, '.');
        if (parts.size() != 3) {
            final_error = JWT_ERROR_INVALID_TOKEN_FORMAT;
            goto cleanup;
        }

        const std::string& header_b64 = parts[0];
        auto decoded_header_opt =
jwt_utils::base64url_decode_to_string(header_b64);
        if (!decoded_header_opt) {
            final_error = JWT_ERROR_INVALID_TOKEN_FORMAT;
            goto cleanup;
        }

        nlohmann::json header_json;
        try {
            header_json =
nlohmann::json::parse(*decoded_header_opt, nullptr, true, true);

```

```

        if (!header_json.is_object()) {
            final_error = JWT_ERROR_JSON;
            goto cleanup;
        }
    }
    catch (const nlohmann::json::parse_error& e) {
        final_error = JWT_ERROR_JSON;
        goto cleanup;
    } catch (const nlohmann::json::exception& e) {
        final_error = JWT_ERROR_JSON;
        goto cleanup;
    }

    if (!header_json.contains("kid") ||
!header_json.at("kid").is_string()) {
        final_error = JWT_ERROR_KID_MISSING_OR_INVALID;
        goto cleanup;
    }
    std::string kid =
header_json.at("kid").get<std::string>();

    if (!header_json.contains("alg") ||
!header_json.at("alg").is_string()) {
        final_error = JWT_ERROR_INVALID_TOKEN_FORMAT;
        goto cleanup;
    }
    std::string jwt_alg =
header_json.at("alg").get<std::string>();

    found_jwk_info = keyset_handle->keyset-
>find_key_info_by_kid(kid);
    if (!found_jwk_info || !found_jwk_info->key_material) {
        final_error = JWT_ERROR_KEY_NOT_FOUND_FOR_KID;
        goto cleanup;
    }

    if (keyset_handle->enforce_use && !found_jwk_info-
>use.empty()) {
        if (found_jwk_info->use != "sig") {
            final_error = JWT_ERROR_KEY_USE_MISMATCH;
            goto cleanup;
        }
    }

    if (keyset_handle->enforce_key_ops && !found_jwk_info-
>key_ops.empty()) {
        if (found_jwk_info->key_ops.find("verify") ==
found_jwk_info->key_ops.end()) {
            final_error = JWT_ERROR_KEY_OP_NOT_ALLOWED;
            goto cleanup;
        }
    }
}

```

```

        if (keyset_handle->enforce_alg && !found_jwk_info-
>alg.empty()) {
            if (found_jwk_info->alg != jwt_alg) {
                final_error = JWT_ERROR_KEY_ALG_MISMATCH;
                goto cleanup;
            }
        }

        JwtError find_err =
jwt_keyset_find_key_by_kid(keyset_handle, kid.c_str(),
&verification_key_handle);
        if (find_err != JWT_SUCCESS) {
            final_error = find_err;
            goto cleanup;
        }

        final_error = jwt_load_verify(token_str,
verification_key_handle, handle_out);

    }
    catch (const std::bad_alloc& e) {
        final_error = JWT_ERROR_ALLOC;
    } catch (...) {
        final_error = JWT_ERROR_INTERNAL;
    }

cleanup:
    if (verification_key_handle) {
        jwt_key_destroy(verification_key_handle);
    }
    return final_error;
}

JwtError jwt_load_verify_x5c(const char* token_str, JwtHandle*
handle_out) {
    if (!token_str || !handle_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *handle_out = nullptr;
    JwtKeyHandle verification_key_handle = nullptr;
    JwtError final_error = JWT_ERROR_INTERNAL;

    auto cleanup_key = [&]() {
        if (verification_key_handle)
jwt_key_destroy(verification_key_handle);
    };

    try {
        std::string token_cpp_str(token_str);
        auto parts = split_string(token_cpp_str, '.');
        if (parts.size() != 3) {
            final_error = JWT_ERROR_INVALID_TOKEN_FORMAT;

```

```

        goto cleanup;
    }

    const std::string& header_b64 = parts[0];
    auto decoded_header_opt =
jwt_utils::base64url_decode_to_string(header_b64);
    if (!decoded_header_opt) {
        final_error = JWT_ERROR_INVALID_TOKEN_FORMAT;
        goto cleanup;
    }

    nlohmann::json header_json;
    try {
        header_json =
nlohmann::json::parse(*decoded_header_opt, nullptr, true, true);
        if (!header_json.is_object()) {
            final_error = JWT_ERROR_JSON;
            goto cleanup;
        }
    }
    catch (const nlohmann::json::parse_error& e) {
        final_error = JWT_ERROR_JSON;
        goto cleanup;
    }
    catch (const nlohmann::json::exception& e) {
        final_error = JWT_ERROR_JSON;
        goto cleanup;
    }
    }

    if (!header_json.contains("x5c") ||
!header_json.at("x5c").is_array() ||
header_json.at("x5c").empty()) {
        final_error = JWT_ERROR_X5C_INVALID_FORMAT;
        goto cleanup;
    }

    const auto& x5c_array = header_json.at("x5c");
    if (!x5c_array[0].is_string()) {
        final_error = JWT_ERROR_X5C_INVALID_FORMAT;
        goto cleanup;
    }

    std::string cert_b64 = x5c_array[0].get<std::string>();

    auto der_bytes_opt = jwt_utils::base64_decode(cert_b64);
    if (!der_bytes_opt) {
        final_error = JWT_ERROR_X5C_INVALID_FORMAT;
        goto cleanup;
    }

    auto x509_cert =
jwt_utils::parse_der_certificate(*der_bytes_opt);
    if (!x509_cert) {
        final_error = JWT_ERROR_X5C_INVALID_FORMAT;
        goto cleanup;
    }

```

```

    }

    auto pub_key =
jwt_utils::get_pkey_from_cert(x509_cert.get());
    if (!pub_key) {
        final_error = JWT_ERROR_X5C_KEY_EXTRACTION;
        goto cleanup;
    }

    auto internal_key_handle = new JwtKeyInternal();
    internal_key_handle->key =
jwt_cpp::ManagedKey::create_shared_instance(pub_key.release());
    verification_key_handle = internal_key_handle;

    final_error = jwt_load_verify(token_str,
verification_key_handle, handle_out);

    }
    catch (const std::bad_alloc& e) {
        final_error = JWT_ERROR_ALLOC;
    } catch (...) {
        final_error = JWT_ERROR_INTERNAL;
    }
}

cleanup:
    cleanup_key();
    return final_error;
}

ABC_JWT_EXPORT JwtError jwt_set_validation_leeway(JwtHandle
handle, unsigned int leeway_seconds) {
    if (!handle) {
        return JWT_ERROR_INVALID_ARG;
    }
    handle->validation_leeway = leeway_seconds;
    return JWT_SUCCESS;
}

ABC_JWT_EXPORT JwtError jwt_set_expected_issuer(JwtHandle handle,
const char* issuer) {
    if (!handle) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        if (issuer) {
            handle->expected_issuer = issuer;
        } else {
            handle->expected_issuer.clear();
        }
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
}

```

```

        } catch (...) {
            return JWT_ERROR_INTERNAL;
        }
    }

ABC_JWT_EXPORT JwtError jwt_add_expected_audience(JwtHandle
handle, const char* audience) {
    if (!handle || !audience) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        handle->expected_audiences.insert(audience);
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_validate_claims(JwtHandle handle) {
    if (!handle || !handle->token) {
        return JWT_ERROR_INVALID_ARG;
    }

    const auto& token = handle->token;
    const auto& payload = token->get_payload();
    unsigned int leeway = handle->validation_leeway;

    try {
        auto now_tp = std::chrono::system_clock::now();
        double current_time =
std::chrono::duration<double>(now_tp.time_since_epoch()).count();

        auto exp_opt = get_numeric_claim_as_double(token.get(),
"exp");
        if (exp_opt) {
            if (*exp_opt + leeway < current_time) {
                return JWT_ERROR_VALIDATION_EXP_EXPIRED;
            }
        }

        auto nbf_opt = get_numeric_claim_as_double(token.get(),
"nbf");
        if (nbf_opt) {
            if (*nbf_opt - leeway > current_time) {
                return JWT_ERROR_VALIDATION_NBF_NOT_YET_VALID;
            }
        }

        auto iat_opt = get_numeric_claim_as_double(token.get(),
"iat");

```

```

    if (iat_opt) {
        if (*iat_opt - leeway > current_time) {
            return JWT_ERROR_VALIDATION_IAT_INVALID;
        }
    }

    if (!handle->expected_issuer.empty()) {
        auto iss_claim_opt = token->get_payload_claim("iss");
        if (!iss_claim_opt) return
JWT_ERROR_VALIDATION_CLAIM_MISSING;
        if (!iss_claim_opt->is_string()) return
JWT_ERROR_CLAIM_TYPE_MISMATCH;
        if (iss_claim_opt->get<std::string>() != handle-
>expected_issuer) {
            return JWT_ERROR_VALIDATION_ISS_MISMATCH;
        }
    }

    if (!handle->expected_audiences.empty()) {
        auto aud_claim_opt = token->get_payload_claim("aud");
        if (!aud_claim_opt) return
JWT_ERROR_VALIDATION_CLAIM_MISSING;

        bool audience_match_found = false;
        if (aud_claim_opt->is_string()) {
            if (handle-
>expected_audiences.count(aud_claim_opt->get<std::string>())) {
                audience_match_found = true;
            }
        } else if (aud_claim_opt->is_array()) {
            for (const auto& aud_item : *aud_claim_opt) {
                if (aud_item.is_string()) {
                    if (handle-
>expected_audiences.count(aud_item.get<std::string>())) {
                        audience_match_found = true;
                        break;
                    }
                } else {
                    return JWT_ERROR_CLAIM_TYPE_MISMATCH;
                }
            }
        } else {
            return JWT_ERROR_CLAIM_TYPE_MISMATCH;
        }

        if (!audience_match_found) {
            return JWT_ERROR_VALIDATION_AUD_MISMATCH;
        }
    }

    return JWT_SUCCESS;

```

```

    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_CLAIM_TYPE_MISMATCH;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_add_understood_crit_header(JwtHandle
handle, const char* header_name) {
    if (!handle || !header_name || *header_name == '\\0') {
        return JWT_ERROR_INVALID_ARG;
    }
    if (jwt_cpp::STANDARD_JOSE_HEADERS.count(header_name)) {
        return JWT_ERROR_INVALID_ARG;
    }
    try {
        handle->understood_crit_headers.insert(header_name);
        return JWT_SUCCESS;
    }
    catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    }
    catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_validate_crit_headers(JwtHandle
handle) {
    if (!handle || !handle->token) {
        return JWT_ERROR_INVALID_ARG;
    }

    int error_code_int = JWT_SUCCESS;
    bool success = handle->token->validate_crit_headers(
        handle->understood_crit_headers,
        error_code_int
    );

    if (success) {
        return JWT_SUCCESS;
    }
    else {
        if (error_code_int >= 0) {
            return JWT_ERROR_CRIT_UNKNOWN_HEADER;
        }
        return static_cast<JwtError>(error_code_int);
    }
}

