Акционерное общество «Национальная платежная корпорация Национального Банка Республики Казахстан»

Утвержден
решением Правления АО «НПК»
от «10» 10 2024 года
(протокол № 25)

Дата вступления в силу с
«10» 10 2024 г.

Межбанковская система обмена информацией по открытым программным интерфейсам (Open API). Стандарт информационной безопасности. Профили безопасности Open API

Рег. № 117

г. Алматы

Должность, подразделение разработчика	ФИО	Подпись	Дата
Начальник отдела развития управления ЦОИД	Сагадиев А.А.		

ЛИСТ СОГЛАСОВАНИЯ

№ п/п	Должность	ФИО	Подпись	Дата
1	И.о. Председателя Правления	Самаева Ж.Т.		
2	Директор департамента информационных технологий	Кандалов С.В.		
3	Начальник управления правового обеспечения	Шагалтаев Д.К.		
4	Начальник управления информационной безопасности	Муканов Т.Л.		
5	Начальник управления открытого банкинга	Измайлова А.Ж.		
6	Главный разработчик-программист управления разработки	Мусрепова Ж.О.		
7	Главный сотрудник отдела информационной безопасности	Курмалаев А.Т.		

Содержание

1. Область применения

1. Настоящий стандарт устанавливает требования и рекомендации для обеспечения безопасного доступа к финансовым данным и общедоступной информации в финансовых сервисах.

2. Нормативные ссылки

2. Для применения настоящего стандарта необходимы, следующие ссылочные документы. Для недатированных ссылок применяют последнее издание ссылочного документа (включая все его изменения):

- СТ РК 34.005-2002 «Информационная технология. Основные термины и определения»;
- СТ РК ГОСТ Р 34.11-2015 «Информационная технология. Криптографическая защита информации. Функция хэширования»;
- СТ РК ГОСТ Р 34.10-2015 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи»;
- СТ РК ИСО/МЭК 10181-1-2008 «Информационная технология Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 1. Обзор»;
- СТ РК ИСО/МЭК 10181-2-2008 «Информационная технология. Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 2. Основы аутентификации»
- СТ РК ИСО/МЭК 10181-4-2008 «Информационная технология. Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 4. Основы неотказуемости»;
- СТ РК ИСО/МЭК 10181-6-2008 «Информационная технология. Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 6. Основы целостности»;
- СТ РК ISO/IEC 9797-1-2017 «Информационные технологии. Методы и средства обеспечения безопасности. Коды аутентификации сообщений (MAC). Часть 1 Механизмы, использующие блочный шифр»;
- СТ РК ИСО/МЭК 15816-2009 «Информационная технология. Методы защиты. Объекты информационной защиты по управлению доступом»;
- ГОСТ Р ИСО/МЭК 8825-1-2003 «Информационная технология. Правила кодирования АСН.1. Часть 1. Спецификация базовых (BER), канонических (CER) и отличительных (DER) правил кодирования»;
- RFC 2617 «HTTP Authentication: Basic and Digest Access Authentication»;
- RFC 4122 «A UniveRSAlly Unique IDentifier (UUID) URN Namespace»;
- RFC 4648 «The Base16, Base32, and Base64 Data Encodings»;
- RFC 5280 «Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile»;
- RFC 6749 «The OAuth 2.0 Authorization Framework»;

- RFC 6750 «The OAuth 2.0 Authorization Framework: Bearer Token Usage»;
- RFC 7515 «JSON Web Signature (JWS)»;
- RFC 7516 «JSON Web Encryption (JWE)»;
- RFC 7517 «JSON Web Key (JWK)»;
- RFC 7519 «JSON Web Token (JWT)»;
- RFC 7521 «Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants»;
- RFC 7523 «JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants»;
- RFC 7591 «OAuth 2.0 Dynamic Client Registration Protocol»;
- RFC 7662 «OAuth 2.0 Token Introspection»;
- RFC 8414 «OAuth 2.0 Authorization Server Metadata»;
- RFC 8705 «OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens».

Примечание - При использовании настоящего стандарта целесообразно проверить действие ссылочных стандартов и классификаторов по ежегодно издаваемому информационному указателю «Нормативные документы по стандартизации» по состоянию на текущий год и соответствующим ежемесячно издаваемым информационным указателям, опубликованным в текущем году. Если ссылочный документ заменен (изменен), то при пользовании настоящего стандарта следует руководствоваться замененным (измененным) документом. Если ссылочный документ отменен без замены, то положение, в котором дана ссылка на него, применяется в части, не затрагивающей эту ссылку.

3. Термины и определения

3. В настоящем стандарте применяются следующие термины с соответствующими определениями:

1) Аутентификация - комплекс мер для подтверждения подлинности участников при обмене электронными сообщениями, а также подлинности электронных сообщений;

2) Односторонняя аутентификация - аутентификация, обеспечивающая только лишь для одного из участников процесса аутентификации (объекта доступа) уверенность в том, что другой участник процесса аутентификации (субъект доступа) является тем, за кого себя выдает, предъявленным идентификатором доступа;

3) Взаимная аутентификация - обоюдная аутентификация, обеспечивающая для каждого из участников процесса аутентификации – и субъекту доступа, и объекту доступа – уверенность в том, что другой участник процесса аутентификации является тем, за кого себя выдает;

4) Протокол аутентификации - протокол, позволяющий участникам процесса аутентификации осуществить аутентификацию и реализующий алгоритм (правила), в рамках которого субъект доступа и объект доступа последовательно выполняют определенные действия и обмениваются сообщениями;

5) Идентификация (identification) - действия по присвоению субъектам и объектам доступа идентификаторов и (или) по сравнению предъявляемого

идентификатора с перечнем присвоенных идентификаторов;

6) Авторизация (authorization) - проверка, подтверждение и предоставление прав логического доступа при осуществлении субъектами доступа логического доступа;

7) Доступ - получение одной стороной информационного взаимодействия возможности использования ресурсов другой стороны;

8) [Криптографический] ключ (key) - изменяемый параметр в виде последовательности символов, определяющий криптографическое преобразование;

9) Хэш-функция (collision-resistant hash-function) - функция, преобразующая строки бит в строки бит фиксированной длины и удовлетворяющая следующим свойствам:

– по данному значению функции сложно вычислить исходные данные, преобразованные в это значение;

– для заданных исходных данных сложно вычислить другие исходные данные, преобразуемые в то же значение функции;

– сложно вычислить какую-либо пару исходных данных, преобразуемых в одно и то же значение;

10) Хэш-код (hash-code) - строка бит, являющаяся выходным результатом хэш-функции;

11) Код аутентификации [сообщения] (message authentication code) - специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных;

12) Код аутентификации сообщения на основе хэш-функции (hash-based message authentication code) - механизм обеспечения аутентичности информации на основе симметричного ключа, построенный с использованием хэш-функции;

13) [Электронная цифровая] подпись (signature) - набор электронных цифровых символов, созданный средствами электронной цифровой подписи и подтверждающий достоверность электронного документа, его принадлежность и неизменность содержания;

14) Процесс проверки подписи (verification process) - процесс, в качестве исходных данных которого используются подписанное сообщение, ключ проверки подписи и параметры схемы электронной цифровой подписи, результатом которого является заключение о правильности или ошибочности цифровой подписи;

15) Процесс формирования подписи (signature process) - процесс, в качестве исходных данных которого используются сообщение, ключ подписи и параметры схемы электронной цифровой подписи, а в результате формируется цифровая подпись;

16) Прикладной программный интерфейс (application program interface, API) - интерфейс между прикладным программным средством и прикладной

платформой, через который обеспечивается доступ к необходимым службам (услугам);

17) Клиент (client) - приложение, выполняющее запросы защищенных ресурсов от имени владельца ресурса и с его авторизацией;

18) Конфиденциальный клиент (confidential client) - клиент, который может обеспечить конфиденциальность своих учетных данных или может выполнить безопасную аутентификацию клиента с использованием других средств;

19) Публичный клиент (public client) - клиент, который не может обеспечить конфиденциальность своих учетных данных и не может выполнить безопасную аутентификацию клиента с помощью других средств;

20) Агент пользователя (user agent) - клиентское приложение, использующее определенный сетевой протокол для доступа к серверу;

21) Сервер авторизации (authorization server) - сервер, выдающий клиенту токены доступа после успешной аутентификации владельца ресурса и получения авторизации;

22) Владелец ресурса (resource owner) - субъект, способный предоставить доступ к защищенному ресурсу;

23) Конечный пользователь (end user) - владелец ресурса в случае, если он является человеком;

24) Защищенный ресурс (protected resource) - ресурс с ограниченным доступом;

25) Сервер ресурсов (resource server) - сервер, на котором размещены защищенные ресурсы, способный принимать и отвечать на запросы защищенных ресурсов с использованием токенов доступа;

26) Разрешение на доступ (authorization grant) - свидетельство, представляющее авторизацию владельца ресурса (для доступа к его защищенным ресурсам), которое далее используется клиентом для получения токена доступа;

27) Конечная точка (endpoint) - адрес ресурса, точка входа интерфейса;

28) Конечная точка авторизации (authorization endpoint) - конечная точка, используемая клиентом для получения авторизации от владельца ресурса посредством перенаправления агента пользователя;

29) Конечная точка токена (token endpoint) - конечная точка, используемая клиентом для обмена разрешения на доступ на токен доступа;

30) Конечная точка клиента (client endpoint) - конечная точка, на которую сервер авторизации возвращает ответы клиенту посредством агента пользователя;

31) Конечная точка UserInfo (UserInfo endpoint) - защищенный ресурс, который при предоставлении клиентом токена доступа возвращает авторизованную информацию о конечном пользователе;

32) Журнал логирования - записи, содержащие информацию о работе системы, используемую для мониторинга ее работы и выявления причин, в случае возникновения сбоя;

33) Параметр (claim) - часть информации о субъекте. Параметр представлен в

виде пары имя/значение, состоящей из имени параметра и значения параметра;

34) JSON веб-токен (JWT) - строка, представляющая набор параметров в формате объекта JSON, который закодирован в виде структуры JWS или JWE, сопровождая параметры цифровой подписью, кодом аутентификации и/или шифрованием;

35) Токен доступа (access token) - свидетельство, представляющее авторизацию, выданную клиенту сервером авторизации с одобрения владельца ресурса. Токен доступа содержит указание на конкретные области действия, к которым разрешен доступ, длительность доступа и другие параметры;

36) Токен обновления (refresh token) - свидетельство, используемое для получения токенов доступа. Токен обновления выдается клиенту сервером авторизации и используется для получения нового токена доступа, когда текущий токен доступа становится недействительным или истекает его срок действия, или для получения дополнительных токенов доступа с идентичной или более узкой областью действия;

37) Идентификационный токен (ID token) - это токен безопасности, который содержит параметры аутентификации конечного пользователя сервером авторизации;

38) Объект запроса (request object) - токен JWT, который содержит набор параметров запроса в качестве своих параметров;

39) Base64 кодирование - стандарт кодирования двоичных данных при помощи только 64 символов ASCII. Алфавит кодирования содержит алфавитно-цифровые латинские символы A-Z, a-z и 0–9 (62 знака) и 2 дополнительных символа, зависящих от системы реализации;

40) Base64url кодирование - Base64 кодирование строки октетов с использованием набора символов, допускающих использование результата кодирования в качестве имени файла или URL;

41) Синхропосылка - значения исходных открытых параметров алгоритма криптографического преобразования;

42) Атака (attack) - попытка уничтожения, раскрытия, изменения, блокирования, кражи, получения несанкционированного доступа к информации или его несанкционированного использования.

4. Обозначения и сокращения

4. В настоящем стандарте применяются следующие обозначения и сокращения:

1) API (Application Program Interface) - прикладной программный интерфейс;[4]).

2) Обеспечение целостности информации, обмениваемой вовлеченными сторонами, должно обеспечиваться путем применения цифровой подписи с соблюдением положений пункта 8.1.2 СТ РК ИСО/МЭК 10181-6-2008 (см. раздел «Библиография», пункт [5]).

Технология авторизации OAuth 2.0

5. OAuth 2.0 – семейство протоколов авторизации, позволяющих одному приложению (клиенту) получить доступ к данным другого приложения (сервера ресурсов). При этом клиент получает разрешение на доступ от имени пользователя (владельца ресурса), который владеет необходимыми учетными данными, позволяющими его аутентифицировать. Непосредственно разрешение на доступ (grant) выдает клиенту сервер авторизации, на котором зарегистрирован владелец ресурса. Сервер ресурсов и сервер авторизации могут совпадать.

Примечание - Сведения о спецификации OAuth 2.0 приведены в документах RFC 6749 (см. раздел «Библиография», пункт [7]) и RFC 6750 (см. раздел «Библиография», пункт [19]).

6. Общая схема протокола OAuth 2.0 включает в себя следующую последовательность шагов (Рисунок 1):

1)Клиент запрашивает авторизацию у владельца ресурса. Запрос авторизации может быть направлен владельцу ресурса напрямую или косвенно через сервер авторизации. Предпочтительным является второй вариант;

2)Клиент получает разрешение на доступ (grant), структуру данных, представляющую авторизацию владельца ресурса, выраженную с использованием одного из четырех типов разрешений: код авторизации (authorization code), неявное разрешение (implicit), пароль владельца ресурса (resource owner password credentials) и учетные данные клиента (client credentials). Тип разрешения на доступ зависит от метода, используемого клиентом для запроса авторизации, и типов разрешений, поддерживаемых сервером авторизации. Типы разрешений, поддерживаемые сервером авторизации, определяются при его разработке, исходя из его прикладных целей и задач. Настоящий стандарт регламентирует использование в качестве типа разрешения код авторизации;

3)Клиент запрашивает токен доступа посредством аутентификации на сервере авторизации и предоставления разрешения на доступ;

4)Сервер авторизации аутентифицирует клиента, проверяет разрешение на доступ и, если оно действительно, выдает токен доступа;

5)Клиент запрашивает защищенный ресурс на сервере ресурсов и аутентифицируется, представляя токен доступа;

6)Сервер ресурсов проверяет токен доступа и, если он действителен, обслуживает запрос.