ABC_JWT_EXPORT JwtError jwt_get_payload_claim_string(JwtHandle
handle, const char* key, char** value_out) {

```

```

    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *value_out = nullptr;

    try {
        auto claim_opt = handle->token->get_payload_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }

        if (!claim_opt->is_string()) {
            return JWT_ERROR_INVALID_ARG;
        }

        std::string value_str = claim_opt->get<std::string>();
        *value_out = strdup(value_str.c_str());
        if (!*value_out) {
            return JWT_ERROR_ALLOC;
        }
        return JWT_SUCCESS;
    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_payload_claim_int(JwtHandle
handle, const char* key, long long* value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto claim_opt = handle->token->get_payload_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }

        if (!claim_opt->is_number_integer() && !claim_opt-
>is_number_unsigned()) {
            if (!claim_opt->is_number_float() || claim_opt-
>get<double>() != floor(claim_opt->get<double>()) ||
                claim_opt->get<double>() <
static_cast<double>(std::numeric_limits<long long>::min()) ||
                claim_opt->get<double>() >
static_cast<double>(std::numeric_limits<long long>::max()))
            {
                return JWT_ERROR_CLAIM_TYPE_MISMATCH;
            }
        }
    }
}

```

```

        }
    }

    *value_out = claim_opt->get<long long>();
    return JWT_SUCCESS;

} catch (const nlohmann::json::exception& e) {
    return JWT_ERROR_CLAIM_TYPE_MISMATCH;
} catch (...) {
    return JWT_ERROR_INTERNAL;
}
}

ABC_JWT_EXPORT JwtError jwt_get_payload_claim_double(JwtHandle
handle, const char* key, double* value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto claim_opt = handle->token->get_payload_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }
        if (!claim_opt->is_number()) {
            return JWT_ERROR_CLAIM_TYPE_MISMATCH;
        }

        *value_out = claim_opt->get<double>();
        return JWT_SUCCESS;

    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_CLAIM_TYPE_MISMATCH;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_payload_claim_bool(JwtHandle
handle, const char* key, int* value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto claim_opt = handle->token->get_payload_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }
        if (!claim_opt->is_boolean()) {
            return JWT_ERROR_CLAIM_TYPE_MISMATCH;
        }
    }
}

```

```

        *value_out = claim_opt->get<bool>() ? 1 : 0;
        return JWT_SUCCESS;

    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_CLAIM_TYPE_MISMATCH;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_payload_claim_json(JwtHandle
handle, const char* key, char** json_value_out) {
    if (!handle || !handle->token || !key || !json_value_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *json_value_out = nullptr;

    try {
        auto claim_opt = handle->token->get_payload_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }

        std::string json_str = claim_opt->dump(-1, ' ', false,
nlohmann::json::error_handler_t::replace);

        *json_value_out = strdup(json_str.c_str());
        if (!*json_value_out) {
            return JWT_ERROR_ALLOC;
        }
        return JWT_SUCCESS;

    } catch (const std::bad_alloc& e) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_header_claim_string(JwtHandle
handle, const char* key, char** value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    *value_out = nullptr;

    try {
        auto claim_opt = handle->token->get_header_claim(key);

```

```

        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }
        if (!claim_opt->is_string()) {
            return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
        }

        std::string value_str = claim_opt->get<std::string>();
        *value_out = strdup(value_str.c_str());
        if (!*value_out) {
            return JWT_ERROR_ALLOC;
        }
        return JWT_SUCCESS;
    }

    catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_header_claim_int(JwtHandle handle,
const char* key, long long* value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto claim_opt = handle->token->get_header_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }
        if (!claim_opt->is_number()) {
            return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
        }

        if (claim_opt->is_number_integer() || claim_opt->
is_number_unsigned()) {
            if (claim_opt->is_number_unsigned()) {
                uint64_t uval = claim_opt->get<uint64_t>();
                if (uval >
static_cast<uint64_t>(std::numeric_limits<long long>::max())) {
                    return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
                }
            }
            *value_out = claim_opt->get<long long>();
            return JWT_SUCCESS;
        } else if (claim_opt->is_number_float()) {
            double dval = claim_opt->get<double>();

```

```

        if (dval != floor(dval) || dval <
static_cast<double>(std::numeric_limits<long long>::min()) || dval
> static_cast<double>(std::numeric_limits<long long>::max())) {
            return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
        }
        *value_out = static_cast<long long>(dval);
        return JWT_SUCCESS;
    } else {
        return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
    }
}

}
catch (const nlohmann::json::exception& e) {
    return JWT_ERROR_JSON;
} catch (...) {
    return JWT_ERROR_INTERNAL;
}
}

ABC_JWT_EXPORT JwtError jwt_get_header_claim_double(JwtHandle
handle, const char* key, double* value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto claim_opt = handle->token->get_header_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }
        if (!claim_opt->is_number()) {
            return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
        }

        *value_out = claim_opt->get<double>();
        return JWT_SUCCESS;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_header_claim_bool(JwtHandle
handle, const char* key, int* value_out) {
    if (!handle || !handle->token || !key || !value_out) {
        return JWT_ERROR_INVALID_ARG;
    }

    try {
        auto claim_opt = handle->token->get_header_claim(key);

```

```

        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }
        if (!claim_opt->is_boolean()) {
            return JWT_ERROR_HEADER_CLAIM_INVALID_TYPE;
        }

        *value_out = claim_opt->get<bool>() ? 1 : 0;
        return JWT_SUCCESS;

    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT JwtError jwt_get_header_claim_json(JwtHandle
handle, const char* key, char** json_value_out) {
    if (!handle || !handle->token || !key || !json_value_out) {
        return JWT_ERROR_INVALID_ARG;
    }
    *json_value_out = nullptr;

    try {
        auto claim_opt = handle->token->get_header_claim(key);
        if (!claim_opt) {
            return JWT_ERROR_CLAIM_NOT_FOUND;
        }

        std::string json_str = claim_opt->dump(-1);

        *json_value_out = strdup(json_str.c_str());
        if (!*json_value_out) {
            return JWT_ERROR_ALLOC;
        }
        return JWT_SUCCESS;

    } catch (const std::bad_alloc&) {
        return JWT_ERROR_ALLOC;
    } catch (const nlohmann::json::exception& e) {
        return JWT_ERROR_JSON;
    } catch (...) {
        return JWT_ERROR_INTERNAL;
    }
}

ABC_JWT_EXPORT int jwt_get_header_x5c_count(JwtHandle handle) {
    const nlohmann::json* x5c_claim =
get_header_claim_json(handle, "x5c");
    if (x5c_claim && x5c_claim->is_array()) {
        return static_cast<int>(x5c_claim->size());
    }
}

```

```

    }
    return -1;
}

ABC_JWT_EXPORT JwtError jwt_get_header_x5c_entry_pem(JwtHandle
handle, int index, char** pem_cert_out) {
    if (!pem_cert_out) return JWT_ERROR_INVALID_ARG;
    *pem_cert_out = nullptr;

    const nlohmann::json* x5c_claim =
get_header_claim_json(handle, "x5c");
    if (!x5c_claim || !x5c_claim->is_array()) {
        return JWT_ERROR_CLAIM_NOT_FOUND;
    }
    if (index < 0 || index >= static_cast<int>(x5c_claim->size()))
{
        return JWT_ERROR_INVALID_ARG;
    }
    if (!(*x5c_claim)[index].is_string()) {
        return JWT_ERROR_X5C_INVALID_FORMAT;
    }

    std::string cert_b64 = (*x5c_claim)[index].get<std::string>();

    auto der_bytes_opt = jwt_utils::base64_decode(cert_b64);
    if (!der_bytes_opt) return JWT_ERROR_X5C_INVALID_FORMAT;

    auto x509_cert =
jwt_utils::parse_der_certificate(*der_bytes_opt);
    if (!x509_cert) return JWT_ERROR_X5C_INVALID_FORMAT;

    auto pem_str_opt = jwt_utils::cert_to_pem(x509_cert.get());
    if (!pem_str_opt) return JWT_ERROR_INTERNAL;

    *pem_cert_out = strdup(pem_str_opt->c_str());
    if (!*pem_cert_out) return JWT_ERROR_ALLOC;

    return JWT_SUCCESS;
}

ABC_JWT_EXPORT JwtError jwt_get_header_x5t(JwtHandle handle,
char** thumbprint_out) {
    return get_header_claim_b64url_string(handle, "x5t",
thumbprint_out);
}

ABC_JWT_EXPORT JwtError jwt_get_header_x5t_s256(JwtHandle handle,
char** thumbprint_out) {
    return get_header_claim_b64url_string(handle, "x5t#S256",
thumbprint_out);
}

```

```
}
```

Листинг А.3 – Файл jws_crypto.cpp

```
#include "jws_crypto.h"
#include "utils.h"

#include <openssl/hmac.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/crypto.h>
#include <openssl/rsa.h>
#include <openssl/asn1.h>
#include <openssl/obj_mac.h>

#include <map>
#include <stdexcept>
#include <vector>
#include <memory>

#if OPENSSL_VERSION_NUMBER < 0x10101000L
#error "OpenSSL 1.1.1 or later is required for EdDSA support"
#endif

namespace jwt_crypto {

const EVP_MD* get_evp_md_from_jwa(const std::string& algorithm) {
    static const std::map<std::string, const EVP_MD* (*)()>
    alg_to_md_map = {
        {"HS256", EVP_sha256}, {"HS384", EVP_sha384}, {"HS512",
EVP_sha512},
        {"RS256", EVP_sha256}, {"RS384", EVP_sha384}, {"RS512",
EVP_sha512},
        {"ES256", EVP_sha256}, {"ES384", EVP_sha384}, {"ES512",
EVP_sha512}, {"ES256K", EVP_sha256},
        {"PS256", EVP_sha256}, {"PS384", EVP_sha384}, {"PS512",
EVP_sha512},
        {"EdDSA", nullptr}
    };

    auto it = alg_to_md_map.find(algorithm);
    if (it != alg_to_md_map.end()) {
        if (it->second) {
            return it->second();
        } else {
            return nullptr;
        }
    }
    return nullptr;
}

int get_ecdsa_rs_length(const EVP_PKEY* pkey) {
    const EC_KEY *ec_key = EVP_PKEY_get0_EC_KEY(pkey);
```

```

    if (!ec_key) return 0;
    const EC_GROUP *group = EC_KEY_get0_group(ec_key);
    if (!group) return 0;
    int degree_bits = EC_GROUP_get_degree(group);
    return (degree_bits + 7) / 8;
}

std::optional<std::vector<unsigned char>>
der_to_raw_ecdsa_signature(
    const unsigned char* der_sig,
    size_t der_sig_len,
    int expected_rs_len)
{
    if (expected_rs_len <= 0) return std::nullopt;

    const unsigned char* p = der_sig;
    std::unique_ptr<ECDSA_SIG, ECDSA_SIGDeleter> sig(
        d2i_ECDSA_SIG(nullptr, &p, static_cast<long>(der_sig_len))
    );
    if (!sig) return std::nullopt;

    const BIGNUM *r = ECDSA_SIG_get0_r(sig.get());
    const BIGNUM *s = ECDSA_SIG_get0_s(sig.get());
    if (!r || !s) return std::nullopt;

    std::vector<unsigned char> raw_sig(expected_rs_len * 2);

    int len_r = BN_bn2binpad(r, raw_sig.data(), expected_rs_len);
    int len_s = BN_bn2binpad(s, raw_sig.data() + expected_rs_len,
expected_rs_len);

    if (len_r != expected_rs_len || len_s != expected_rs_len) {
        return std::nullopt;
    }

    return raw_sig;
}

std::optional<std::vector<unsigned char>>
raw_to_der_ecdsa_signature(
    const unsigned char* raw_sig,
    size_t raw_sig_len)
{
    if (raw_sig_len == 0 || raw_sig_len % 2 != 0) return
std::nullopt;
    int rs_len = static_cast<int>(raw_sig_len / 2);

    std::unique_ptr<BIGNUM, BIGNUMDeleter> r(BN_bin2bn(raw_sig,
rs_len, nullptr));
    std::unique_ptr<BIGNUM, BIGNUMDeleter> s(BN_bin2bn(raw_sig +
rs_len, rs_len, nullptr));
    if (!r || !s) return std::nullopt;

```

```

        std::unique_ptr<ECDSA_SIG, ECDSA_SIGDeleter>
sig(ECDSA_SIG_new());
    if (!sig) return std::nullopt;

    if (ECDSA_SIG_set0(sig.get(), r.release(), s.release()) != 1)
    {
        sig.reset();
        return std::nullopt;
    }

    int der_len = i2d_ECDSA_SIG(sig.get(), nullptr);
    if (der_len <= 0) return std::nullopt;

    std::vector<unsigned char> der_sig(der_len);
    unsigned char* p = der_sig.data();
    der_len = i2d_ECDSA_SIG(sig.get(), &p);
    if (der_len <= 0) return std::nullopt;

    der_sig.resize(der_len);
    return der_sig;
}

bool check_ec_curve(const EVP_PKEY* pkey, int expected_nid) {
    const EC_KEY *ec_key = EVP_PKEY_get0_EC_KEY(pkey);
    if (!ec_key) return false;
    const EC_GROUP *group = EC_KEY_get0_group(ec_key);
    if (!group) return false;
    int curve_nid = EC_GROUP_get_curve_name(group);
    return (curve_nid == expected_nid);
}

std::optional<std::vector<unsigned char>> sign_hmac(
    const EVP_MD* md_type,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_sign)
{
    if (!md_type) return std::nullopt;

    EVP_PKEY* pkey = key.get();
    if (!pkey) {
        return std::nullopt;
    }

    unsigned int signature_len = EVP_MAX_MD_SIZE;
    std::vector<unsigned char> signature(signature_len);

    std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
    if (!ctx) {
        return std::nullopt;
    }

```

```

        if (EVP_DigestSignInit(ctx.get(), nullptr, md_type, nullptr,
pkey) <= 0) {
            return std::nullopt;
        }

        if (EVP_DigestSignUpdate(ctx.get(), data_to_sign.data(),
data_to_sign.size()) <= 0) {
            return std::nullopt;
        }

        size_t req_len = 0;
        if (EVP_DigestSignFinal(ctx.get(), nullptr, &req_len) <= 0) {
            return std::nullopt;
        }

        signature.resize(req_len);

        if (EVP_DigestSignFinal(ctx.get(), signature.data(), &req_len)
<= 0) {
            return std::nullopt;
        }

        signature.resize(req_len);

        return signature;
    }

bool verify_hmac(
    const EVP_MD* md_type,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_verify,
    const std::vector<unsigned char>& signature)
{
    auto calculated_signature_opt = sign_hmac(md_type, key,
data_to_verify);
    if (!calculated_signature_opt) {
        return false;
    }

    const auto& calculated_signature = *calculated_signature_opt;

    return calculated_signature.size() == signature.size() &&
        CRYPTO_memcmp(calculated_signature.data(),
signature.data(), signature.size()) == 0;
}

std::optional<std::vector<unsigned char>> sign_rsa(
    const EVP_MD* md_type,
    int padding_type,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_sign)

```

```

{
    if (!md_type) {
        return std::nullopt;
    }

    EVP_PKEY* pkey = key.get();
    int pkey_type = EVP_PKEY_base_id(pkey);
    if (!pkey || (pkey_type != EVP_PKEY_RSA && pkey_type !=
EVP_PKEY_RSA_PSS)) {
        return std::nullopt;
    }

    std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
    if (!ctx) {
        return std::nullopt;
    }

    EVP_PKEY_CTX *pctx = nullptr;

    if (EVP_DigestSignInit(ctx.get(), &pctx, md_type, nullptr,
pkey) <= 0) {
        return std::nullopt;
    }

    if (EVP_PKEY_CTX_set_rsa_padding(pctx, padding_type) <= 0) {
        return std::nullopt;
    }

    if (padding_type == RSA_PKCS1_PSS_PADDING) {
        if (EVP_PKEY_CTX_set_rsa_mgf1_md(pctx, md_type) <= 0) {
            return std::nullopt;
        }
        if (EVP_PKEY_CTX_set_rsa_pss_saltlen(pctx, -1) <= 0) {
            return std::nullopt;
        }
    }
    if (EVP_PKEY_CTX_set_signature_md(pctx, md_type) <= 0) {
        return std::nullopt;
    }

    if (EVP_DigestSignUpdate(ctx.get(), data_to_sign.data(),
data_to_sign.size()) <= 0) { return std::nullopt; }
    size_t signature_len = 0;
    if (EVP_DigestSignFinal(ctx.get(), nullptr, &signature_len) <=
0) { return std::nullopt; }
    std::vector<unsigned char> signature(signature_len);
    if (EVP_DigestSignFinal(ctx.get(), signature.data(),
&signature_len) <= 0) { return std::nullopt; }
    signature.resize(signature_len);

    return signature;
}

```

```

}

bool verify_rsa(
    const EVP_MD* md_type,
    int padding_type,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_verify,
    const std::vector<unsigned char>& signature)
{
    if (!md_type) return false;

    EVP_PKEY* pkey = key.get();
    int pkey_type = EVP_PKEY_base_id(pkey);
    if (!pkey || (pkey_type != EVP_PKEY_RSA && pkey_type !=
EVP_PKEY_RSA_PSS)) {
        return false;
    }

    std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
    if (!ctx) {
        return false;
    }

    EVP_PKEY_CTX *pctx = nullptr;

    if (EVP_DigestVerifyInit(ctx.get(), &pctx, md_type, nullptr,
pkey) <= 0) {
        return false;
    }

    if (EVP_PKEY_CTX_set_rsa_padding(pctx, padding_type) <= 0) {
        return false;
    }

    if (padding_type == RSA_PKCS1_PSS_PADDING) {
        if (EVP_PKEY_CTX_set_rsa_mgf1_md(pctx, md_type) <= 0) {
            return false;
        }
        if (EVP_PKEY_CTX_set_rsa_pss_saltlen(pctx, -1) <= 0) {
            return false;
        }
    }

    if (EVP_PKEY_CTX_set_signature_md(pctx, md_type) <= 0) {
        return false;
    }

    if (EVP_DigestVerifyUpdate(ctx.get(), data_to_verify.data(),
data_to_verify.size()) <= 0) {
        return false;
    }
}

```