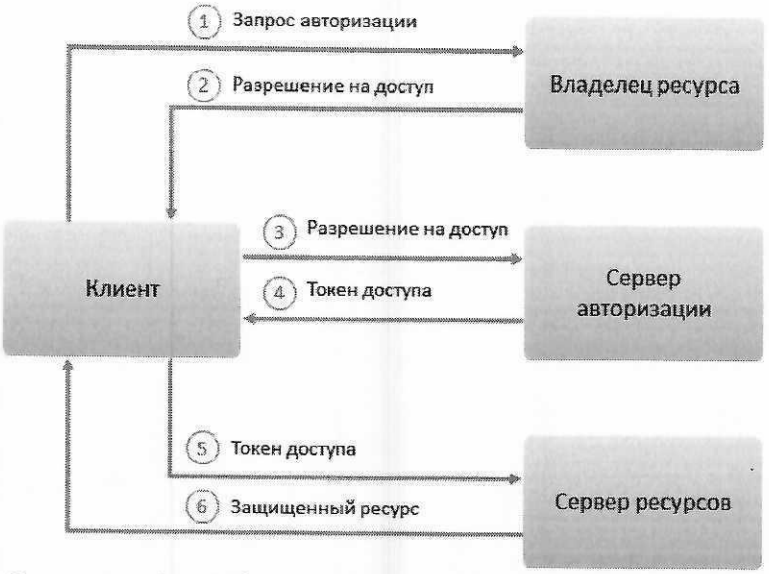

Рисунок 1 - Общая схема протокола OAuth 2.0

7. Для связи сервер авторизации и клиент используют конечные точки – адреса ресурсов или точки входа соответствующих сервисов. В процессе авторизации OAuth 2.0 используются две конечные точки сервера авторизации – конечная точка авторизации и конечная точка токена, а также одна конечная точка клиента.

8. Сервер авторизации должен поддерживать использование метода HTTP GET на конечной точке авторизации и может также поддерживать использование метода POST. При осуществлении запросов токена доступа клиент должен использовать метод HTTP POST.

9. Передача сообщений между клиентом и сервером авторизации должна производиться с использованием протокола TLS.

Протокол OpenID Connect

10. OpenID Connect – семейство протоколов, являющихся расширением протоколов OAuth 2.0, позволяющих расширить их функционал путем более точного описания процесса аутентификации владельца ресурса и возможности клиенту получить информацию о нем.

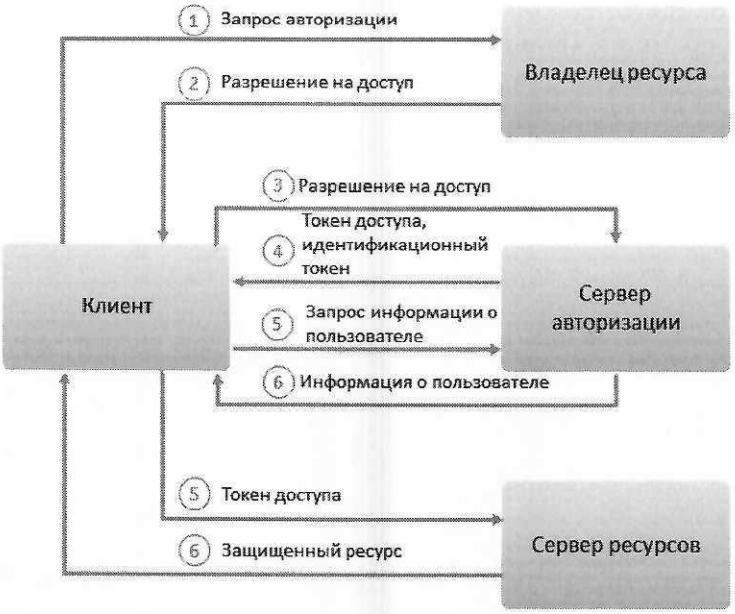

Рисунок 2 - Общая схема протокола OpenID Connect

11. Общая схема протокола OpenID Connect включает в себя следующую последовательность шагов (Рисунок 2):

1)Клиент отправляет серверу авторизации запрос аутентификации;

2)Сервер авторизации аутентифицирует конечного пользователя и получает согласие пользователя на доступ клиента к запрошенному ресурсу;

3)Сервер авторизации отвечает клиенту идентификационным токеном и (опционально) токеном доступа;

4)Клиент может отправить серверу авторизации запрос информации о пользователе по токену доступа;

5)Сервер авторизации возвращает клиенту информацию о конечном пользователе.

12. В протоколе OpenID Connect определены три сценария аутентификации:

1)с генерацией кода авторизации (Authorization Code Flow);

2)неявный сценарий (Implicit Flow);

3)гибридный сценарий (Hybrid Flow).

13. В настоящем стандарте не предполагается использование неявного сценария.

14. Описание сценария протокола аутентификации OpenID Connect с генерацией кода авторизации приведено в приложении 1.

15. Описание гибридного сценария протокола аутентификации OpenID Connect приведено в приложении 2.

16. Сервер авторизации публикует свои метаданные в ходе процедуры OpenID Connect Discovery. Описание процедуры регистрации сервера авторизации (OpenID Connect Discovery) и динамической регистрации клиента приведено в приложении 3.

Примечание - Дополнительные сведения о протоколе OpenID Connect приведены в документе «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3]).

Аутентификация клиента

17. При динамической регистрации (см. Приложение 3, пункт 5) клиент может зарегистрировать метод аутентификации клиента, указав его идентификатор в качестве значения параметра <token_endpoint_auth_method>. Если параметр <token_endpoint_auth_method> не зарегистрирован, то такой запрос не должен обрабатываться.

18. Допускается использование следующих вариантов механизмов аутентификации:

1) «client_secret_basic»: аутентификация с использованием базовой схемы аутентификации HTTP и <client_secret>; в настоящем стандарте допускается использование данного механизма аутентификации клиента только в тестовом режиме;

2) «client_secret_post»: аутентификация с включением учетных данных клиента в тело запроса и с использованием <client_secret>; в настоящем стандарте допускается использование данного механизма аутентификации клиента только в тестовом режиме;

Примечание - Спецификации механизмов «client_secret_basic» и «client_secret_post» представлены в «The OAuth 2.0 Authorization Framework» (см. раздел «Библиография», пункт [7]).

3) «client_secret_jwt»: аутентификация с использованием структуры JWT на основе кода аутентификации HMAC и <client_secret> (см. Приложение 4, пункт 1);

4) «private_key_jwt»: аутентификация с использованием структуры JWT на основе цифровой подписи (см. Приложение 4, пункт 2);

5) «tls_client_auth»: аутентификация с использованием протокола TLS с взаимной аутентификацией и PKI для связывания сертификата с клиентом (см. Приложение 4, пункт 3);

6) «self_signed_tls_client_auth»: аутентификация с использованием протокола TLS с взаимной аутентификацией и самоподписанного сертификата клиента (см. Приложение 4, пункт 4);

7) «none»: клиент не аутентифицирует себя в конечной точке токена; в настоящем стандарте допускается использование данного механизма аутентификации клиента только в тестовом режиме.

Доступ к защищенному ресурсу

19. Доступ клиента к защищенному ресурсу обеспечивает сервер ресурсов. Выполняя запрос, клиент передает ему полученный от сервера авторизации токен доступа. Сервер ресурса проверяет этот токен доступа: его срок действия, присутствие запрашиваемого ресурса в области действия токена.

20. При запросе защищенного ресурса токен доступа может быть передан клиентом серверу ресурса в заголовке авторизации HTTP (Authorization Request Header). В этом случае значение заголовка <Authorization> должно иметь формат: Authorization = «Bearer» || <token>, где <token> – значение токена доступа в кодировании Base64url.

Примечание - Спецификации структуры и алгоритмов работы с заголовком авторизации HTTP приведены в RFC 2617 (см. раздел «Библиография», пункт [18]).

21. Клиент и сервер ресурса должны поддерживать данный метод.

Примечание - Дополнительные сведения о протоколе доступа клиента к защищенному ресурсу с использованием токена доступа приведены в RFC 6750 (см. раздел «Библиография», пункт [13]).

22. Канал передачи информации между клиентом и сервером ресурса должен быть защищен с помощью протокола TLS с обеспечением конфиденциальности и целостности.

Требования к протоколу TLS

23. Все взаимодействия между клиентом и сервером авторизации, между клиентом и сервером ресурсов, между сервером авторизации и сервером ресурсов должны быть защищены с использованием протокола TLS (HTTPS).

24. Для всех взаимодействий должен использоваться протокол TLS версии 1.2 или более поздней.

25. Должна осуществляться проверка сертификата, используемого сервером для организации TLS взаимодействия на предмет срока действия и действительности (не отозван) по CRL (Certificate Revocation List) листу или с помощью OCSP (Online Certificate Status Protocol), при этом сертификат должен быть выпущен удостоверяющим центром, аккредитованным в соответствии с законодательством Республики Казахстан.

26. Криптографические ключи, используемые в протоколе TLS, и ключи протокола OIDC должны быть различными.

27. Сервер авторизации, сервер ресурсов не должны быть доступны без использования протокола TLS.

28. В случае обращения клиента без использования протокола TLS сервер авторизации, сервер ресурсов должны отказать в соединении или возвратить сообщение со статусом HTTP 301 и кодом ошибки «Moved Permanently».

Цифровая подпись

29. В зависимости от транспорта, с помощью которого отправляются сообщения, целостность сообщения может не гарантироваться, а отправитель сообщения может не проходить проверку подлинности. Для снижения данных рисков значения идентификационного токена, ответа UserInfo, объекта запроса и JWT аутентификации клиента может использоваться JWS для цифровой подписи содержимого этих структур данных. Для обеспечения конфиденциальности

сообщений также может использоваться JWE для шифрования содержимого этих структур данных.

30. Сервер авторизации может объявлять о поддерживаемых им алгоритмах подписи и шифрования в своем документе OpenID Connect Discovery (см. Приложение 3, пункт 1) или предоставлять эту информацию другими способами. Клиент может декларировать поддерживаемые им алгоритмы цифровой подписи и шифрования в своем запросе динамической регистрации (см. Приложение 3, пункт 5) или передавать эту информацию другими способами.

31. Сервер авторизации может объявлять свои открытые ключи цифровой подписи и шифрования через документ OpenID Connect Discovery или предоставлять эту информацию другими способами. Клиент может объявлять свои открытые ключи через запрос динамической регистрации или передавать эту информацию другими способами.

32. Подписывающая сторона должна использовать алгоритм цифровой подписи, поддерживаемый получателем.

33. Ключ асимметричной цифровой подписи, который используется для формирования подписи контента, должен быть связан с открытым ключом, используемым для проверки цифровой подписи и опубликованным отправителем в его документе JWK Set. Если в указанном документе JWK Set присутствует несколько ключей, в JOSE заголовке должно присутствовать значение параметра <kid>. Параметр <use> соответствующего ключа должен принимать значение “sig”, указывая на то, что этот ключ поддерживает алгоритм цифровой подписи.

34. При использовании подписи JWS на основе кода аутентификации сообщения (MAC) значение параметра <alg> JOSE заголовка должно быть равно идентификатору алгоритма MAC. Используемый ключ MAC – это октеты UTF-8 представления значения <client_secret>. Симметричная подпись не должна использоваться публичными клиентами из-за их неспособности сохранять ключ клиента.

35. Смена асимметричных ключей цифровой подписи должна быть выполнена с помощью следующего подхода:

1) Подписывающий публикует свои ключи в документе JWK Set, местоположение которого указывает в качестве значения параметра <jwks_uri> документа OpenID Connect Discovery (см. Приложение 3, пункт 1), и включает идентификатор ключа цифровой подписи в качестве значения параметра <kid> JOSE-заголовка каждого JWS сообщения, чтобы указать проверяющему, какой ключ должен использоваться для проверки цифровой подписи;

2) Ключи можно обновлять, периодически добавляя новые ключи проверки цифровой подписи в JWK Set по адресу <jwks_uri>;

3) Подписывающий может начать использовать новый ключ цифровой подписи по своему усмотрению, сигнализируя об этом проверяющему, использовав новое значение параметра <kid>;

4) Проверяющий, обнаружив неизвестное значение параметра <kid>, должен

обратиться по адресу <jwks_uri> для повторного получения ключей отправителя. Документ JWK Set по адресу <jwks_uri> должен сохранять недавно выведенные из действия ключи проверки цифровой подписи в течение периода времени, определенного при разработке сервера авторизации с целью плавного перехода на новые ключи.

Шифрование

36. Отправитель, выполняющий шифрование передаваемой информации, должен выбрать алгоритм шифрования из числа тех алгоритмов, которые поддерживает получатель. Перечень этих алгоритмов указан в конфигурации сервера авторизации (см. Приложение 3, пункт 1) и клиента (см. Приложение 3, пункт 5).