```

    int verify_result = EVP_DigestVerifyFinal(ctx.get(),
signature.data(), signature.size());

    return (verify_result == 1);
}

std::optional<std::vector<unsigned char>> sign_ecdsa(
    const std::string& algorithm,
    const EVP_MD* md_type,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_sign)
{
    if (!md_type) return std::nullopt;

    EVP_PKEY* pkey = key.get();
    if (!pkey || EVP_PKEY_base_id(pkey) != EVP_PKEY_EC) {
        return std::nullopt;
    }

    int expected_nid = NID_undef;
    if (algorithm == "ES256") expected_nid = NID_X9_62_prime256v1;
    else if (algorithm == "ES256K") expected_nid = NID_secp256k1;
    else if (algorithm == "ES384") expected_nid = NID_secp384r1;
    else if (algorithm == "ES512") expected_nid = NID_secp521r1;

    if (expected_nid != NID_undef && !check_ec_curve(pkey,
expected_nid)) {
        return std::nullopt;
    }

    int rs_len = get_ecdsa_rs_length(pkey);
    if (rs_len <= 0) return std::nullopt;

    std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
    if (!ctx) return std::nullopt;

    if (EVP_DigestSignInit(ctx.get(), nullptr, md_type, nullptr,
pkey) <= 0) {
        return std::nullopt;
    }

    if (EVP_DigestSignUpdate(ctx.get(), data_to_sign.data(),
data_to_sign.size()) <= 0) {
        return std::nullopt;
    }

    size_t der_sig_len = 0;
    if (EVP_DigestSignFinal(ctx.get(), nullptr, &der_sig_len) <=
0) {
        return std::nullopt;
    }
}

```

```

        std::vector<unsigned char> der_signature(der_sig_len);

        if (EVP_DigestSignFinal(ctx.get(), der_signature.data(),
&der_sig_len) <= 0) {
            return std::nullopt;
        }
        der_signature.resize(der_sig_len);

        return der_to_raw_ecdsa_signature(der_signature.data(),
der_signature.size(), rs_len);
    }

bool verify_ecdsa(
    const std::string& algorithm,
    const EVP_MD* md_type,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_verify,
    const std::vector<unsigned char>& signature)
{
    if (!md_type) return false;

    EVP_PKEY* pkey = key.get();
    if (!pkey || EVP_PKEY_base_id(pkey) != EVP_PKEY_EC) {
        return false;
    }

    int expected_nid = NID_undef;
    if (algorithm == "ES256") expected_nid = NID_X9_62_prime256v1;
    else if (algorithm == "ES256K") expected_nid = NID_secp256k1;
    else if (algorithm == "ES384") expected_nid = NID_secp384r1;
    else if (algorithm == "ES512") expected_nid = NID_secp521r1;

    if (expected_nid != NID_undef && !check_ec_curve(pkey,
expected_nid)) {
        return false;
    }

    int expected_rs_len = get_ecdsa_rs_length(pkey);
    if (expected_rs_len <= 0 || signature.size() !=
static_cast<size_t>(2 * expected_rs_len)) {
        return false;
    }

    auto der_sig_opt =
raw_to_der_ecdsa_signature(signature.data(), signature.size());
    if (!der_sig_opt) {
        return false;
    }
    const auto& der_signature = *der_sig_opt;

```

```

        std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
        if (!ctx) return false;

        if (EVP_DigestVerifyInit(ctx.get(), nullptr, md_type, nullptr,
pkey) <= 0) {
            return false;
        }

        if (EVP_DigestVerifyUpdate(ctx.get(), data_to_verify.data(),
data_to_verify.size()) <= 0) {
            return false;
        }

        int verify_result = EVP_DigestVerifyFinal(ctx.get(),
der_signature.data(), der_signature.size());

        return (verify_result == 1);
    }

std::optional<std::vector<unsigned char>> sign_eddsa(
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_sign)
{
    EVP_PKEY* pkey = key.get();
    int pkey_type = EVP_PKEY_base_id(pkey);
    if (!pkey || (pkey_type != EVP_PKEY_ED25519 && pkey_type !=
EVP_PKEY_ED448)) {
        return std::nullopt;
    }

    std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
    if (!ctx) return std::nullopt;

    if (!EVP_DigestSignInit(ctx.get(), nullptr, nullptr, nullptr,
pkey)) {
        return std::nullopt;
    }

    size_t signature_len = EVP_PKEY_size(pkey);
    std::vector<unsigned char> signature(signature_len);

    if (EVP_DigestSign(ctx.get(), signature.data(),
&signature_len, reinterpret_cast<const unsigned
char*>(data_to_sign.data()), data_to_sign.size()) != 1) {
        return std::nullopt;
    }

    signature.resize(signature_len);

    return signature;
}

```

```

}

bool verify_eddsa(
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_verify,
    const std::vector<unsigned char>& signature)
{
    EVP_PKEY* pkey = key.get();
    int pkey_type = EVP_PKEY_base_id(pkey);
    if (!pkey || (pkey_type != EVP_PKEY_ED25519 && pkey_type !=
EVP_PKEY_ED448)) {
        return false;
    }
    size_t expected_len = EVP_PKEY_size(pkey);
    if (signature.size() != expected_len) {
        return false;
    }

    std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>
ctx(EVP_MD_CTX_new(), EVP_MD_CTX_free);
    if (!ctx) return false;

    if (EVP_DigestVerifyInit(ctx.get(), nullptr, nullptr, nullptr,
pkey) <= 0) {
        return false;
    }

    int verify_result = EVP_DigestVerify(ctx.get(),
signature.data(), signature.size(), reinterpret_cast<const
unsigned char*>(data_to_verify.data()), data_to_verify.size());

    return (verify_result == 1);
}

std::optional<std::vector<unsigned char>> sign(
    const std::string& algorithm,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_sign)
{
    const EVP_MD* md_type = get_evp_md_from_jwa(algorithm);

    if (algorithm.rfind("HS", 0) == 0) {
        if (!md_type) return std::nullopt;
        return sign_hmac(md_type, key, data_to_sign);
    } else if (algorithm.rfind("RS", 0) == 0) {
        if (!md_type) return std::nullopt;
        return sign_rsa(md_type, RSA_PKCS1_PADDING, key,
data_to_sign);
    } else if (algorithm.rfind("ES", 0) == 0) {
        if (!md_type) return std::nullopt;
        return sign_ecdsa(algorithm, md_type, key, data_to_sign);
    } else if (algorithm.rfind("PS", 0) == 0) {

```

```

        if (!md_type) return std::nullopt;
        return sign_rsa(md_type, RSA_PKCS1_PSS_PADDING, key,
data_to_sign);
    } else if (algorithm == "EdDSA") {
        return sign_eddsa(key, data_to_sign);
    }

    return std::nullopt;
}

bool verify(
    const std::string& algorithm,
    const jwt_cpp::ManagedKey& key,
    const std::string& data_to_verify,
    const std::vector<unsigned char>& signature)
{
    const EVP_MD* md_type = get_evp_md_from_jwa(algorithm);

    if (algorithm.rfind("HS", 0) == 0) {
        if (!md_type) return false;
        return verify_hmac(md_type, key, data_to_verify,
signature);
    } else if (algorithm.rfind("RS", 0) == 0) {
        if (!md_type) return false;
        return verify_rsa(md_type, RSA_PKCS1_PADDING, key,
data_to_verify, signature);
    } else if (algorithm.rfind("ES", 0) == 0) {
        if (!md_type) return false;
        return verify_ecdsa(algorithm, md_type, key,
data_to_verify, signature);
    } else if (algorithm.rfind("PS", 0) == 0) {
        if (!md_type) return false;
        return verify_rsa(md_type, RSA_PKCS1_PSS_PADDING, key,
data_to_verify, signature);
    } else if (algorithm == "EdDSA") {
        return verify_eddsa(key, data_to_verify, signature);
    }