37. При использовании асимметричного шифрования на основе арифметики эллиптической кривой должен генерироваться эфемерный открытый ключ с использованием арифметики эллиптической кривой в качестве элемента <epk> JOSE-заголовка JWE. Второй открытый ключ, используемый для вычисления ключевого соглашения (key agreement), должен быть открытым ключом, опубликованным получателем в его документе JWK Set. Если в указанном документе JWK Set присутствует несколько ключей, в JOSE-заголовке JWE должно быть указано значение параметра <kid>. Для согласования ключа шифрования контента, который будет использован для шифрования подписанного JWT, необходимо использовать алгоритм элиптической кривой Диффи-Хэлмана с эфемерно-статическими ключами (ECDH-ES). Параметр <use> соответствующего ключа должен включать шифрование («enc»).

38. При использовании симметричного шифрования симметричный ключ шифрования должен вычисляться из значения <client_secret> с использованием усеченного слева хэш-кода октетов UTF-8 представления параметра <client_secret>. Симметричное шифрование не должно использоваться публичными клиентами из-за их неспособности сохранять ключ клиента.

39. При смене асимметричных ключей шифрования следует использовать процесс, отличный от процесса смены ключей цифровой подписи, поскольку процесс смены открытого ключа получателя запускает получатель, не являющийся отправителем (шифрующим), и, таким образом, отправитель не может полагаться на изменение параметра <kid> в качестве сигнала о необходимости смены ключей. Выполняющий шифрование продолжает использовать прежний параметр <kid> в заголовке JWE. При этом он должен выбрать наиболее подходящий ключ из представленных в JWK Set, расположенном по адресу <jwks_uri> получателя. При отсутствии в JWK Set открытого ключа получателя с разрешенным сроком использования шифрование не допускается. Об этом следует сообщить получателю, пользуясь другим надежным каналом связи.

40. Для смены ключей расшифровывающая сторона своевременно должна опубликовать новые ключи по своему адресу <jwks_uri> и удалить из JWK Set те

ключи, которые выводятся из эксплуатации. Должны быть приняты меры по корректному кэшированию <jwks_uri> в соответствии с рекомендациями «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3]). Получатель должен удалить отозванные ключи из JWK Set, но сохранять их у себя в течение некоторого периода времени, согласованного с продолжительностью кэша, чтобы обеспечить плавный переход между ключами, предоставляя отправителю некоторое время на загрузку новых ключей.

41. Продолжительность кэша следует также координировать с выдачей новых ключей подписи.

Требования к энтропии симметричных ключей

42. Ключи формирования MAC JWS и симметричного шифрования JWE вычисляются на основе значения ключа клиента <client_secret> (см. Приложение 3, пункт 5). Таким образом, при использовании с симметричными операциями подписи (MAC) или шифрования значения <client_secret> должны содержать достаточную энтропию для генерации криптографически стойких ключей.

43. Значения ключа клиента <client_secret> и другая ключевая информация должны генерироваться с помощью СКЗИ, используемого при реализации криптографической защиты информации.

44. Длина значения <client_secret> должна быть не менее той, которая требуется для ключей MAC для конкретного используемого алгоритма. Например, при использовании кода аутентификации HMAC с длиной значения 256 битов значение <client_secret> должно быть не менее 256 битов. При использовании кода аутентификации HMAC с длиной значения 512 битов значение <client_secret> должно быть не менее 512 битов.

Токен доступа, связанный с MTLS-сертификатом клиента

45. Если клиент использует взаимную аутентификацию по протоколу TLS в соответствии с RFC8705 (см. раздел «Библиография», пункт [25]) при подключении к конечной точке токена, сервер авторизации может связать выданный токен доступа с предъявленным сертификатом клиента. Такое связывание достигается путем встраивания хэш-кода сертификата в выданный токен доступа с использованием JWT-синтаксиса или посредством интроспекции токена (запроса информации о токене) у сервера авторизации. Эта привязка может выполняться как совместно с аутентификацией клиента по сертификату MTLS (см. Приложение 4, пункты 3 и 4), так и отдельно от аутентификации клиента сервером авторизации, что позволяет MTLS во время защищенного доступа к ресурсам служить исключительно механизмом подтверждения владения закрытым ключом.

46. Чтобы сервер ресурсов мог использовать токены доступа с привязкой к сертификату, он должен заранее знать, что для обращения к защищенным ресурсам должен использоваться MTLS. В частности, сам токен доступа не может

использоваться в качестве входных данных для принятия решения о том, запрашивать или нет установление MTLS-соединения.

47. В процессе доступа к ресурсам, защищенным протоколом TLS с взаимной аутентификацией сторон, клиент может выполнять запросы к защищенным ресурсам, как описано в подразделе 0, однако эти запросы должны быть выполнены по аутентифицированному MTLS-соединению, используя тот же сертификат, который использовался для MTLS-соединения в конечной точке токена при запросе токена доступа.

48. Сервер ресурсов должен получить TLS-сертификат клиента, используемый для установления взаимно аутентифицированного TLS-соединения, должен проверить, что сертификат соответствует сертификату, связанному с предъявленным токеном доступа. Если они не совпадают, попытка доступа к ресурсу должна быть отклонена со статусом HTTP 401 и кодом ошибки «invalid_token».

49. Метаданные, необходимые для того, чтобы сервер и клиент сообщали предложение об использовании токенов доступа с привязкой к MTLS-сертификату клиента, определены в подпунктах 56 и 57.

50. При использовании метода подтверждения отпечатка сертификата с использованием JWT, чтобы представить хэш-код X.509 сертификата в формате JWT (Приложение 5), в качестве значения параметра токена <cnf> должна использоваться строка, идентифицирующая метод подтверждения на основе хэш-функции с длиной хэш-кода не менее 256 битов, и значением хэш-кода, которое формируется как Base64url кодирование значения хэш-функции, вычисленное от DER-представления (ГОСТ Р ИСО/МЭК 8825-1-2003 (см. раздел «Библиография», пункт [8])) сертификата формата X.509.

Примечание - Дополнительные сведения о методе подтверждения отпечатка сертификата с использованием JWT приведены в RFC 8705 (см. раздел «Библиография», пункт [25]).

51. Пример функционального содержимого JWT, содержащего подтверждение отпечатка сертификата, отображает Рисунок 3.


```
{
 "iss": "https://server.example.com",
 "sub": "ty.webb@example.com",
 "exp": 1493726400,
 "nbf": 1493722800,
 "cnf":{
 "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
 }
}
```


Рисунок 3 - Пример функционального содержимого JWT, содержащего подтверждение отпечатка сертификата

52. Метод подтверждения отпечатка сертификата с использованием анализа токена.

53. Анализ токена OAuth 2.0 определяет способ, с помощью которого сервер ресурсов запрашивает у сервера авторизации информацию о состоянии активности токена доступа, а также другую информацию о токене доступа.

54. Для токена доступа, связанного с сертификатом MTLS, хэш-код сертификата, с которым связан токен, должен передаваться серверу защищенного ресурса в виде информации в составе ответа об результатах анализа токена. Хэш-код должен передаваться с использованием того же параметра <cnf> с элементом, идентифицирующим метод подтверждения на основе хэш-функции, что и при использовании метода подтверждения отпечатка сертификата, описанного в подпункте 54, в качестве параметра верхнего уровня JSON-ответа анализа. Сервер ресурсов сравнивает полученный таким образом хэш-код сертификата со значением хэш-кода, вычисленного на основе сертификата клиента, использованного для взаимной аутентификации сеанса TLS, и отклоняет запрос, если они не совпадают.

Примечания

1 Дополнительные сведения о протоколе анализа токена доступа представлены в RFC 7662 (см. раздел «Библиография», пункт [26]).

2 Дополнительные сведения о методе подтверждения отпечатка сертификата с использованием анализа токена приведены в RFC 8705 (см. раздел «Библиография», пункт [25]).