    return false;
}
}

```

Приложение Б
(рекомендуемое)
Спецификации разработанных функций

Название функции: <code>jwt_create</code>
Описание: создает контекст для формирования нового JWT
Параметры: – <code>handle_out: JwtHandle*</code> – указатель, куда будет записан созданный дескриптор.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.1 – Спецификация функции создания контекста токена

Название функции: <code>jwt_destroy</code>
Описание: уничтожает контекст JWT и освобождает связанные ресурсы
Параметры: – <code>handle: JwtHandle</code> – дескриптор для уничтожения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.2 – Спецификация функции уничтожения контекста токена

Название функции: <code>jwt_sign</code>
Описание: подписывает текущий JWT заданным ключом
Параметры: – <code>handle: JwtHandle</code> – контекст JWT с установленным алгоритмом и утверждениями; – <code>key_handle: JwtKeyHandle</code> – дескриптор ключа для подписи; – <code>token_out: char**</code> – указатель, куда будет записана строка с подписанным JWT.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.3 – Спецификация функции подписи токена

Название функции: <code>jwt_load_verify</code>
Описание: загружает JWT из строки, проверяет его подпись и создает из него объект токена
Параметры: – <code>token_str: const char*</code> – строка с JWT; – <code>key_handle: JwtKeyHandle</code> – дескриптор ключа для проверки подписи; – <code>handle_out: JwtHandle*</code> – указатель, куда будет записан дескриптор контекста JWT на основе переданного токена в случае успешной проверки токена.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: – не выполняет валидацию утверждений и заголовка.

Рисунок Б.4 – Спецификация функции проверки подписи токена с помощью загруженного ключа

Название функции: <code>jwt_load_verify_jwks</code>
Описание: загружает JWT, ищет ключ для проверки в JWK Set по «kid» из заголовка, проверяет подпись найденным ключом и создает объект токена
Параметры: <ul style="list-style-type: none"> – <code>token_str: const char*</code> – строка с JWT, содержащим «kid» в заголовке; – <code>keyset_handle: KeySetHandle</code> – дескриптор набора ключей JWKS для поиска ключа проверки; – <code>handle_out: JwtHandle</code> – указатель, куда будет записан дескриптор контекста JWT на основе переданного токена в случае успешной проверки токена.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: <ul style="list-style-type: none"> – не выполняет валидацию утверждений и заголовка.

Рисунок Б.5 – Спецификация функции проверки подписи с помощью JWKS

Название функции: <code>jwt_load_verify_x5c</code>
Описание: загружает JWT, извлекает открытый ключ из первого сертификата в заголовке «x5c», проверяет подпись этим ключом и создает объект токена
Параметры: <ul style="list-style-type: none"> – <code>token_str: const char*</code> – строка с JWT, содержащим «x5c» в заголовке; – <code>handle_out: JwtHandle*</code> – указатель, куда будет записан дескриптор контекста JWT на основе переданного токена в случае успешной проверки токена.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: <ul style="list-style-type: none"> – не выполняет валидацию утверждений и заголовка.

Рисунок Б.6 – Спецификация функции проверки подписи токена с помощью ключа из сертификата в заголовке

Название функции: <code>jwt_add_claim_string</code>
Описание: добавляет строковое утверждение в полезную нагрузку JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value: const char*</code> – значение утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.7 – Спецификация функции добавления строкового утверждения

Название функции: <code>jwt_add_claim_int</code>
Описание: добавляет целочисленное утверждение в полезную нагрузку JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value: long long</code> – значение утверждения (64 битное целое).
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.8 – Спецификация функции добавления целочисленного утверждения

Название функции: <code>jwt_add_claim_double</code>
Описание: добавляет утверждение с плавающей точкой в полезную нагрузку JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value: double</code> – значение утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.9 – Спецификация функции добавления дробного утверждения

Название функции: <code>jwt_add_claim_bool</code>
Описание: добавляет булево утверждение в полезную нагрузку JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value: int</code> – значение утверждения (0 для false, не 0 для true).
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.10 – Спецификация функции добавления булевого утверждения в токен

Название функции: <code>jwt_add_claim_json</code>
Описание: добавляет утверждение из строки, содержащей валидный JSON (для массивов и объектов)
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>key</code>: <code>const char*</code> – ключ (имя) утверждения; – <code>json_value_str</code>: <code>const char*</code> – строка с валидным JSON значением.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.11 – Спецификация функции добавления JSON утверждения в токен

Название функции: <code>jwt_set_alg</code>
Описание: устанавливает алгоритм подписи «alg» в заголовке JWT
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>alg</code>: <code>const char*</code> – строка с названием алгоритма (например «HS256»).
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.12 – Спецификация функции установки алгоритма подписи в заголовок токена

Название функции: <code>jwt_add_header_string</code>
Описание: добавляет строковый параметр в заголовок JWT
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>key</code>: <code>const char*</code> – ключ (имя) параметра; – <code>value</code>: <code>const char*</code> – значение параметра.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: <ul style="list-style-type: none"> – нельзя использовать для установки «alg».

Рисунок Б.13 – Спецификация функции добавления строкового параметра в заголовок токена

Название функции: <code>jwt_add_header_int</code>
Описание: добавляет целочисленный параметр в заголовок JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value: long long</code> – значение параметра (64 битное целое).
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.14 – Спецификация функции добавления целочисленного параметра
в заголовок токена

Название функции: <code>jwt_add_header_double</code>
Описание: добавляет параметр с плавающей точкой в заголовок JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value: double</code> – значение параметра.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.15 – Спецификация функции добавления параметра с плавающей
точкой в заголовок токена

Название функции: <code>jwt_add_header_bool</code>
Описание: добавляет булев параметр в заголовок JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value: int</code> – значение параметра (0 для false, не 0 для true).
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.16 – Спецификация функции добавления булевого параметра в
заголовок токена

Название функции: <code>jwt_add_header_json</code>
Описание: добавляет параметр из строки, содержащей валидный JSON (для массивов и объектов) в заголовок
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>key</code>: <code>const char*</code> – ключ (имя) параметра; – <code>json_value_str</code>: <code>const char*</code> – строка с валидным JSON значением.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.17 – Спецификация функции добавления JSON параметра в заголовок токена

Название функции: <code>jwt_get_payload_claim_string</code>
Описание: получает строковое утверждение из полезной нагрузки проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>key</code>: <code>const char*</code> – ключ (имя) утверждения; – <code>value_out</code>: <code>char**</code> – указатель, куда будет записана строка со значением утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.18 – Спецификация функции получения строкового утверждения

Название функции: <code>jwt_get_payload_claim_int</code>
Описание: получает целочисленное утверждение из полезной нагрузки проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>key</code>: <code>const char*</code> – ключ (имя) утверждения; – <code>value_out</code>: <code>long long*</code> – указатель, куда будет записано значение утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.19 – Спецификация функции получения целочисленного утверждения

Название функции: <code>jwt_get_payload_claim_double</code>
Описание: получает утверждение с плавающей точкой из полезной нагрузки проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value_out: double*</code> – указатель, куда будет записано значение утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.20 – Спецификация функции получения значения с плавающей точкой

Название функции: <code>jwt_get_payload_claim_bool</code>
Описание: получает булево утверждение из полезной нагрузки проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value_out: bool*</code> – указатель, куда будет записано значение утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.21 – Спецификация функции получения булевого утверждения

Название функции: <code>jwt_get_payload_claim_json</code>
Описание: получает утверждение как строку с JSON из полезной нагрузки проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) утверждения; – <code>value_out: char**</code> – указатель, куда будет записано значение утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.22 – Спецификация функции получения JSON утверждения

Название функции: <code>jwt_get_header_claim_string</code>
Описание: получает строковый параметр из заголовка проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value_out: char**</code> – указатель, куда будет записана строка со значением параметра.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.23 – Спецификация функции получения строкового параметра

Название функции: <code>jwt_get_header_claim_int</code>
Описание: получает целочисленный параметр из заголовка проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value_out: long long*</code> – указатель, куда будет записан параметр утверждения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.24 – Спецификация функции получения целочисленного параметра

Название функции: <code>jwt_get_header_claim_double</code>
Описание: получает параметр с плавающей точкой из полезной нагрузки проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value_out: double*</code> – указатель, куда будет записано значение параметра.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.25 – Спецификация функции получения дробного параметра

Название функции: <code>jwt_get_header_claim_bool</code>
Описание: получает булев параметр из заголовка проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value_out: bool*</code> – указатель, куда будет записано значение параметра.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.26 – Спецификация функции получения булевого параметра

Название функции: <code>jwt_get_header_claim_json</code>
Описание: получает параметр как строку с JSON из заголовка проверенного JWT
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>key: const char*</code> – ключ (имя) параметра; – <code>value_out: char**</code> – указатель, куда будет записано значение параметра.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.27 – Спецификация функции получения JSON параметра

Название функции: <code>jwt_get_header_x5t</code>
Описание: получает отпечаток сертификата SHA-1 из заголовка «x5c» в Base64URL
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>thumbprint_out: char**</code> – указатель, куда будет записана строка с отпечатком.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.28 – Спецификация функции получения отпечатка SHA-1

Название функции: <code>jwt_get_header_x5t_s256</code>
Описание: получает отпечаток сертификата SHA-256 из заголовка «x5t#S256» в Base64URL
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>thumbprint_out: char**</code> – указатель, куда будет записана строка с отпечатком.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.29 – Спецификация функции получения отпечатка SHA-256

Название функции: <code>jwt_add_header_x5c_certificate_data</code>
Описание: добавляет сертификат X.509 в заголовок «x5c» JWS
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>cert_data</code>: <code>const unsigned char*</code> – указатель на данные сертификата; – <code>cert_len</code>: <code>size_t</code> – длина данных сертификата в байтах.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.30 – Спецификация функции добавления сертификата в «x5c»

Название функции: <code>jwt_set_validation_leeway</code>
Описание: устанавливает допустимое отклонение времени в секундах для проверок «exp», «nbf», «iat» для компенсации рассинхронизации часов, если необходимо
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>leeway_seconds</code>: <code>unsigned int</code> – допустимое отклонение в секундах, например 60.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.31 – Спецификация функции установки отклонения времени

Название функции: <code>jwt_set_expected_issuer</code>
Описание: устанавливает ожидаемого издателя «iss» для валидации
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>issuer</code>: <code>const char*</code> – ожидаемая строка издателя.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: <ul style="list-style-type: none"> – параметр <code>issuer</code> может быть значением 0 для отмены проверки.

Рисунок Б.32 – Спецификация функции ожидаемого издателя

Название функции: <code>jwt_add_expected_issuer</code>
Описание: добавляет ожидаемую аудиторию «aud» в список валидации
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>audience: const char*</code> – ожидаемая строка аудитории.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.33 – Спецификация функции добавления ожидаемой аудитории

Название функции: <code>jwt_validate_claims</code>
Описание: выполняет валидацию стандартных утверждений «exp», «nbf», «iat», «iss», «aud» для загруженного и проверенного токена
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.34 – Спецификация функции валидации стандартных утверждений

Название функции: <code>jwt_add_understood_crit_header</code>
Описание: добавляет имя заголовка в список понятных приложению для валидации «crit»
Параметры: <ul style="list-style-type: none"> – <code>handle: JwtHandle</code> – контекст JWT; – <code>header_name: const char*</code> – имя заголовка, который приложение понимает и обрабатывает.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.35 – Спецификация функции добавления имени заголовка в список понятных приложению

Название функции: <code>jwt_validate_crit_headers</code>
Описание: выполняет валидацию заголовка «crit», предварительно заголовки должны быть добавлены в список понятных
Параметры: – <code>handle</code> : <code>JwtHandle</code> – контекст JWT.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.36 – Спецификация функции валидации заголовка «crit»

Название функции: <code>jwt_keyset_load_jwks</code>
Описание: загружает набор ключей из строки, представляющей JSON объект с массивом «keys», где каждый элемент – JWK с «kid»
Параметры: – <code>jwks_json_str</code> : <code>const char*</code> – строка с JWKS в формате JSON; – <code>ignore_load_headers</code> : <code>int</code> – игнорирование ошибок загрузки отдельных ключей, если не 0 – игнорировать; – <code>keyset_handle_out</code> : <code>KeySetHandle*</code> – указатель, куда будет записан дескриптор набора ключей.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.37 – Спецификация функции загрузки набора ключей из строки

Название функции: <code>jwt_keyset_export_jwks</code>
Описание: экспортирует весь набор ключей из <code>KeySetHandle</code> в строку формата JWKS JSON
Параметры: – <code>ks_handle</code> : <code>KeySetHandle</code> – дескриптор набора ключей для экспорта; – <code>include_private</code> : <code>int</code> – включение экспорта закрытых ключей, да – если не 0; – <code>jwks_json_out</code> : <code>char**</code> – указатель, куда будет записана строка с JWK Set JSON
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.38 – Спецификация функции экспорта набора ключей в строку

Название функции: jwt_keyset_find_key_by_kid
Описание: ищет ключ в наборе по его «kid»
Параметры: <ul style="list-style-type: none"> – keyset_handle: KeySetHandle – дескриптор набора ключей; – kid: const char* – идентификатор ключа для поиска; – key_handle_out: JwtKeyHandle – указатель, куда будет записан дескриптор найденного ключа.
Возвращаемое значение: JWT_SUCCESS (0) в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.39 – Спецификация функции поиска ключа в наборе по его «kid»

Название функции: jwt_keyset_set_enforce_use
Описание: включает или выключает принудительную проверку параметра «use» ключа JWK при поиске ключа для верификации
Параметры: <ul style="list-style-type: none"> – keyset_handle: KeySetHandle – дескриптор набора ключей; – enforce: int – значение проверки, 0 – выключить, иначе – включить.
Возвращаемое значение: JWT_SUCCESS (0) в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.40 – Спецификация функции управления проверкой «use»

Название функции: jwt_keyset_set_enforce_key_ops
Описание: включает или выключает принудительную проверку параметра «key_ops» ключа JWK при поиске ключа для верификации
Параметры: <ul style="list-style-type: none"> – keyset_handle: KeySetHandle – дескриптор набора ключей; – enforce: int – значение проверки, 0 – выключить, иначе – включить.
Возвращаемое значение: JWT_SUCCESS (0) в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.41 – Спецификация функции управления проверкой «key_ops»

Название функции: jwt_keyset_set_enforce_alg
Описание: включает или выключает принудительную проверку параметра «alg» ключа JWK при поиске ключа для верификации
Параметры: <ul style="list-style-type: none"> – keyset_handle: KeySetHandle – дескриптор набора ключей; – enforce: int – значение проверки, 0 – выключить, иначе – включить.
Возвращаемое значение: JWT_SUCCESS (0) в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.42 – Спецификация функции управления проверкой «alg»

Название функции: <code>jwt_keyset_destroy</code>
Описание: уничтожает набор ключей и освобождает все связанные с ним ресурсы, включая ключи
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей для уничтожения.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.43 – Спецификация функции уничтожения набора ключей

Название функции: <code>jwt_keyset_get_key_use</code>
Описание: получает параметр «use» ключа из набора JWKS
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа.
Возвращаемое значение: указатель на строку со значением, если присутствует, иначе 0
Примечания: отсутствуют

Рисунок Б.44 – Спецификация функции получения параметра «use» ключа

Название функции: <code>jwt_keyset_get_key_alg</code>
Описание: получает параметр «alg» ключа из набора JWKS
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа.
Возвращаемое значение: указатель на строку со значением, если присутствует, иначе 0
Примечания: отсутствуют

Рисунок Б.45 – Спецификация функции получения параметра «alg» ключа

Название функции: <code>jwt_keyset_get_key_ops_count</code>
Описание: получает количество операций в параметре «key_ops» ключа из набора JWKS
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа.
Возвращаемое значение: количество операций или -1 при отсутствии
Примечания: отсутствуют

Рисунок Б.46 – Спецификация функции получения количества операций в параметре «key_ops» ключа из набора

Название функции: <code>jwt_keyset_get_key_op_at_index</code>
Описание: получает строку операции из параметра «key_ops» по индексу ключа из набора
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа; – <code>index</code>: <code>int</code> – индекс операции.
Возвращаемое значение: указатель на строку с операцией или 0 при отсутствии
Примечания: отсутствуют

Рисунок Б.47 – Спецификация функции получения операции ключа по индексу

Название функции: <code>jwt_keyset_get_key_x5c_count</code>
Описание: получает количество сертификатов в параметре «x5c» ключа из набора JWKS
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа.
Возвращаемое значение: количество сертификатов или -1 при отсутствии
Примечания: отсутствуют

Рисунок Б.48 – Спецификация функции получения числа сертификатов в параметре «x5c» ключа из набора

Название функции: <code>jwt_keyset_get_key_x5c_entry_pem</code>
Описание: получает сертификат из параметра «x5c» ключа по индексу в формате PEM
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа; – <code>index</code>: <code>int</code> – индекс сертификата в массиве «x5c».
Возвращаемое значение: указатель на строку с сертификатом в формате PEM или 0 при ошибке
Примечания: отсутствуют

Рисунок Б.49 – Спецификация функции получения сертификата из параметра ключа из набора

Название функции: <code>jwt_keyset_get_key_x5t</code>
Описание: получает отпечаток сертификата SHA-1 из параметра «x5t» ключа
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа.
Возвращаемое значение: указатель на строку с отпечатком или 0 при отсутствии
Примечания: отсутствуют

Рисунок Б.50 – Спецификация функции получения отпечатка ключа из параметра «x5t»

Название функции: <code>jwt_keyset_get_key_x5t_s256</code>
Описание: получает отпечаток сертификата SHA-256 из параметра «x5t#S256» ключа
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа.
Возвращаемое значение: указатель на строку с отпечатком или 0 при отсутствии
Примечания: отсутствуют

Рисунок Б.51 – Спецификация функции получения отпечатка ключа из параметра «x5t#S256»

Название функции: <code>jwt_key_load_secret</code>
Описание: загружает секретный ключ
Параметры: <ul style="list-style-type: none"> – <code>secret</code>: <code>const unsigned char*</code> – указатель на байты ключа; – <code>secret_len</code>: <code>size_t</code> – длина секрета в байтах; – <code>key_handle_out</code>: <code>JwtKeyHandle</code> – указатель, куда будет записан дескриптор ключа.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.52 – Спецификация функции загрузки секретного ключа

Название функции: <code>jwt_key_load_private_pem</code>
Описание: загружает закрытый ключ из строки PEM
Параметры: <ul style="list-style-type: none"> – <code>pem_data: const char*</code> – строка с ключом в формате PEM; – <code>password: const char*</code> – пароль для расшифровки, если необходим; – <code>key_handle_out: JwtKeyHandle</code> – указатель, куда будет записан дескриптор ключа.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.53 – Спецификация функции загрузки закрытого ключа из строки

Название функции: <code>jwt_key_load_public_pem</code>
Описание: загружает открытый ключ из строки PEM
Параметры: <ul style="list-style-type: none"> – <code>pem_data: const char*</code> – строка с ключом в формате PEM; – <code>key_handle_out: JwtKeyHandle</code> – указатель, куда будет записан дескриптор ключа.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.54 – Спецификация функции загрузки открытого ключа из строки

Название функции: <code>jwt_key_load_jwk</code>
Описание: загружает ключ из строки, представляющей JWK
Параметры: <ul style="list-style-type: none"> – <code>jwk_json_str: const char*</code> – строка с JWK; – <code>key_handle_out: JwtKeyHandle</code> – указатель, куда будет записан дескриптор ключа.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.55 – Спецификация функции загрузки ключа из JWK

Название функции: <code>jwt_keyset_export_jwk</code>
Описание: экспортирует ключ из набора JWKS в формат JWK строки, включая метаданные
Параметры: <ul style="list-style-type: none"> – <code>keyset_handle</code>: <code>KeySetHandle</code> – дескриптор набора ключей; – <code>kid</code>: <code>const char*</code> – идентификатор ключа; – <code>include_private</code>: <code>int</code> – включение данных закрытого ключа, если 0 – нет, иначе – да; – <code>jwk_json_out</code>: <code>char**</code> – указатель, куда будет записана строка с полным JWK.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.56 – Спецификация функции экспорта ключа из набора в строку JWK

Название функции: <code>jwt_key_destroy</code>
Описание: уничтожает ключ и освобождает связанные с ним ресурсы
Параметры: <ul style="list-style-type: none"> – <code>key_handle</code>: <code>JwtKeyHandle</code> – дескриптор ключа для удаления.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.57 – Спецификация функции уничтожения криптографического ключа

Название функции: <code>jwt_get_header_x5c_count</code>
Описание: получает количество сертификатов в заголовке «x5c»
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.58 – Спецификация функции получения количества сертификатов из «x5c»

Название функции: <code>jwt_get_header_x5c_entry_pem</code>
Описание: получает сертификат из заголовка «x5c» по индексу в формате PEM
Параметры: <ul style="list-style-type: none"> – <code>handle</code>: <code>JwtHandle</code> – контекст JWT; – <code>index</code>: <code>int</code> – индекс сертификата в «x5c»; – <code>pem_cert_out</code>: <code>char**</code> – указатель, куда будет записана строка с сертификатом.
Возвращаемое значение: <code>JWT_SUCCESS (0)</code> в случае успеха, иначе код ошибки
Примечания: отсутствуют

Рисунок Б.59 – Спецификация функции получения сертификата из «x5c» по индексу

Приложение В
(рекомендуемое)
Листинг примеров использования

Листинг В.1 – Файл claims_headers_example.c