55. Пример ответа по результатам анализа для активного токена с подтверждением отпечатка сертификата отображает Рисунок 4.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
 "active": true,
 "iss": "https://server.example.com",
 "sub": "ty.webb@example.com",
 "exp": 1493726400,
 "nbf": 1493722800,
 "cnf":{
 "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
 }
}
```

Рисунок 4 - Пример ответа по результатам анализа для активного токена с подтверждением отпечатка сертификата

56. Сервер авторизации должен публиковать в соответствии с требованиями 1 следующий параметр метаданных, если он поддерживает выдачу токена доступа с привязкой к сертификату:

57. <tls_client_certificate_bound_access_tokens>: (опциональный) логическое значение, указывающее на поддержку сервером токенов доступа с привязкой к MTLS-сертификату клиента. Значением по умолчанию является «false».

58. Клиент должен публиковать в соответствии с требованиями подпункта 5 следующий параметр метаданных, если он намерен использовать токены доступа с привязкой к сертификату:

<tls_client_certificate_bound_access_tokens>: (опциональный) логическое значение, используемое для указания намерения клиента использовать токены доступа, привязанные к MTLS-сертификату клиента. Значением по умолчанию является «false».

59. Если клиент, который указал намерение использовать токены, привязанные к MTLS-сертификату клиента, отправляет запрос на конечную точку токена по соединению, не являющемуся MTLS, то сервер авторизации должен отклонять запрос с сообщением об ошибке.

Требования к журналированию событий

60. Следующие виды событий должны фиксироваться в журналах элементов, реализующих финансовые сервисы:

1) все действия привилегированных пользователей;
2) успешные и не успешные события установления соединений, идентификации, аутентификации и авторизации;
3) факты выпуска и отзыва токенов доступа;
4) ошибки обработки и проверки полученных данных (например, нарушения протокола, недопустимые кодировки, недопустимые имена и значения параметров);
5) сбои авторизации и управления сеансами;
6) запуск и завершение работы API, ошибки в работе API;
7) обнаружение вредоносных объектов;
8) изменения конфигурации API; события модификации настроек информационной безопасности;
9) события модификации учетных записей, групп, пользователей и их полномочий;
10) события, отражающие установку обновлений и(или) изменений в информационной системе;
11) другие события информационной безопасности, классифицируемые участниками как критичные.

61. Следующая информация не должна сохраняться в журналах событий в явном виде (допускается хранение только хэш-значений от данных объектов):

1) значения токенов доступа и токенов продления;
2) пароли;
3) ключи шифрования;
4) данные, составляющие персональные данные, банковскую или иную охраняемую законом тайну.

62. Журналы должны формироваться с указанием точного времени и даты события, с обязательной синхронизацией с эталонным источником времени в формате UTC.

63. Должны фиксироваться IP адреса сетевых соединений, имена или идентификаторы учетных записей, от лица которых совершаются взаимодействия, идентификаторы сообщений, настройки служб API при их запуске.

64. Должны фиксироваться объекты, в отношении которых производилось действие и результат действия (успешно или не успешно)

65. Должна обеспечиваться целостность и конфиденциальность журналов событий (доступ к журналам событий должен быть ограничен только необходимому кругу лиц).

66. При возникновении ошибок в работе API сторонам взаимодействия должно возвращаться диагностическое сообщение только с номером ошибки, не содержащее какой-либо конфиденциальной информации.

67. Должен быть обеспечен постоянный мониторинг журналов событий API, а также поддерживающей инфраструктуры для выявления нарушений конфиденциальности, целостности, доступности совершаемых посредством API взаимодействий.

68. Факты получения и отправки GET/POST запросов и ответов на них, осуществляемые в рамках настоящего стандарта, должны фиксироваться в журналах событий принимающей и передающей сторон и хранится в течение одного года.

Требование к формату идентификаторов

69. При назначении идентификаторов объектам и субъектам информационных систем рекомендуется использовать 4 версию метода universally unique identifier (UUID).

Примечание - Форматы и алгоритмы формирования UUID определены в RFC 4122 (см. раздел «Библиография», пункт [10]).

Требования к отказоустойчивости API интерфейсов (защите от атак типа отказ обслуживания)

70. Для API взаимодействий должно быть установлено максимально допустимое количество запросов в единицу времени для исключения атак типа отказ в обслуживании, а также для минимизации рисков несанкционированного доступа к конфиденциальной информации при множественных запросах с инъекциями.

71. Должна быть обеспечена защита API путем фильтрации, мониторинга и блокировки вредоносного трафика. Должна обеспечиваться защита API от

вредоносных атак и нежелательного трафика, в том числе инъекционных атак, межсайтового скриптинга (XSS), инъекции SQL и других уязвимостей.

Требования по проверке данных, передаваемых через API

72. Передаваемые данные (как входящие, так и исходящие) должны быть структурированы и приведены к необходимому типу в соответствии с утвержденным форматом API взаимодействий.

73. Передаваемые данные должны проходить проверку на соответствие принятому стандарту формата данных, при этом для каждого поля сообщения должен быть определен допустимый тип и длинна данных.

74. Передача и дальнейшая обработка данных, не прошедших проверку корректности на каком-либо этапе, должна быть заблокирована.

75. Должна быть запрещена обработка сообщений или вложений, содержащих вредоносный код, зашифрованные объекты, которые невозможно проверить, а также исполняемые файлы.

76. Для предотвращения инъекций вредоносного кода в рамках обмена через API взаимодействия необходимо проводить проверку всех поступающих сообщений на наличие специальных символов и управляющих команд. Сообщения, которые содержат управляющие команды и специальные символы, должны безопасным образом обрабатываться (например, с помощью экранирования специальных символов).

Требование к минимизации передаваемых через API данных

77. При информационном обмене через API взаимодействия на клиентскую сторону всегда должен отправляться только минимально необходимый объем информации для снижения риска раскрытия передаваемых данных.

5. Обеспечение безопасности авторизации при доступе к защищенным ресурсам

Спецификация сервера авторизации при доступе к защищенным ресурсам

78. Сервер авторизации должен обеспечивать выполнение следующих требований:

1) должен поддерживать конфиденциальных клиентов;

2) в случае использования симметричного ключа должен предоставлять ключ клиента <client_secret>;

3) должен аутентифицировать конфиденциального клиента в конечной точке токена, используя один из следующих методов:

- протокола TLS с взаимной аутентификацией сторон взаимодействия протокола OAuth 2.0 и его профилей, включая OpenID Connect, как определено в 3 или 4 (Приложение 4);
- <client_secret_jwt> или <private_key_jwt> как определено в пунктах 1 и 2

(Приложение 4);

4) должен требовать ключ, размер которого составляет 256 бит или больше, если для аутентификации клиента используются алгоритмы, основанные на использовании эллиптической кривой;

5) должен требовать предварительной регистрации URI переадресации клиента;

6) должен требовать наличия параметра <redirect_uri> в запросе аутентификации (Приложение 1);

7) должен требовать, чтобы значение параметра <redirect_uri> точно соответствовало одному из предварительно зарегистрированных URI переадресации клиента (см. Приложение 3, пункт 5);

8) должен требовать явного согласия конечного пользователя на авторизацию доступа к запрашиваемой области действия (параметр <scope>), если она не была ранее авторизована;

9) должен исключать повторное использование кодов авторизации в соответствии с рекомендациями пункта 12 (Приложение 1);

10) должен возвращать ответ с токеном доступа и, если это установлено при разработке сервера авторизации, с токеном обновления в соответствии с пунктом 9 (Приложение 1);

11) должен возвращать список предоставленных областей действия (параметр <scope>) по выданному токену доступа (Приложение 1);

12) должен предоставлять непредсказуемые значения токенов доступа как минимум с 128-битной энтропией; при этом рекомендуется генерировать токены с энтропией не ниже 160 бит;

13) необходимо уведомлять владельца ресурса о том, что клиент запрашивает долгосрочное разрешение на доступ, см. пункт 18 (Приложение 1);

14) при обращении к методам аутентификации клиента, которые могут передавать идентификатор клиента более чем одним разрешенным способом, в случае предоставления несовпадающих идентификаторов клиента должен возвращать код ошибки «invalid_client»;

15) должен требовать, чтобы сервисы по адресу URI переадресации использовали протокол HTTPS;

16) должен выдавать токены доступа с ограниченным сроком действия.

79. При предоставлении идентификатора аутентифицированного пользователя клиенту в ответе на запрос токена доступа сервер авторизации:

1) должен поддерживать запрос аутентификации (Приложение 1);

2) должен осуществлять проверку запроса аутентификации в соответствии пунктом 4 (Приложение 1);

3) должен предоставлять ответ на запрос аутентификации в соответствии с пунктами 6-8 (Приложение 1) в зависимости от результата аутентификации;

4) должен осуществлять проверку запроса токена в соответствии с пунктом 11 (Приложение 1);

5) если параметр <scope> запроса аутентификации включает «openid», должен генерировать идентификационный токен в составе ответа на запрос токена в соответствии с пунктом 13 (Приложение 1) при этом значение параметра <sub> должно соответствовать аутентифицированному пользователю.

80. Сервер авторизации для аутентификации конечного пользователя должен применять механизмы аутентификации, соответствующие требованиям законодательства РК (в частности, 4 классу СТ РК ИСО/МЭК 10181-2-2008 (раздел 7.1 Классификация по уязвимостям) (см. раздел «Библиография», пункт [6])).

Спецификация клиента для обеспечения безопасности авторизации при доступе к защищенным ресурсам

81. Клиент должен обеспечивать выполнение следующих требований:

1) должен использовать разные URI переадресации для каждого сервера авторизации, на котором зарегистрирован клиент;

2) должен поддерживать следующие методы аутентификации в конечной точке токена:

- протокола TLS с взаимной аутентификацией клиента OAuth 2.0, как определено в пунктах 3 или 4 (Приложение 4);
- <client_secret_jwt> или <private_key_jwt> как определено в пунктах 1 и 2 (Приложение 4);

3) должен использовать ключи с минимальным размером 256 бит, если используются криптографические алгоритмы на основе арифметики эллиптической кривой;

4) должен проверять, что значение ключа клиента <client_secret> имеет длину не менее 256 бит, если используется криптография с симметричным ключом;

Если по результатам аутентификации клиент запрашивает получение постоянного идентификатора аутентифицированного клиента, клиент:

5) среди значений, передаваемых в параметре <scope>, должен указывать значение «openid»;

Если строка «openid» не включена в перечень значений параметра <scope>, конфиденциальный клиент должен:

6) включать параметр <state> (Приложение 1).

6. Предоставление доступа к защищенным ресурсам

Спецификация сервера ресурсов, предоставляющего доступ к защищенным данным

82. Конечные точки ресурсов возвращают клиенту защищенную информацию владельца ресурса, связанного с предъявленным токеном доступа.

83. Сервер ресурсов должен обеспечивать выполнение следующих требований:

1) должен поддерживать метод HTTP GET;

2) должен принимать токены доступа в HTTP заголовке;

3) не должен принимать токены доступа в параметрах запроса;

4) должен проверять, что срок действия токена доступа не истек и токен доступа не отозван;

5) должен проверять, что область действия (параметр <scope>), связанная с предъявленным токеном доступа, разрешает доступ к ресурсу, который запрашивается;

6) должен идентифицировать объект, связанный с токеном доступа;

7) должен возвращать только ресурс, соответствующий объекту, явно указанному в запросе на доступ, и при условии, что доступ к этому ресурсу является допустимым с учетом разрешенной области действия (параметр <scope>), иначе возвращать ошибку, см. Приложение 1, пункт 8;

8) должен кодировать ответ в кодировке UTF-8;

9) должен отправлять параметр <Content-Type> HTTP заголовка в виде «Content-Type: application/json; charset = UTF-8».

Примечания

1 Форматы и алгоритмы формирования UUID определены в RFC 4122 (см. раздел «Библиография», пункт [10]).

2 Для получения информации об объекте, связанном с токеном доступа и предоставленной областью действия, сервер ресурсов может использовать протокол анализа токена, описанный в RFC 7662 (см. раздел «Библиография», пункт [26]).

Спецификация клиента, реализующего доступ к защищенному ресурсу

84. Клиент, реализующий доступ к защищенному ресурсу, должен отправлять токены доступа в заголовке HTTP.

7. Защищенный с использованием JWT режим ответа на запрос авторизации для OAuth 2.0 (JARM)

Режим ответа на основе JWT

85. В защищенном с использованием JWT режиме ответа на запрос авторизации для OAuth 2.0 все параметры ответов передаются в JWT вместе со вспомогательными полями.

86. JWT (см. Приложение 5) содержит следующие базовые параметры, используемые для обеспечения безопасности передачи:

- <iss>: URL сервера авторизации, который сформировал ответ;
- <aud>: <client_id> клиента, для которого предназначен ответ;
- <exp>: окончание срока действия JWT.

Кроме того, JWT содержит параметры ответа конечной точки авторизации, определенные для конкретных типов ответов даже в случае ответа об ошибке. Этот шаблон применим ко всем типам ответов.

Примечание - В JWT также добавляются дополнительные параметры ответа конечной точки авторизации, определяемые расширениями, например <session_state> в соответствии с «OpenID Connect Session Management 1.0» (см. раздел «Библиография», пункт [27]).

87. В случае если разрешение на авторизацию имеет тип «code», JWT содержит параметры <code> и <state>, определенные в пункте 7 Приложения 1.

88. Пример параметров компоненты <payload> токена JWT для успешного ответа типа «code» на запрос авторизации указан на рисунке ниже (Рисунок 5).


```
{
 "iss":"https://accounts.example.com",
 "aud":"s6BhdRkqt3",
 "exp":1311281970,
 "code":"PyyFaux2o7Q0YfXBU32jhw.5FXSQpvr8akv9CeRDSd0QA",
 "state":"S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw"
}
```


Рисунок 5 - Пример параметров компоненты <payload> токена JWT для успешного ответа типа «code» на запрос авторизации

89. В случае ответа об ошибке JWT будет содержать параметры ответа об ошибке, <error>, <error_description>, <error_URI>, <state>, определенные в пункте 8 Приложения 1.

90. В случае если сгенерированное сервером авторизации разрешение, подтверждающее факт авторизации доступа к конфиденциальному ресурсу его владельцем, имеет тип «token», JWT содержит следующие параметры ответа:

- <access_token> – токен доступа;
- <token_type> – тип токена доступа;
- <expires_in> – окончание срока действия токена доступа;
- <scope> – область действия, предоставляемая токеном доступа;
- <state> – значение состояния, отправленное клиентом в запросе авторизации.

91. Пример, демонстрирующий параметры компоненты <payload> токена JWT для успешного ответа типа «token» на запрос авторизации указана на рисунке ниже (Рисунок 6).

```
{
 "iss":"https://accounts.example.com",
 "aud":"s6BhdRkqt3",
 "exp":1311281970,
 "access_token":"2YotnFZFEjr1zCsicMWpAA",
 "state":"S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw",
 "token_type":"bearer",
 "expires_in":"3600",
 "scope":"example"
}
```


Рисунок 6 - Пример параметров компоненты <payload> токена JWT для успешного ответа типа «token» на запрос авторизации

92. В случае ответа об ошибке JWT содержит параметры ответа об ошибке, как в случае типа ответа «code».

Цифровая подпись и шифрование

93. JWT может быть либо подписан (JWS), либо подписан и зашифрован (JWS и JWE). Если JWT подписан и зашифрован, JSON документ должен быть вначале подписан JWS, затем зашифрован JWE, а результатом будет Nested JWT.

94. Сервер авторизации определяет, какой алгоритм использовать для обеспечения защиты JWT для конкретного ответа на запрос авторизации. Это решение может быть основано на зарегистрированных параметрах метаданных для клиента, как определено в подразделе 0.

95. Рекомендации по управлению ключами в целом и в особенности по использованию симметричных алгоритмов для подписи и шифрования на основе ключа клиента см. в пунктах 29-41.

Кодирование ответа

96. Настоящий стандарт определяет следующие значения параметра <response_mode> в составе запроса аутентификации:

- «query.jwt»;
- «fragment.jwt»;
- «form_post.jwt»;
- «jwt».

97. Режим ответа «query.jwt» указывает, что сервер авторизации должен отправлять ответ на запрос авторизации в виде HTTP переадресации на URI переадресации (<redirect_uri>) клиента. Сервер авторизации добавляет параметр <response>, содержащий JWT, как определено в пункте 7, к компоненту <query> <redirect_uri>, используя формат «application/x-www-form-urlencoded».

Примечание - Режим ответа «query.jwt» не должен использоваться в сочетании с типами ответов, которые содержат «token» или «id_token», если JWT ответа не зашифрован для предотвращения утечки токена в URL.

98. Режим ответа «fragment.jwt» указывает, что сервер авторизации должен отправлять ответ на запрос авторизации как HTTP переадресацию на URI переадресации (<redirect_uri>) клиента. Сервер авторизации добавляет параметр <response>, содержащий JWT, как определено в пункте 7, к компоненту <fragment> <redirect_uri>, используя формат «application/x-www-form-urlencoded».

99. Режим ответа «form_post.jwt». Параметр <response>, содержащий JWT, кодируется как значение HTML-формы, которое автоматически передается в агент пользователя и, таким образом, доставляется клиенту с помощью метода HTTP POST, а параметры результата кодируются в HTML-элементе <body> с использованием формата «application/x-www-form-urlencoded».

Примечание - Общие сведения по использованию метода POST для передачи JWT клиенту представлены в «OAuth 2.0 Form Post Response Mode» (см. раздел «Библиография», пункт [24]).

100. Режим ответа «jwt» является ссылкой и указывает используемую по умолчанию кодировку переадресации (<query>, <fragment>) для запрошенного типа ответа. Значением по умолчанию для типа ответа «code» является «query.jwt», для типа ответа «token» и других типов ответов, за исключением «none», является «fragment.jwt».

Правила обработки клиентом, защищенного с помощью JWT ответа

101. Клиент должен хранить на своей стороне состояние сессии, в котором записывается идентификатор сервера авторизации, на который направлен запрос авторизации, и связанная с этим значением информация о браузере (коммуникационном агенте) пользователя.

102. Клиент должен обрабатывать защищенный с помощью JWT ответ следующим образом:

1) (опционально) клиент расшифровывает JWT, используя ключ, определенный параметром заголовка <kid> полученного JWT. Ключ может быть закрытым ключом, соответствующий открытый ключ которого зарегистрирован ожидаемым отправителем ответа («use:enc» в метаданных клиента <jwks> или <jwks_URI>), или ключом, полученным из значения поля client_secret клиента;

2) клиент получает значение параметра JWT <state> и проверяет его привязку к агенту пользователя. Если проверка не пройдена, клиент должен прервать обработку и отказать в ответе;

3) клиент получает параметр JWT <iss>, проверяет, известно ли ему это значение, и идентифицирует ожидаемого отправителя текущей сессии авторизации. Если проверка не пройдена, клиент должен прервать обработку и отказать в ответе;

4) клиент получает параметр JWT <aud> и проверяет, соответствует ли он идентификатору клиента, который использовал клиент для идентификации себя в соответствующем запросе авторизации. Если проверка не пройдена, клиент должен прервать обработку и отказать в ответе;

5) клиент проверяет параметр JWT <exp>, чтобы определить, действителен

ли еще JWT. Если проверка не пройдена, клиент должен прервать обработку и отказать в ответе;

6) клиент получает ключ, необходимый для проверки подписи, используя элемент JWT <iss> и элемент заголовка <kid>, после чего проверяет подпись. Если проверка не пройдена, клиент должен прервать обработку и отказать в ответе.

103. Значение <state> должно обрабатываться как одноразовый CSRF-токен. Он должен быть объявлен недействительным после осуществления проверки (шаг 2).

104. Получение клиентом ключей для проверки подписи JWT (шаг 5) должно выполняться в соответствии с алгоритмами подпункта 35.

105. Пока все проверки не будут успешно пройдены, клиент не должен обрабатывать параметры ответа на запрос авторизации, специфичные для типа разрешения.

Метаданные клиента

106. Названия параметров должны следовать шаблону, установленному динамической регистрацией клиентов OpenID Connect (см. Приложение 3, пункт 5) для конфигурирования алгоритмов подписи и шифрования JWT ответов в конечной точке UserInfo.

107. Определены следующие параметры **метаданных клиента:**

- <authorization_signed_response_alg>: алгоритм цифровой подписи JWS (см. Приложение 5, пункт 1) типа <alg>, который требуется для формирования подписи ответов на запросы авторизации. Если этот параметр определен, ответ будет подписан в соответствии с требованиями JWS (см. Приложение 5, пункт 1) и сконфигурированного алгоритма. Алгоритм «none» не допускается;
- <authorization_encrypted_response_alg>: алгоритм шифрования JWE (см. Приложение 5, пункт 2) типа <alg>, который требуется для шифрования ответов на запросы авторизации. Если требуются и подписание, и шифрование, ответ будет подписан, затем зашифрован, и результатом будет структура Nested JWT. По умолчанию, если не указано, шифрование не выполняется;
- <authorization_encrypted_response_enc>: алгоритм шифрования, который требуется для шифрования ответов на запросы авторизации, если определен <authorization_encrypted_response_alg>. Если включен <authorization_encrypted_response_enc>, также должен присутствовать <authorization_encrypted_response_alg>.

108. В процессе регистрации на сервере авторизации клиенты могут передавать последнему ссылки на ресурсы хранения своих открытых ключей, используя параметры метаданных <jwks_URI> или <jwks> (см. Приложение 3, пункт 5).

Метаданные сервера авторизации

109. Сервер авторизации при публикации типов поддерживаемых алгоритмов формирования цифровой подписи и шифрования JWT ответа на запрос авторизации должен использовать параметры метаданных в соответствии с пунктом 1 (Приложение 3).

110. Определены следующие параметры поддерживаемых алгоритмов формирования цифровой подписи и **шифрования:**

- <authorization_signing_alg_values_supported>: (опционально) JSON-массив, содержащий список значений типа <alg> алгоритмов подписи JWS, поддерживаемых конечной точкой авторизации для подписания ответа;
- <authorization_encryption_alg_values_supported>: (опционально) JSON-массив, содержащий список значений типа <alg> алгоритмов шифрования JWE, поддерживаемых конечной точкой авторизации для шифрования ответа;
- <authorization_encryption_enc_values_supported>: (опционально) JSON-массив, содержащий список значений типа <enc> алгоритмов шифрования JWE, поддерживаемых конечной точкой авторизации для шифрования ответа.

111. Сервер авторизации при публикации значения поддерживаемых режимов ответа должен использовать параметр <response_modes_supported> в соответствии с пунктом 1 (Приложение 3). В настоящем стандарте определены следующие возможные значения этого параметра:

- «query.jwt»;
- «fragment.jwt»;
- «form_post.jwt»;
- «jwt».

112. Клиент должен сначала проверить, что эмитент JWT известен и допустим для конкретного ответа авторизации, прежде чем использовать эти данные для получения ключа, необходимого для проверки подписи JWT. Данное требование связано с защитой от атак типа отказ в обслуживании, так как JWT могут быть созданы специальным образом, чтобы JWT содержал URL-адрес с огромным содержимым или доставлялся очень медленно, потребляя пропускную способность сети и вычислительные мощности.

113. Для предотвращения вида атак типа смешивание (атака на сценарии, в которых клиент OAuth взаимодействует с несколькими серверами авторизации для получения кода авторизации или токена доступа, обманом заставив клиента отправить эти учетные данные злоумышленнику вместо того, чтобы использовать их в соответствующей конечной точке на сервере авторизации/ресурсов) должен использоваться режим защищенного ответа JWT, который предоставляет идентификацию отправителя (iss) и предполагаемую аудиторию ответа авторизации (aud).

114. Для предотвращения утечки кодов авторизации в пользовательском агенте (например, во время передачи, в истории браузера или через заголовки referrer) сервер авторизации может зашифровать ответ авторизации (опционально).

Приложение 1
к документу «Межбанковская система
обмена информацией
по открытым программным
интерфейсам (Open API)
Стандарт информационной безопасности
Профили безопасности Open API»
(информационное)

Протокол OpenID Connect. Аутентификация с генерацией кода авторизации

1. Сценарий протокола аутентификации OpenID Connect с генерацией кода авторизации использует код авторизации в качестве типа разрешения на доступ (grant). При этом выполняются следующие действия:

1)Клиент генерирует запрос аутентификации;

2)Клиент, используя агент пользователя (User Agent), отправляет запрос аутентификации на конечную точку авторизации сервера авторизации;

3)Сервер авторизации аутентифицирует конечного пользователя;

4)Сервер авторизации получает разрешение конечного пользователя на доступ клиента к защищенным ресурсам;

5)Сервер авторизации генерирует код авторизации и перенаправляет агент пользователя на конечную точку клиента, включая значение сгенерированного кода авторизации в состав параметров запроса на переадресацию;

6)Клиент запрашивает идентификационный токен и токен доступа, используя код авторизации на конечной точке токена;

7) Клиент получает ответ, содержащий идентификационный токен и токен доступа;

8)Клиент проверяет идентификационный токен (см. Приложение 1, пункт 16) и получает идентификатор конечного пользователя.

2. Серверы авторизации должны поддерживать на конечной точке авторизации методы HTTP GET и POST. Клиенты могут использовать методы HTTP GET или POST для отправки запроса аутентификации на сервер авторизации. При использовании метода HTTP GET параметры запроса сериализуются с использованием URI Query String Serialization (см. раздел «Библиография», пункт [3]). При использовании метода HTTP POST параметры запроса сериализуются с использованием сериализации форм (Form Serialization) (см. раздел «Библиография», пункт [3]).

3. Также параметры запроса аутентификации могут быть переданы в качестве параметров (claims) JSON объекта запроса – JWT токена. JWT токен должен быть подписан и может быть зашифрован. JWT токен передается в кодировании Base64url согласно RFC 4648 (см. раздел «Библиография», пункт [9])

по значению через параметр <request> или по ссылке через параметр <request_uri>.

Примечание - Дополнительные сведения по передаче запроса авторизации от клиента к серверу авторизации приведены в «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3]).

4. Сервер авторизации должен проверить полученный запрос следующим образом:

1)Проверить наличие обязательных параметров запроса и их соответствие спецификациям OAuth 2.0 и OpenID Connect;

2)В случае использования протокола OpenID Connect, убедиться, что параметр <scope> содержит значение области действия «openid»;

3)Проверить, что все обязательные параметры присутствуют и их использование соответствует «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3]).

Серверу авторизации рекомендуется игнорировать неопознанные параметры запроса. В случае если хотя бы одна из вышеперечисленных проверок не пройдена, сервер авторизации должен вернуть ошибку в соответствии с пунктом 8 Приложения 1.

5. Если запрос аутентификации действителен, сервер авторизации пытается аутентифицировать конечного пользователя или определяет, аутентифицирован ли конечный пользователь. Сервер авторизации должен запрашивать аутентификацию конечного пользователя в следующих случаях:

1)конечный пользователь еще не аутентифицирован;

2)запрос аутентификации содержит параметр <prompt> со значением «login». В этом случае сервер авторизации должен повторно аутентифицировать конечного пользователя, даже если конечный пользователь уже аутентифицирован.

Сервер авторизации не должен взаимодействовать с конечным пользователем в следующем случае:

- запрос аутентификации содержит параметр <prompt> со значением «none». В этом случае сервер авторизации должен возвратить ошибку, если конечный пользователь еще не прошел аутентификацию или не может быть аутентифицирован в режиме без вывода сообщений.

6. После аутентификации конечного пользователя сервер авторизации должен получить его разрешение на доступ к запрошенному ресурсу, прежде чем предоставлять информацию. В случае если это определено параметрами запроса, разрешение может быть получено одним из следующих способов:

1)через интерактивный диалог с конечным пользователем, в ходе которого сервер авторизации отображает запрашиваемую клиентом область действия и запрашивает у конечного пользователя согласие на доступ;

2)путем установления согласия с помощью условий обработки запроса;

3)другими способами.

7. После успешной аутентификации конечного пользователя и получения его разрешения на доступ клиента к защищенному ресурсу сервер авторизации генерирует код авторизации и передает его на конечную точку клиента, воспользовавшись адресом, который был ранее указан в параметре <redirect_uri> запроса аутентификации, с использованием формата «application/x-www-form-urlencoded». Параметры успешного ответа на запрос аутентификации:

- <code>: (обязательный) значение кода авторизации; размер строки кода авторизации должен определяться при проектировании сервера авторизации;
- <state>: (обязательный, если параметр <state> присутствует в запросе аутентификации) значение параметра <state> запроса аутентификации, полученного от клиента.

8. В случае ошибки при проверке запроса аутентификации либо в процессе аутентификации конечного пользователя сервер авторизации в ответе передает информацию об ошибке.

9. Получив успешный ответ на запрос аутентификации, клиент проверяет его. Клиент должен игнорировать нераспознанные параметры ответа. Клиент должен проверить соответствие длин параметров установленным при разработке сервера авторизации ограничениям. Клиент должен проверить, совпадает ли значение параметра <state>, полученное в составе ответа, со значением параметра <state> в запросе аутентификации, переданным клиентом.

10. В случае успешной проверки ответа сервера авторизации клиент формирует и отправляет на адрес конечной точки токена запрос токена. В запросе токена передаются следующие сведения:

- <grant_type>: (обязательный) тип разрешения; в данном случае значением должна быть строка «authorization_code»;
- <code>: (обязательный) значение кода авторизации, полученное от сервера авторизации;
- <redirect_uri>: (обязательный) URI переадресации, на который будет отправлен ответ; должен быть предварительно зарегистрирован на сервере авторизации; значение этого параметра должно совпадать со значением параметра <redirect_uri> запроса авторизации;
- <client_id>: (обязательный) идентификатор клиента, зарегистрированный сервером авторизации.

Примечание - Дополнительные сведения по запросу токена см. в «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3]) и в RFC 6749 (см. раздел «Библиография», пункт [7]).

11. Сервер авторизации должен проверить запрос токена следующим образом:

- аутентифицировать клиента в соответствии с подразделом 0;
- убедиться, что код авторизации был выдан аутентифицированному клиенту;

- убедиться, что код авторизации действителен;
- убедиться, что код авторизации ранее не использовался;
- убедиться, что значение параметра <redirect_uri> совпадает со значением параметра <redirect_uri>, которое было включено в начальный запрос авторизации. Если значение параметра <redirect_uri> отсутствует при наличии только одного зарегистрированного значения <redirect_uri>, сервер авторизации может вернуть ошибку (так как клиент должен был включить параметр) или может работать без ошибки (так как OAuth 2.0 разрешает опускать этот параметр в таком случае);

Примечание - Реакция сервера на такой запрос определяется при его разработке, исходя из его прикладных целей и задач.

- убедиться, что использованный код авторизации был выдан в ответ на запрос аутентификации OpenID Connect.

12. Коды авторизации должны быть кратковременными и одноразовыми. Если сервер авторизации регистрирует несколько попыток обмена одного и того же кода авторизации на токен доступа, серверу авторизации следует отозвать все токены доступа, уже предоставленные на основе скомпрометированного кода авторизации.

Сервер авторизации должен аутентифицировать клиента (см. подраздел 0) и гарантировать, что код авторизации был выдан клиенту, запросившему его.

13. Получив и успешно проверив запрос токена, сервер авторизации генерирует идентификационный токен (см. Приложение 1, пункт 15), токен доступа (см. Приложение 1, пункт 17), а также, если необходимо, токен обновления (см. Приложение 1, пункт 18) и возвращает их клиенту по адресу <redirect_uri> в теле HTTP ответа с кодом состояния 200 (OK). В ответе используется тип формата «application/json». Параметры JSON структуры ответа на запрос токена:

- <access_token>: (обязательный) токен доступа;
- <token_type>: (обязательный) тип токена, должен иметь значение «Bearer»;
- <expires_in>: (рекомендуемый) время жизни токена в секундах;
- <refresh_token>: (опциональный) токен обновления;
- <scope>: (опциональный) область действия токена доступа;
- <id_token>: (обязательный) идентификационный токен, связанный с текущей сессией доступа.

В случае ошибки проверки запроса токена сервер авторизации должен ответить сообщением об ошибке. Параметры и формат сообщения об ошибке приведены в пункте 8 Приложения 1. В теле HTTP ответа используется тип «application/json» с кодом HTTP ответа 400.

Примечание - Дополнительные сведения по параметрам ответа на запрос токена и процедурам работы с ними см. в «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3] (подпункты 3.1.3.3 и 3.1.3.4)).

14. Клиент должен проверять правильность ответа на запрос токена следующим образом:

- клиент должен игнорировать нераспознанные параметры ответа;
- клиент должен прекратить проверку, если отсутствует значение хотя бы одного из обязательных параметров;
- клиент должен сверить длины параметров с установленными в системе;
- клиент должен проверить правильность идентификационного токена.

15. Идентификационный токен представляется в виде структуры JWT (см. Приложение 5, пункт 4).

16. Проверка идентификационного токена клиентом должна выполняться следующим образом:

1)Если идентификационный токен зашифрован, следует расшифровать его, используя ключи и алгоритмы, которые сервер авторизации должен был использовать для шифрования идентификационного токена как JWE (см. Шифрование и Приложение 5, пункт 2);

2)Идентификатор эмитента (сервера авторизации), полученный при регистрации клиента, должен точно соответствовать значению параметра <iss> идентификационного токена;

3)Клиент должен убедиться, что значение параметра <aud> содержит значение <client_id>, полученное клиентом от сервера авторизации при регистрации. Идентификационный токен должен быть отклонен, если <aud> не содержит значение <client_id> клиента;

4)Если идентификационный токен содержит несколько компонент в составе параметра <aud>, клиент должен проверить наличие параметра <azp> и при его наличии проверить, что значение равно <client_id>;

5)Клиент должен проверить подпись структуры JWS идентификационного токена, применяя алгоритм, указанный в параметре <alg>, используя ключ, предоставленный сервером авторизации. Если идентификационный токен получен посредством прямого обмена данными между клиентом и конечной точкой токена, проверка сообщений протокола TLS от сервера может использоваться для проверки эмитента вместо проверки подписи токена;

6)Значение параметра <alg> должно быть значением по умолчанию или алгоритмом, зарегистрированным клиентом в значении параметра <id_token_signed_response_alg> во время регистрации;

7) Если в значении параметра <alg> указан алгоритм MAC, то в качестве ключа должны использоваться октеты UTF-8 представления <client_secret>, соответствующего <client_id> клиента;

8)Текущее время проверки должно быть раньше времени, указанного в значении параметра <exp>;

9)Для отзыва токенов, выпущенных слишком давно, может использоваться параметр <iat>, который ограничивает время, необходимое для хранения значений

параметра <nonce>. Приемлемый диапазон определяется при разработке сервера авторизации;

10) Если в запросе аутентификации, отосланном клиентом в адрес сервера авторизации, указано значение параметра <nonce>, в структуре идентификационного токена также должно присутствовать значение параметра <nonce>. Следует убедиться, что эти два значения совпадают;

11) Если запрашивалось значение параметра <auth_time> либо посредством специального запроса этого параметра, либо указанием параметра <max_age> в запросе аутентификации, клиенту следует проверить значение параметра <auth_time> на допустимость диапазону <max_age> и сделать запрос повторной аутентификации, если он определяет, что с момента последней аутентификации конечного пользователя прошло слишком много времени.

Примечание - Дополнительные сведения об использовании идентификационного токена см. в «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3] (раздел 2)).

17. Токен доступа содержит указание на конкретные области действия, к которым разрешен доступ, длительность доступа и другие параметры. Рекомендуется время жизни токена ограничить однократным использованием или коротким временем жизни.

Сервер авторизации должен во время авторизации уведомлять владельца ресурса о том, что он должен обратить внимание на запрос долгосрочного разрешения на доступ клиента к защищенному ресурсу. Сервер авторизации должен предоставить конечному пользователю механизм отзыва токенов доступа и токенов обновления, предоставленных клиенту.

Примечание - Рекомендации по использованию токенов доступа см. в «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3] (подраздел 16.18)).

18. Токен обновления выдается клиенту сервером авторизации и используется для получения нового токена доступа, когда текущий токен доступа становится недействительным или истекает его срок действия, или для получения дополнительных токенов доступа с идентичной или более узкой областью действия.

Выдача токена обновления необязательна и выполняется по усмотрению сервера авторизации. Если сервер авторизации выдает токен обновления, он включается в ответ при выдаче токена доступа. Данная функциональность не входит в область действия настоящего документа и определяется на этапе разработки сервера авторизации, исходя из его прикладных целей и задач.

19. Для постоянного доступа к конечной точке UserInfo или другим

защищенным ресурсам используется токен обновления. Клиент может обменять токен обновления в конечной точке токена на новый кратковременный токен доступа, который можно использовать для доступа к ресурсу.

Решение о необходимости выдачи токена обновления принимается на этапе разработки сервера авторизации.

Приложение **2**
к документу «Межбанковская система
обмена информацией
по открытым программным
интерфейсам (Open API)
Стандарт информационной безопасности
Профили безопасности Open API»
(информационное)

Протокол OpenID Connect. Аутентификация по гибридному сценарию

1. При использовании гибридного сценария некоторые токены возвращаются из конечной точки авторизации, другие – из конечной точки токена.

Гибридный сценарий состоит из следующих действий:

1)клиент генерирует запрос аутентификации, содержащий желаемые параметры запроса;

2)клиент отправляет запрос на сервер авторизации;

3)сервер авторизации аутентифицирует конечного пользователя;

4)сервер авторизации получает согласие/авторизацию конечного пользователя;

5)сервер авторизации возвращает конечного пользователя на клиент с передачей кода авторизации и в зависимости от типа ответа – одного или нескольких дополнительных параметров;

6)клиент запрашивает ответ на конечной точке токена, используя код авторизации;

7) клиент получает ответ, содержащий идентификационный токен и токен доступа;

8)клиент проверяет идентификационный токен и извлекает идентификатор субъекта конечного пользователя (значение параметра <sub>).

2. При использовании гибридного сценария действия сервера авторизации и клиента, связанные с формированием запроса аутентификации, его передачей, проверкой, а также аутентификацией пользователя и получением его согласия, в целом те же, что и в сценарии с кодом авторизации. При этом параметр <response_type> запроса аутентификации должен иметь одно из следующих значений: «code id_token», либо «code token», либо «code id_token token».

3. В случае ошибки проверки запроса аутентификации ответ возвращается клиенту, как в подпункте 8 Приложения 1. Если конечный пользователь отклоняет запрос или аутентификация конечного пользователя завершается неудачно, сервер авторизации должен возвратить ответ об ошибке авторизации в компоненте фрагмента URI переадресации.

В случае успешного ответа на запрос аутентификации все параметры ответа добавляются к компоненте фрагмента URI переадресации. При этом клиенту

возвращаются значения следующих параметров:

- <access_token>: токен доступа; обязательный, если значение параметра <response_type> запроса аутентификации равно «code token» или «code id_token token»;
- <token_type>: тип токена; обязательный, если значение параметра <response_type> запроса аутентификации равно «code token» или «code id_token token»; если присутствует, значением должен быть тип «Bearer»;
- <id_token>: идентификационный токен; обязательный, если значением <response_type> является «code id_token» или «code id_token token»;
- <code>: (обязательный) код авторизации;
- <state>: (обязательный, если в составе запроса аутентификации присутствует значение параметра <state>) значение параметра <state> из запроса аутентификации; клиент должен проверить равенство возвращенного значения параметра <state> значению параметра <state>, переданному им в составе запроса аутентификации;
- <expires_in>: (опциональный) время жизни токена доступа в секундах с момента генерации ответа.

4. Формат идентификационного токена в данном сценарии аналогичен подпункту 15 Приложения 1 со следующими дополнениями:

- параметр <nonce> обязательный;
- добавлен обязательный параметр <at_hash>: хэш-значение токена доступа; вычисляется сервером авторизации и проверяется клиентом как Base64url кодирование левой половины хэшкода октетов ASCII представления значения <access_token>; если токен доступа не запрашивается, значение параметра <at_hash> должно отсутствовать;
- добавлен обязательный параметр <c_hash>, значение которого вычисляется сервером авторизации и проверяется клиентом как Base64url кодирование левой половины значения хэш-функции октетов ASCII представления значения параметра <code>.

5. Клиент должен убедиться в правильности ответа аутентификации:

- следуя правилам 14 Приложения 1;
- проверив целостность токена доступа (если значение параметра <response_type> запроса аутентификации равно «code token» или «code id_token token») путем сравнения значения Base64url кодирования левой половины хэш-кода полученного значения <access_token> с полученным значением параметра <at_hash>;
- проверив целостность кода авторизации (если значение параметра <response_type> запроса аутентификации равно «code id_token» или «code id_token token») путем сравнения значения Base64url кодирования левой половины хэш-кода полученного значения <code> с полученным значением параметра <c_hash>.

6. При использовании гибридного сценария запрос токена клиентом, проверка запроса токена, генерация токена доступа и идентификационного токена

на конечной точке токена, их проверка клиентом, а также реакция на ошибки выполняются в соответствии с аналогичными действиями в сценарии с генерацией кода авторизации (см. Приложение 1, подпункты 10-14).

Если идентификационный токен возвращается как из конечной точки авторизации, так и из конечной точки токена, что имеет место для одного из двух значений <response_type>: «code id_token» и «code id_token token», значения <iss> и <sub> должны быть идентичны в обоих идентификационных токенах. Все значения параметров события аутентификации, присутствующие в любом из них, должны присутствовать в обоих идентификационных токенах. Если какой-либо идентификационный токен содержит параметр конечного пользователя, присутствующий в обоих идентификационных токенах, он должен иметь одинаковые значения в обоих идентификационных токенах. Сервер авторизации может вернуть меньшее количество параметров о конечном пользователе из конечной точки авторизации, например по соображениям конфиденциальности.

Примечание - Дополнительные сведения о гибридном сценарии аутентификации приведены в «OpenID Connect Core 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [3] (подраздел 3.3)).

Приложение 3
к документу «Межбанковская система
обмена информацией
по открытым программным
интерфейсам (Open API)
Стандарт информационной безопасности
Профили безопасности Open API»
(информационное)

Протокол OpenID Connect. Регистрация сервера авторизации (OpenID Connect Discovery) и динамическая регистрация клиента

1. Прежде чем запустить сервис аутентификации и авторизации клиентов, сервер авторизации должен опубликовать свои метаданные, описывающие его адрес и параметры доступа в ходе процедуры OpenID Connect Discovery.
2. Настоящий стандарт не регламентирует способ получения адреса сервера авторизации. Этот адрес может предоставляться клиенту, например, указанием его в документации.
3. Клиент может, используя HTTP GET запрос по адресу сервера авторизации, получить конфигурацию сервера авторизации (документ Discovery), в которой в JSON формате перечислены метаданные сервера авторизации. В частности, сведения о конфигурации должны содержать значения следующих параметров:

- <issuer>: (обязательный) https URL адрес, являющийся идентификатором эмитента (Issuer Identifier); далее это значение должно быть указано в качестве параметра <iss> в идентификационных токенах; получив ответ, клиент должен проверить совпадение значения параметра <issuer> с адресом сервера авторизации, на который он делал запрос;
- <authorization_endpoint>: (обязательный) UR конечной точки авторизации; может включать в себя компоненту URI запроса (query) в формате «application/x-www-form-urlencoded»; не должен включать компоненту URI фрагмент (fragment);
- <token_endpoint>: (обязательный) URI конечной точки токена (см. Приложение 1, пункты 9-13); может включать в себя компоненту URI запрос (query) в формате «application/x-www-form-urlencoded»; не должен включать компоненту URI фрагмент (fragment);

Примечание – Параметр <token_endpoint> может быть необязательным в неявном сценарии аутентификации (Implicit Flow), который не предполагается к использованию в рамках данного стандарта (см. подраздел 13).

- <userinfo_endpoint>: (рекомендуемый) URI конечной точки UserInfo, предназначенный для запроса информации об аутентифицированном конечном

пользователе;

- <registration_endpoint>: (рекомендуемый) URI конечной точки динамической регистрации клиентов сервера авторизации (см. Приложение 3, пункт 5);
- <jwks_uri>: (обязательный) URL документа JSON Web Key Set (см. Приложение 5, пункт 3.3) сервера авторизации, где публикуются его ключи в формате JWK;
- <grant_types_supported>: (опциональный) перечень поддерживаемых сервером авторизации типов разрешений на доступ (grants) в виде JSON массива; Сервера авторизации OpenID Connect должны поддерживать значения «authorization_code» и «implicit»; если параметр не задан, то по умолчанию используется значение [«authorization_code», «implicit»];
- <scopes_supported>: (рекомендуемый) JSON массив значений параметра <scope>, которые поддерживает сервер авторизации; Сервер авторизации должен поддерживать значение <openid>;
- <response_types_supported>: (обязательный) JSON массив с перечнем поддерживаемых значений параметра <response_type>; сервера авторизации OpenID Connect должны поддерживать значения «code», «id_token» и «token id_token»;
- <response_modes_supported>: (опциональный) JSON массив с перечнем поддерживаемых значений параметра <response_mode>; если параметр не задан, то по умолчанию [«query», «fragment»]; также в подразделе 0 приведен перечень значений <response_mode>, связанный с использованием технологии JARM;
- <claims_supported>: (рекомендуемый) JSON массив, содержащий список имен параметров, значения которых сервер авторизации может предоставить клиенту;
- service_documentation: (опциональный) URL-адрес страницы, содержащей справочную информацию, которая представляется разработчиками или которую нужно знать при использовании сервера авторизации. В частности, если сервер авторизации не поддерживает динамическую регистрацию клиентов, в этой документации должна быть представлена информация о том, как регистрировать клиентов.

В подразделе 0 приведен перечень дополнительных метаданных сервера авторизации, связанных с использованием технологии JARM. Адреса всех перечисленных выше конечных точек сервера авторизации должны быть разными.

Примечание – Дополнительные сведения о протоколе OpenID Connect Discovery представлены в документах «OpenID Connect Discovery 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [11]) и RFC 8414 (см. раздел «Библиография», пункт [12]).

4. Передача сообщений OpenID Connect Discovery между клиентом и сервером авторизации должна быть защищена с помощью протокола TLS с

обеспечением конфиденциальности и целостности. При этом должна выполняться аутентификация источника информации OpenID Connect Discovery.

5. Процедура динамической регистрации клиента выполняется после того, как выполнена регистрация сервера авторизации, описанная в пунктах 1-4 Приложения 3.

Для регистрации клиент посылает HTTP POST запрос серверу авторизации на конечную точку регистрации клиентов (параметр <registration_endpoint> документа OpenID Connect Discovery, см. Приложение 3, пункт 3) с типом содержимого «application/json», в котором указываются следующие метаданные клиента:

- <redirect_uris>: (обязательный) перечень поддерживаемых адресов переадресации клиента; одно из этих зарегистрированных значений URI переадресации должно точно соответствовать значению параметра <redirect_uri>, используемому в каждом запросе авторизации;
- <response_types>: (опциональный) JSON массив, содержащий перечень поддерживаемых клиентом типов ответа; по умолчанию – значение «code»;
- <grant_types>: (опциональный) JSON массив, содержащий перечень поддерживаемых клиентом типов разрешений на доступ (grants) из перечня: «authorization_code», «implicit», «refresh_token»; по умолчанию используется значение «authorization_code»;

Примечание – Ниже приведена таблица значений <response_type> и соответствующих им значений <grant_type>:

- code: authorization_code
- id_token: implicit
- token id_token: implicit
- code id_token: authorization_code, implicit
- code token: authorization_code, implicit
- code token id_token: authorization_code, implicit

- <application_type>: (опциональный) тип клиента («native» или «web»); по умолчанию – значение «web»;
- <contacts>: (опциональный) массив адресов электронной почты лиц, ответственных за этого клиента;
- <client_name>: (опциональный) имя клиента; оно будет показано конечному пользователю при запросе авторизации;
- <logo_uri>: (опциональный) URL ссылка на логотип клиентского приложения; если присутствует, сервер должен отображать это изображение конечному пользователю во время запроса авторизации;
- <client_uri>: (опциональный) URL домашней страницы клиента; если присутствует, сервер должен отображать этот URL-адрес конечному пользователю;

- <policy_uri>: (опциональный) URL-адрес, который клиент предоставляет конечному пользователю с информацией о том, как будут использоваться данные его профиля; если присутствует, сервер авторизации должен показать этот URL-адрес конечному пользователю;

- <tos_uri>: (опциональный) URL-адрес, который клиент предоставляет конечному пользователю для ознакомления с условиями обслуживания клиента; если присутствует, сервер авторизации должен показать этот URL-адрес конечному пользователю;

- <jwks_uri>: (опциональный) URI документа JWK Set (см. Приложение 5, пункт 3.3), содержащего ключи клиента в формате JWK;

- <jwks>: (опциональный) документ JWK Set, передаваемый по значению; параметры <jwks_uri> и <jwks> не должны одновременно присутствовать в метаданных клиента;

- <default_max_age>: (опциональный) максимальный период аутентификации по умолчанию; указывает, что конечный пользователь должен быть повторно аутентифицирован, если с момента его аутентификации прошло более чем указанное количество секунд;

- <require_auth_time>: (опциональный) логическое значение, указывающее, требуется ли указывать параметр <auth_time> в идентификационном токене; значением по умолчанию является «false»;

- <token_endpoint_auth_method>: (опциональный) поддерживаемый метод аутентификации клиента на конечной точке токена; возможные варианты: «client_secret_post», «client_secret_basic», «client_secret_jwt», «private_key_jwt», «tls_client_auth», «self_signed_tls_client_auth» и «none» (см. подраздел 0).

6. Сервер авторизации может игнорировать значения, предоставленные клиентом, и должен игнорировать любые присланные клиентом метаданные, которые он не смог распознать. Он должен проверить все заявленные значения метаданных клиента на их совместимость со спецификациями OpenID Connect, а также с ограничениями, которые устанавливаются при разработке сервера авторизации. Сервер авторизации может выбрать замену действительного значения для любого запрошенного параметра метаданных клиента.

В случае если все запрошенные значения параметров клиента корректны, сервер авторизации должен, используя код HTTP 201, возвратить клиенту JSON документ с типом содержимого «application/json» со следующей информацией:

- <client_id>: (обязательный) уникальный идентификатор клиента, который далее будет использоваться клиентом в протоколах OpenID Connect;

- <client_secret>: (опциональный) конфиденциальная информация, которая возвращается конфиденциальным клиентам и используется для аутентификации клиента сервером авторизации, а также для защиты взаимодействия клиента и сервера авторизации; значение этого параметра обязательно при использовании механизма аутентификации клиента <client_secret_jwt>;

- <client_id_issued_at>: (опциональный) время выдачи идентификатора

клиента; значение представляет собой число, представляющее количество секунд от 1970-01-01T0:0:0Z UTC до текущего момента;

- <client_secret_expires_at>: (обязательный, если присутствует <client_secret>) время окончания действия <client_secret> или 0, если он не устаревает никогда; значение представляет собой число, представляющее количество секунд от 1970-01-01T0:0:0Z UTC до текущего момента.

Кроме того, сервер авторизации возвращает клиенту в составе ответа метаданные клиента, принятые сервером в запросе регистрации.

Должна быть обеспечена конфиденциальность передачи значения параметра <client_secret>. Механизм защиты должен быть определен на этапе разработки сервера авторизации.

7. При ошибке обработки запроса регистрации (см. Приложение 3, пункт 6) сервер авторизации не выполняет регистрацию клиента, а конечная точка регистрации возвращает клиенту код состояния HTTP 400, включая JSON объект, описывающий ошибку в теле ответа.

JSON объект, описывающий ошибку, содержит два обязательных параметра <error> и <error_description> в формате, описанном в пункте 8 Приложения 1. Коды ошибки:

- «invalid_redirect_uri»: значение одного или нескольких значений массива <redirect_uris> недопустимо;
- «invalid_client_metadata»: значение одного из полей метаданных клиента недопустимо, и сервер отклонил этот запрос.

Примечание – Дополнительные сведения о динамической регистрации клиента приведены в документах «OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1» (см. раздел «Библиография», пункт [13]) и RFC 7591 (см. раздел «Библиография», пункт [14]).

8. Передача информации между клиентом и конечной точкой регистрации сервера авторизации должна быть защищена с помощью протокола TLS с аутентификацией сервера авторизации.

Приложение 4
к документу «Межбанковская система
обмена информацией
по открытым программным
интерфейсам (Open API)
Стандарт информационной безопасности
Профили безопасности Open API»
(информационное)

Описание механизмов аутентификации

1. Механизм аутентификации «client_secret_jwt».

1.1. Клиент, получивший при регистрации значение <client_secret> от сервера авторизации, создает токен JWT, используя для обеспечения его целостности алгоритмы вычисления кода аутентификации сообщения HMAC на основе хэш-функции и значение <client_secret> в качестве ключа.

Данный механизм использует следующие параметры JWT (см. Приложение 5, пункт 4):

- <iss>: (обязательный) источник; должен содержать <client_id> клиента;
- <sub>: (обязательный) субъект; должен содержать <client_id> клиента;
- <aud>: (обязательный) аудитория; значением должен быть URL-адрес конечной точки токена сервера авторизации;
- <jti>: (обязательный) идентификатор JWT; уникальный идентификатор токена, который необходим для предотвращения повторного использования токена;
- <exp>: (обязательный) срок действия JWT - время, по истечении которого токен не должен быть принят в обработку;
- <iat>: (опциональный) время выпуска JWT.

1.2. Токен JWT должен быть отправлен клиентом серверу авторизации в качестве значения параметра <client_assertion>. Значение параметра <client_assertion_type> должно быть «urn:ietf:params:-oauth:client-assertion-type:jwt-bearer». Эти параметры передаются в качестве утверждений (assertions) методом HTTP POST в адрес конечной точки авторизации с типом содержимого «application/x-www-form-urlencoded».

1.3. При использовании JWT для аутентификации клиента сервер авторизации должен проверить JWT в соответствии с приведенными ниже критериями:

1)JWT должен содержать значение параметра <iss> (эмитент);

2)JWT должен содержать значение параметра <sub> (субъект), идентифицирующего субъект JWT. Для аутентификации клиента это значение должно совпадать с <client_id> клиента;

3)JWT должен содержать значение параметра <aud> (аудитория), которое идентифицирует сервер авторизации как предполагаемую аудиторию. В качестве значения элемента <aud> может использоваться URL-адрес конечной точки токена сервера авторизации. Сервер авторизации должен отклонить JWT, который не содержит своего идентификатора в качестве аудитории;

4)JWT должен содержать значение параметра <exp> (время действия), ограничивающее временное окно, в течение которого JWT может обслуживаться. Сервер авторизации должен отклонить любой JWT с истекшим временем действия. Сервер авторизации может отклонить JWT с необоснованно большим значением параметра <exp>;

5)JWT может содержать значение параметра <nbf> (не ранее), определяющее время, до которого токен не должен приниматься для обработки;

6)JWT может содержать значение параметра <iat> (время выпуска). Сервер авторизации может отклонять JWT со значением <iat>, которое указывает, что токен был выпущен необоснованно давно;

7) JWT может содержать значение параметра <jti> (JWT ID), которое предоставляет уникальный идентификатор токена. Сервер авторизации должен гарантировать, что JWT не будут повторно воспроизведены, посредством ведения сведений о перечне используемых значений <jti> (значения <jti> хранятся на протяжении времени, в течении которого JWT будет считаться действительным, на основе значения <exp>);

8)JWT может содержать другие параметры, значение которых не устанавливается в настоящем стандарте;

9)JWT должен иметь цифровую подпись или MAC, применяемый эмитентом. Сервер авторизации должен проверить подпись JWT как JWS и отклонить JWT с недопустимой подписью или MAC.

1.4. В случае ошибки проверки JWT сервер авторизации должен ответить кодом ошибки «invalid_client» в качестве значения параметра <error> (см. Приложение 1, пункт 8). В случае успешной проверки JWT клиент считается аутентифицированным.

Примечание – Дополнительные сведения о данном методе аутентификации «client_secret_jwt» приведены в RFC 7521 (см. раздел «Библиография», пункт [15]) и RFC 7523 (см. раздел «Библиография», пункт [16]).

1.5. Для обеспечения безопасности передачи сообщений аутентификации клиента должен использоваться протокол TLS.

2. Механизм аутентификации «private_key_jwt».

2.1. Клиент, который зарегистрировал ключ проверки цифровой подписи, подписывает JWT, используя соответствующий ключ цифровой подписи для

обеспечения целостности JWT. Перечень параметров JWT, а также процедура передачи такие же, как в пункте 1 Приложения 4.

3. Механизм аутентификации с использованием протокола TLS с взаимной аутентификацией и PKI (tls_client_auth).

3.1. Механизм аутентификации клиента с использованием протокола TLS с взаимной аутентификацией и PKI требует выполнения следующих действий:

1)В ходе динамической регистрации (см. Приложение 3, пункт 5) клиент должен получить у сервера авторизации идентификатор <client_id>;

2)В ходе динамической регистрации клиент должен зарегистрировать механизм аутентификации «tls_client_auth»;

3)В ходе динамической регистрации клиент должен зарегистрировать ровно одно из имен субъекта сертификата клиента в качестве значения одного из следующих параметров:

- <tls_client_auth_subject_dn>: отличительное имя субъекта сертификата;