```
// В данном примере демонстрируется:
// 1) подпись/проверка jwt
// 2) установка разных типов claims
// 3) установка заголовков
// 4) установка критических заголовков
// 5) проверка стандартных claims
// 6) проверка критических заголовков
// 7) получение значений claims и заголовков

#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#include <abc_jwt.h>

// Проверка кода ошибки и выход при неудаче.
// Пренебрегаем освобождением ресурсов в случае ошибки для этого
// примера
void check_jwt_error(JwtError err) {
    if (err != JWT_SUCCESS) {
        fprintf(stderr, "ERROR: %s (%d)\n",
            jwt_error_str(err), (int)err);
        exit(EXIT_FAILURE);
    }
}

int main()
{
    JwtKeyHandle key = NULL;
    JwtHandle handle_sign = NULL;
    JwtHandle handle_verify = NULL;
    char* token = NULL;
    char* token_data_claim = NULL;
    char* token_header_value = NULL;

    const unsigned char SECRET[] = "top_secret";
    const size_t SECRET_LEN = 10;

    // Получаем хэндл ключа
    check_jwt_error(jwt_key_load_secret(SECRET, SECRET_LEN,
    &key));

    // Инициализируем хэндл токена для подписи
    check_jwt_error(jwt_create(&handle_sign));
    // Устанавливаем алгоритм
    check_jwt_error(jwt_set_alg(handle_sign, "HS256"));

    const char* critical_header_name = "my_critical_header";
    const char* critical_header_value = "important_value";
    const char* crit_value_json = "[\"my_critical_header\"]";
}
```

```

    // Добавление значения в заголовок
    check_jwt_error(json_add_header_string(handle_sign,
critical_header_name, critical_header_value));
    // Добавление критического заголовка
    check_jwt_error(json_add_header_json(handle_sign, "crit",
crit_value_json));

    // Подготавливаем данные для claims
    long long current_time = (long long)time(NULL);      // Берем
текущее время
    long long expiry_time = current_time + 3600;          // Валиден
1 час
    long long nbf_time = current_time - 60;              // Вступил
в силу 60 секунд назад

    // Добавляем стандартные claims
    check_jwt_error(json_add_claim_string(handle_sign, "iss",
"me_issuer"));
    check_jwt_error(json_add_claim_string(handle_sign, "sub",
"user12345"));
    check_jwt_error(json_add_claim_string(handle_sign, "aud",
"my_audience"));
    check_jwt_error(json_add_claim_int(handle_sign, "exp",
expiry_time));
    check_jwt_error(json_add_claim_int(handle_sign, "nbf",
nbf_time));
    check_jwt_error(json_add_claim_int(handle_sign, "iat",
current_time));

    // Добавляем другие claims разных типов
    check_jwt_error(json_add_claim_string(handle_sign, "data",
"very important"));
    check_jwt_error(json_add_claim_double(handle_sign, "balance",
423.23));
    check_jwt_error(json_add_claim_bool(handle_sign, "is_admin",
0));
    check_jwt_error(json_add_claim_json(handle_sign, "roles",
"[\"user\", \"reader\"]"));

    // Подписываем jwt
    check_jwt_error(json_sign(handle_sign, key, &token));
    printf("Created token: %s\n\n", token);

//=====

    // Проверяем подпись jwt
    check_jwt_error(json_load_verify(token, key, &handle_verify));
    printf("Successfully verified token: %s\n\n", token);

```

```

    // Добавляем имя заголовка для последующей валидации crit
    check_jwt_error.jwt_add_understood_crit_header(handle_verify,
"my_critical_header"));
    // Проверяем критические заголовки
    check_jwt_error.jwt_validate_crit_headers(handle_verify));
    // Получаем значение заголовка
    check_jwt_error.jwt_get_header_claim_string(handle_verify,
"my_critical_header", &token_header_value));
    printf("Header value: %s\n\n", token_header_value);

    // Настраиваем проверку claims
    check_jwt_error.jwt_set_validation_leeway(handle_verify, 60));
    check_jwt_error.jwt_set_expected_issuer(handle_verify,
"me_issuer"));
    check_jwt_error.jwt_add_expected_audience(handle_verify,
"my_audience"));

    // Валидируем claims. Сюда входят exp, nbf, iat по умолчанию,
а также добавленные iss и aud
    check_jwt_error.jwt_validate_claims(handle_verify));
    printf("Successfully verified claims\n\n");

    // Получаем значение claim из токена
    check_jwt_error.jwt_get_payload_claim_string(handle_verify,
"data", &token_data_claim));
    printf("Claim aquired: %s\n\n", token_data_claim);

    // Освобождаем структуры
    jwt_key_destroy(key);
    jwt_destroy(handle_sign);
    jwt_destroy(handle_verify);

    // Все строки возвращаемые библиотекой аллоцируются в куче,
освобождаем
    jwt_free_string(token);
    jwt_free_string(token_data_claim);
    jwt_free_string(token_header_value);

    return 0;
}

```

Листинг В.2 – Файл full_sign_verify_example.c

```

// В данном примере демонстрируется:
// 1) работа с полным набором ассиметричных алгоритмов JWS
// 2) Установка заголовков, необходимых для проверки подписи
// 3) Использование открытого ключа из разных источников для
проверки подписи, в частности обычный PEM ключ, JWKS, x5c
заголовок

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#include <abc_jwt.h>

// Функция чтения файла по полному пути.
// Возвращает содержимое-строку файла при успехе, NULL - при
неудаче
// full_path [in] - Полный путь до файла
// out_size [out] - Размер строки-содержимого
char* load_file_content(const char* full_path, long* out_size);

// Вспомогательная функция, проверяющая корректность указанного
алгоритма.
// Возвращает 1 при соответствии допустимым алгоритмам, 0 - при
несоответствии
// alg [in] - Название алгоритма
int check_alg(const char* alg);

// Функция подписи jwt закрытым ключом (предварительно
загруженного из PEM файла) и проверки подписи чистым открытым
ключом
// Возвращает 0 при успешном завершении, другое значение в случае
ошибки в одной из функций
// alg [in] - Название алгоритма подписи
// private_key [in] - Хэндл закрытого ключа
// pub_key_file_content [in] - Открытый ключ в виде PEM строки
int sign_verify_pem_pubkey(const char* alg, JwtKeyHandle
private_key, const char* pub_key_file_content)
{
    JwtError err;
    JwtKeyHandle public_key = NULL;
    JwtHandle sign_handle = NULL;
    JwtHandle verify_handle = NULL;
    char* token = NULL;

    // Загрузка открытого ключа из PEM строки в хэндл
    if ((err = jwt_key_load_public_pem(pub_key_file_content,
&public_key)) != JWT_SUCCESS)
    {
        goto end;
    }
    // Создание хэндла jwt
    if ((err = jwt_create(&sign_handle)) != JWT_SUCCESS)
    {
        goto end;
    }
    // Установка алгоритма подписи
    if ((err = jwt_set_alg(sign_handle, alg)) != JWT_SUCCESS)
    {
        goto end;
    }
    // Подпись

```

```

    if ((err = jwt_sign(sign_handle, private_key, &token)) !=
JWT_SUCCESS)
    {
        goto end;
    }
    printf("\nResult token: %s\n\n", token);
    // Проверка с использованием хэндла открытого ключа
    err = jwt_load_verify(token, public_key, &verify_handle);

end:

    // Освобождение ресурсов
    jwt_free_string(token);
    jwt_key_destroy(public_key);
    jwt_destroy(sign_handle);
    jwt_destroy(verify_handle);

    return err;
}

// Функция подписи jwt закрытым ключом (предварительно
загруженного из PEM файла) и проверки подписи открытым ключом из
сертификата
// Возвращает 0 при успешном завершении, другое значение в случае
ошибки в одной из функций
// alg [in] - Название алгоритма подписи
// private_key [in] - Хэндл закрытого ключа
// cert_file_content [in] - Сертификат в виде строки
// cert_size [in] - Размер строки сертификата
int sign_verify_cert_pubkey(const char* alg, JwtKeyHandle
private_key, const char* cert_file_content, size_t cert_size)
{
    JwtError err;
    JwtHandle sign_handle = NULL;
    JwtHandle verify_handle = NULL;
    char* token = NULL;

    // Создание хэндла jwt
    if ((err = jwt_create(&sign_handle)) != JWT_SUCCESS)
    {
        goto end;
    }
    // Установка алгоритма подписи
    if ((err = jwt_set_alg(sign_handle, alg)) != JWT_SUCCESS)
    {
        goto end;
    }
    // Установка заголовка x5c и добавление содержимого
сертификата как значение, чтобы функция проверки смогла вытащить
из него открытый ключ

```

```

    if ((err = jwt_add_header_x5c_certificate_data(sign_handle,
(const unsigned char*)cert_file_content, cert_size)) !=
JWT_SUCCESS)
    {
        goto end;
    }
    // Подпись
    if ((err = jwt_sign(sign_handle, private_key, &token)) !=
JWT_SUCCESS)
    {
        goto end;
    }
    printf("\nResult token: %s\n\n", token);
    // Проверка подписи с использованием открытого ключа из
сертификата, найденного в заголовке x5c
    err = jwt_load_verify_x5c(token, &verify_handle);

end:

    // Освобождение ресурсов
    jwt_free_string(token);
    jwt_destroy(sign_handle);
    jwt_destroy(verify_handle);

    return err;
}

// Функция подписи jwt закрытым ключом (предварительно
загруженного из PEM файла) и проверки подписи открытым ключом из
набора ключей JWKS
// Возвращает 0 при успешном завершении, другое значение в случае
ошибки в одной из функций
// alg [in] - Название алгоритма подписи
// private_key [in] - Хэндл закрытого ключа
// keyset_file_content [in] - Содержимое JWKS
// kid [in] - kid, по которому предполагается поиск открытого
ключа
int sign_verify_keyset_pubkey(const char* alg, JwtKeyHandle
private_key, const char* keyset_file_content, const char* kid)
{
    JwtError err;
    KeySetHandle keyset = NULL;
    JwtHandle sign_handle = NULL;
    JwtHandle verify_handle = NULL;
    char* token = NULL;

    // Загружаем JWKS в специальный хэндл набора ключей
    if ((err = jwt_keyset_load_jwks(keyset_file_content, 1,
&keyset)) != JWT_SUCCESS)
    {
        goto end;
    }

```

```

// Создание хэндла jwt
if ((err = jwt_create(&sign_handle)) != JWT_SUCCESS)
{
    goto end;
}
// Установка алгоритма подписи
if ((err = jwt_set_alg(sign_handle, alg)) != JWT_SUCCESS)
{
    goto end;
}
// Установка заголовка kid со значением kid jwk который будет
использоваться в качестве открытого ключа
if ((err = jwt_add_header_string(sign_handle, "kid", kid)) !=
JWT_SUCCESS)
{
    goto end;
}
// Подпись
if ((err = jwt_sign(sign_handle, private_key, &token)) !=
JWT_SUCCESS)
{
    goto end;
}
printf("\nResult token: %s\n\n", token);
// Проверка подписи с использованием открытого ключа,
найденного в keyset по kid в заголовке
err = jwt_load_verify_jwks(token, keyset, &verify_handle);

end:

// Освобождение ресурсов
jwt_free_string(token);
jwt_keyset_destroy(keyset);
jwt_destroy(sign_handle);
jwt_destroy(verify_handle);

return err;
}

// Функция вывода справки по использованию
void print_help()
{
    printf("Usage: --alg <algorithm> --priv_key_file <filepath> --
pub_key_type <type> [other options]\n");
    printf("All options:\n");
    printf("--alg                Chosen algorithm. Avaliable:
RS256/384/512, ES256/384/512/256K, PS256/384/512, EdDSA.
REQUIRED\n");
    printf("--pub_key_type        Form of public key used to
verify signature. Avaliable: pem_file, cert_file, keyset/
REQUIRED\n");
}

```

```

    printf("--priv_key_file      Full filepath to a file with PEM
encoded private key. REQUIRED\n");
    printf("--pub_key_file      Full filepath to a file with PEM
encoded public key\n");
    printf("--cert_file        Full filepath to a file with
certificate (DER/PEM encoded)\n");
    printf("--keyset_file      Full filepath to a file with
keyset, where needed public JWK is located\n");
    printf("--kid              Kid of a key in a keyset\n");
    printf("--help              Print this help list\n");
    printf("Example: --alg RS256 --priv_key_file
/home/user/private.pem --pub_key_type keyset --keyset_file
/home/user/keyset.jwks --kid my-rsa-key\n");
}

```

```

int main(int argc, char** argv) {

    JwtKeyHandle private_key = NULL;

    int ret = 1;

    const char* kid = NULL;
    const char* alg = NULL;
    const char* pub_key_type = NULL;

    char* priv_key_file_content = NULL;
    char* pub_key_file_content = NULL;
    char* cert_file_content = NULL;
    char* keyset_file_content = NULL;

    const char* private_key_filepath = NULL;
    const char* public_key_filepath = NULL;
    const char* cert_filepath = NULL;
    const char* keyset_filepath = NULL;

    for (int i = 1; i < argc; ++i)
    {
        if ((strcmp(argv[i], "--alg") == 0) && i + 1 < argc)
        {
            alg = argv[++i];
        }
        else if ((strcmp(argv[i], "--pub_key_type") == 0) && i + 1
< argc)
        {
            pub_key_type = argv[++i];
        }
        else if ((strcmp(argv[i], "--kid") == 0) && i + 1 < argc)
        {
            kid = argv[++i];
        }
        else if ((strcmp(argv[i], "--priv_key_file") == 0) && i +
1 < argc)

```

```

        {
            private_key_filepath = argv[++i];
        }
    else if ((strcmp(argv[i], "--pub_key_file") == 0) && i + 1
< argc)
    {
        public_key_filepath = argv[++i];
    }
    else if ((strcmp(argv[i], "--cert_file") == 0) && i + 1 <
argc)
    {
        cert_filepath = argv[++i];
    }
    else if ((strcmp(argv[i], "--keyset_file") == 0) && i + 1
< argc)
    {
        keyset_filepath = argv[i];
    }
    else if (strcmp(argv[i], "--help") == 0)
    {
        print_help();
        return 0;
    }
    else
    {
        printf("Unrecognized option\n");
        print_help();
        return 1;
    }
}

if (!pub_key_type || !private_key_filepath || !alg)
{
    printf("Error: Required options are not set\n");
    return 1;
}

if (!check_alg(alg))
{
    printf("Error: Incorrect algorithm. For this example
allowed these: RS256/384/512, ES256/384/512/256K, PS256/384/512,
EdDSA\n");
    return 1;
}

priv_key_file_content =
load_file_content(private_key_filepath, NULL);
if (!priv_key_file_content)
{
    return 1;
}

```

```

    if (jwt_key_load_private_pem(priv_key_file_content, NULL,
&private_key) != JWT_SUCCESS)
    {
        printf("Error: %s", jwt_error_str(ret));
        goto end;
    }

    if (strcmp(pub_key_type, "pem_file") == 0)
    {
        if (!public_key_filepath)
        {
            printf("Error: In \'pem_file\' scenario --pub_key_type
option is required\n");
            goto end;
        }
        pub_key_file_content =
load_file_content(public_key_filepath, NULL);
        if (!pub_key_file_content)
        {
            goto end;
        }
        ret = sign_verify_pem_pubkey(alg, private_key,
pub_key_file_content);
    }
    else if (strcmp(pub_key_type, "cert_file") == 0)
    {
        if (!cert_filepath)
        {
            printf("Error: In \'cert_file\' scenario --cert_file
option is required\n");
            goto end;
        }
        size_t cert_size = 0;
        cert_file_content = load_file_content(cert_filepath,
&cert_size);
        if (!cert_file_content)
        {
            goto end;
        }
        ret = sign_verify_cert_pubkey(alg, private_key,
cert_file_content, cert_size);
    }
    else if (strcmp(pub_key_type, "keyset") == 0)
    {
        if (!keyset_filepath || !kid)
        {
            printf("Error: In \'keyset\' scenario --keyset_file
and --kid options are required\n");
            goto end;
        }
        keyset_file_content = load_file_content(keyset_filepath,
NULL);
    }

```

```

        if (!keyset_file_content)
        {
            goto end;
        }
        ret = sign_verify_keyset_pubkey(alg, private_key,
keyset_file_content, kid);
    }
    else
    {
        printf("Error: Unknown --pub_key_type value\n");
        goto end;
    }

    if (ret)
    {
        printf("Error: %s", jwt_error_str(ret));
    }
    else
    {
        printf("Successfully created and verified JWT\n");
    }

end:

    if (priv_key_file_content)
    {
        free(priv_key_file_content);
    }
    if (pub_key_file_content)
    {
        free(pub_key_file_content);
    }
    if (cert_file_content)
    {
        free(cert_file_content);
    }
    jwt_key_destroy(private_key);

    return ret;
}

char* load_file_content(const char* full_path, long* out_size) {

    FILE* fp = fopen(full_path, "rb");
    if (!fp) {
        fprintf(stderr, "ERROR: Cannot open file %s\n",
full_path);
        return NULL;
    }

    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);

```

```

    fseek(fp, 0, SEEK_SET);

    if (size < 0) {
        fclose(fp);
        fprintf(stderr, "ERROR: Cannot get size of file %s\n",
full_path);
        return NULL;
    }

    char* buffer = (char*)malloc(size + 1);
    if (!buffer) {
        fclose(fp);
        fprintf(stderr, "ERROR: Cannot allocate memory for file
%s\n", full_path);
        return NULL;
    }

    if (fread(buffer, 1, size, fp) != (size_t)size) {
        free(buffer);
        fclose(fp);
        fprintf(stderr, "ERROR: Cannot read file %s\n",
full_path);
        return NULL;
    }

    buffer[size] = '\0';
    fclose(fp);

    if (out_size)
    {
        *out_size = size;
    }
    return buffer; // free(buffer)
}

int check_alg(const char* alg)
{
    if (strcmp(alg, "RS256") == 0 ||
        strcmp(alg, "RS384") == 0 ||
        strcmp(alg, "RS512") == 0)
    {
        return 1;
    }
    if (strcmp(alg, "ES256") == 0 ||
        strcmp(alg, "ES384") == 0 ||
        strcmp(alg, "ES512") == 0 ||
        strcmp(alg, "ES256K") == 0)
    {
        return 1;
    }
    if (strcmp(alg, "PS256") == 0 ||
        strcmp(alg, "PS384") == 0 ||

```

```
        strcmp(alg, "PS512") == 0)
    {
        return 1;
    }
    if (strcmp(alg, "EdDSA") == 0)
    {
        return 1;
    }
    return 0;
}
```