- <tls_client_auth_san_dns>: значение записи SAN dNSName в сертификате;
- <tls_client_auth_san_uri>: значение записи SANiformResourceIdentifier в сертификате;
- <tls_client_auth_san_ip>: IP-адрес IPv4 или IPv6 из записи iPAddress SAN в сертификате;
- <tls_client_auth_san_email>: значение записи SAN rfc822Name в сертификате.

4)У клиента должен быть установлен сертификат открытого ключа формата X.509, в котором указано одно отличительное имя субъекта (subject distinguished name, DN) и одно альтернативное имя субъекта (SAN). Сертификат должен успешно проходить процедуру валидации цепочки сертификатов;

Примечание – Общие требования к процедуре валидации цепочки сертификатов открытого ключа формата X.509 приведены в RFC 5280 (см. раздел «Библиография», пункт [17]).

5)При установлении TLS-соединения сервер авторизации должен потребовать сертификат клиента и проверить его валидность, а также получить подтверждение владения закрытым ключом, соответствующим открытому ключу, указанному в сертификате;

6)Сервер авторизации на основании предъявленного значения <client_id> должен извлечь из данных регистрации клиента его имя (см. перечисление в пункте 3) данного списка) и сравнить с именем, полученным из предъявленного сертификата. В случае совпадения имен принимается решение об успешном прохождении аутентификации клиента. Иначе сервер авторизации должен ответить кодом ошибки «invalid_client».

4. Механизм аутентификации с использованием протокола TLS с взаимной аутентификацией и самоподписанного сертификата клиента (self_signed_tls_client_auth).

4.1. MTLS-аутентификация клиента с использованием самоподписанного сертификата (подписан ключом подписи клиента, соответствующим ключу проверки подписи в составе сертификата) требует выполнения следующих действий:

1)В ходе динамической регистрации клиент должен зарегистрировать механизм аутентификации «self_signed_tls_client_auth»;

2)В ходе динамической регистрации клиент должен зарегистрировать свои самоподписанные сертификаты формата X.509, либо непосредственно публикуя их в качестве значения параметра <jwks> в формате JWK Set (см. Приложение 5, пункт 3.3), либо указывая в качестве значения параметра <jwks_uri> ссылку на доверенный источник, содержащий его сертификаты в формате JWK Set;

3)Во время установления TLS-соединения сервер авторизации должен запросить сертификат клиента, проверить его валидность, а также выполнить проверку владения клиентом закрытым ключом, соответствующим открытому ключу, представленному в сертификате. В отличие от метода PKI (см. Приложение 4, пункт 3) цепочка сертификатов клиента в этом случае не проверяется сервером авторизации;

4)Клиент успешно аутентифицируется, если сертификат, который он представил во время установления TLS-соединения, соответствует одному из сертификатов, зарегистрированных для этого клиента (см. перечисление в пункте 2) данного списка).

4.2. Метод самоподписанного сертификата позволяет использовать MTLS для аутентификации клиентов без необходимости поддерживать PKI. При использовании вместе с <jwks_uri> клиента он также позволяет клиенту обновлять свои сертификаты X.509 без необходимости изменять свои аутентификационные данные.

Приложение 5
к документу «Межбанковская система
обмена информацией
по открытым программным
интерфейсам (Open API)
Стандарт информационной безопасности
Профили безопасности Open API»
(информационное)

Сведения о специализированных JSON структурах: JWS, JWE, JWK, JWT

1. Цифровая подпись в формате JSON

1.1. JWS (JSON Web Signature) – структура данных в формате JSON, представляющая сообщение с цифровой подписью или кодом аутентификации сообщений.

1.2. JWS состоит из трех частей:

1)<JOSE Header>: JOSE заголовок – JSON-объект, содержащий параметры, описывающие криптографические операции и их параметры (см. Приложение 5, пункт 1.4);

2)<JWS Payload>: функциональное содержимое JWS (полезная нагрузка) – последовательность октетов, подлежащих защите, другими словами – сообщение; функциональное содержимое может состоять из произвольной последовательности октетов;

3)<JWS Signature>: цифровая подпись или код аутентификации сообщений, вычисленные на основе защищенного заголовка JWS и функционального содержимого JWS.

Заголовок <JOSE Header> JWS состоит из двух частей:

1)<JWS Protected Header>: защищенный заголовок JWS – JSON-объект, содержащий параметры заголовка, целостность которых защищена цифровой подписью JWS или операцией MAC;

2)<JWS Unprotected Header>: незащищенный заголовок JWS – JSON-объект, который содержит параметры заголовка, целостность которых не обеспечивается.

1.3. Для передачи структуры JWS может использоваться один из двух типов сериализации (преобразования):

1)компактная сериализация – представление JWS в виде компактной URL-строки;

2)JSON-сериализация – представление JWS в виде объекта JSON.

При использовании компактной сериализации JWS представляется как строка:

$$BASE64URL(UTF8(< JWS\ Protected\ Header >)) \parallel '.' \parallel BASE64URL(< JWS\ Payload >) \parallel '.' \parallel BASE64URL(< JWS\ Signature >)$$

Порядок элементов строки строго определен.

В JSON-сериализации JWS представляется в виде JSON-объекта, содержащего значения следующих параметров:

- <protected>: со значением BASE64URL(UTF8(<JWS Protected Header>)), если значение параметра <JWS Protected Header> не пустое; иначе значение параметра <protected> должно отсутствовать;
- <header>: со значением <JWS Unprotected Header>; этот параметр должен присутствовать, если значение <JWS Unprotected Header> не пустое, иначе он должен отсутствовать;
- <payload>: (обязательный) со значением BASE64URL(<JWS Payload>);
- <signature>: (обязательный) со значением BASE64URL(<JWS Signature>).

1.4. Имена параметров заголовка <JOSE Header> должны быть уникальными; синтаксические анализаторы JWS должны либо отклонить JWS с повторяющимися именами параметров заголовка, либо использовать анализатор JSON, который возвращает только лексически последнее повторяющееся имя элемента.

<JOSE Header> включает следующие параметры:

- <alg>: (обязательный) идентификатор криптографического алгоритма цифровой подписи или MAC, используемого для защиты JWS;
- <jku>: (опциональный) URI, который указывает на ресурс, где находится ключ проверки подписи в представлении JWK (см. Приложение 5, пункт 3), который используется для проверки цифровой подписи JWS;
- <jwk>: (опциональный) ключ проверки подписи в представлении JWK (см. Приложение 5, пункт 3), который используется для проверки цифровой подписи JWS;
- <kid>: (опциональный) идентификатор ключа, который используется для защиты JWS;
- <x5u>: (опциональный) URI, который ссылается на ресурс сертификата формата X.509 или цепочки сертификатов ключа проверки цифровой подписи JWS; данный ресурс должен содержать представление сертификата или цепочки сертификатов в соответствии RFC 5280 (см. раздел «Библиография», пункт [17]) в PEM кодировании; сертификат ключа проверки цифровой подписи JWS должен быть первым сертификатом; в цепочке сертификатов каждый последующий сертификат должен использоваться для сертификации предыдущего; получатель должен проверить цепочку сертификатов в соответствии с RFC 5280 (см. раздел «Библиография», пункт [17]), считать сертификат или цепочку сертификатов и подпись JWS недействительными в случае отрицательного результата проверки; для получения ресурса должен использоваться HTTP запрос GET и протокол TLS;
- <x5c>: (опциональный) сертификат формата X.509 или цепочка сертификатов проверки цифровой подписи JWS; сертификат или цепочка сертификатов представлены в виде JSON-массива строк значений сертификата; каждая строка массива содержит кодировку Base64 (раздел 4 RFC 4648 (см. раздел

«Библиография», пункт [9])) значения DER-представления сертификата формата X.509; сертификат ключа проверки цифровой подписи JWS должен быть первым сертификатом; в цепочке сертификатов каждый последующий сертификат должен использоваться для сертификации предыдущего; получатель должен проверить цепочку сертификатов в соответствии с RFC 5280 (см. раздел «Библиография», пункт [17]), считать сертификат или цепочку сертификатов и подпись JWS недействительными в случае отрицательного результата проверки;

- <typ> (Type): (опциональный) тип сериализации (преобразования) («JOSE» – компактная сериализация, «JOSE+-JSON» – JSON-сериализация);

- <cty> (Content Type): (опциональный) тип (media type) функционального содержимого JWS; используется, если в приложении несколько типов объектов могут быть представлены в качестве содержимого <JWS Payload>;

- <crit>: (опциональный) JSON-массив имен критичных параметров заголовка; если какой-то из параметров, чье имя внесено в этот массив, не поддерживается или не понято получателем, JWS считается недействительной; если параметр <crit> присутствует, он должен быть обработан получателем.

1.5. Значение цифровой подписи или MAC <JWS Signature> вычисляется с помощью алгоритма, указанного значением параметра алгоритма <alg>, используя на входе алгоритма подписи (MAC) ключ, идентифицируемый параметрами <jku>, <jwk>, <kid>, <x5u>, <x5c>, и следующую подписываемую последовательность октетов:

$$ASCII(BASE64URL(UTF8((< JWS\ Protected\ Header >)) \parallel '.' \parallel BASE64URL(< JWS\ Payload >))$$

Проверка подписи JWS выполняется с помощью преобразования проверки цифровой подписи. Если проверка формата JWS или проверка подписи дала отрицательный результат, JWS отвергается как не валидная.

Примечания

1 Общие сведения о структуре JWS и алгоритмах работы с ней приведены в RFC 7515 (см. раздел «Библиография», пункт [20]).

2 Выбор используемых криптографических алгоритмов производится на этапе разработки сервера авторизации и клиента.

2. JSON-структура шифрованного текста

2.1. JWE (JSON Web Encryption) – структура данных в формате JSON, представляющая зашифрованное и защищенное от модификации сообщение.

Структура JWE:

1)<JOSE Header>: JOSE заголовок – JSON-объект, содержащий параметры, описывающие криптографические операции и их параметры (см. Приложение 5, подпункт 1.4);

2)<JWE Encrypted Key>: зашифрованный ключ защиты содержимого JWE для

каждого получателя;

3)<JWE Initialization Vector>: вектор инициализации шифрования. Для алгоритмов, не использующих вектор инициализации, допускается пустое значение параметра;

4)<JWE AAD>: дополнительные аутентифицируемые, не шифруемые данные JWE;

5)<JWE Ciphertext>: зашифрованный текст JWE;

6)<JWE Authentication Tag>: имитовставка JWE.

В состав <JOSE Header> входят следующие компоненты:

1)<JWE Protected Header>: защищенный заголовок JWE, целостность значения которого защищается с помощью JWE;

2)<JWE Shared Unprotected Header>: общий незащищенный заголовок JWE;

3)<JWE Per-Recipient Unprotected Header>: незащищенный заголовок для каждого получателя JWE (целостность заголовка не обеспечивается).

Для обеспечения конфиденциальности и целостности открытого текста, а также целостности защищенного заголовка и <JWE AAD> используется алгоритм аутентифицированного шифрования с присоединенными данными.

2.2. Как и JWS, структура JWE использует компактную и JSON-сериализацию. В компактной сериализации JWE представляется следующим образом:

$$BASE64URL(UTF8(< JWE\ Protected\ Header >)) \parallel '.' \parallel BASE64URL(< JWE\ Encrypted\ Key >) \parallel '.' \parallel BASE64URL(< JWE\ Initialization\ Vector >) \parallel '.' \parallel BASE64URL(< JWE\ Ciphertext >) \parallel '.' \parallel BASE64URL(< JWE\ Authentication\ Tag >)$$

Компактная сериализация не поддерживает параметры <JWE Shared Unprotected Header>, <JWE Per-Recipient Unprotected Header> и <JWE AAD>. Кроме того, она предполагает одного получателя сообщения.

В случае JSON-сериализации JWE представлен в виде JSON-объекта, содержащего следующие параметры:

1)<protected>: если заголовок <JWE Protected Header> не пустой, этот параметр должен присутствовать со значением BASE64URL(UTF8(<JWE Protected Header>)); иначе этот параметр должен быть опущен;

2)<unprotected>: если заголовок <JWE Shared Unprotected Header> не пустой, этот параметр должен присутствовать со значением <JWE Shared Unprotected Header>; иначе этот параметр должен быть опущен; как правило, это JSON-объект;

3)<recipients>: (обязательный) массив JSON-объектов, в адрес каждого получателя JWE; структура этого объекта:

- <header>: если заголовок <JWE Per-Recipient Unprotected Header> не пустой, этот параметр должен присутствовать со значением <JWE Per-Recipient Unprotected Header>; иначе этот параметр должен быть опущен; как правило, это

JSON-объект;

- <encrypted_key>: если значение <JWE Encrypted Key> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE Encrypted Key>); иначе этот параметр должен быть опущен;

4)<iv>: если значение <JWE Initialization Vector> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE Initialization Vector>); иначе этот параметр должен быть опущен;

5)<aad>: если значение <JWE AAD> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE AAD>); иначе этот параметр должен быть опущен;

6)<ciphertext>: (обязательный) зашифрованный текст (шифротекст) со значением BASE64URL(<JWE Ciphertext>);

7) <tag>: если значение <JWE Authentication Tag> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE Authentication Tag>); иначе этот параметр должен быть опущен.

2.3. JOSE заголовок включает в себя следующие параметры:

- <enc>: (обязательный) идентификатор алгоритма шифрования, используемого для выполнения аутентифицированного шифрования открытого текста; этот параметр должен быть обработан получателем;

- <zip>: (опциональный) идентификатор алгоритма сжатия, который применяется к незашифрованному тексту перед шифрованием; определено единственное значение: «DEF» – алгоритм сжатия DEFLATE; если значение этого параметра отсутствует, выполняется шифрование открытого текста без сжатия;

- <alg>, <jku>, <jwk>, <kid>, <x5u>, <x5c>, <typ>, <cty>, <crit>: как в JWS (см. Приложение 5, пункт 1); при этом параметры <jku>, <jwk>, <kid>, <x5u>, <x5c> определяют открытый ключ, который был использован при формировании ключа шифрования, получатель может воспользоваться этой информацией, чтобы выбрать соответствующий закрытый ключ.

2.4. Шифрование выполняется следующим образом:

- в соответствии с идентификатором алгоритма шифрования вычислить ключ шифрования содержимого - CEK;

- если используется алгоритм с шифрованием ключа, зашифровать ключ CEK, присвоив значению <JWE Encrypted Key> результат шифрования; если алгоритм шифрования не предполагает шифрования ключа, присвоить <JWE Encrypted Key> пустое значение;

- если требуется для алгоритма шифрования, сгенерировать значение случайного вектора инициализации <JWE Initialization Vector> необходимой длины; иначе значение <JWE Initialization Vector> должно быть пустое;

- если указано значение параметра «zip», присвоить M – последовательность октетов, представляющих сжатый открытый текст; иначе присвоить M – последовательность октетов, представляющих открытый текст;

- зашифровать M с помощью алгоритма, который идентифицирован

параметрами <alg> и <enc>, с использованием значений ключа CEK, вектора инициализации <JWE Initialization Vector> и <JWE AAD>; результат – зашифрованный текст (шифротекст) <JWE Ciphertext> и имитовставка <JWE Authentication Tag>.

Расшифрование выполняет получатель JWE обратным преобразованием. Если сообщение зашифровано в адрес нескольких получателей, обрабатывается только один из подходящих заголовков, адресованный получателю. Если проверка целостности дала отрицательный результат, зашифрованное сообщение должно быть отвергнуто.

Примечания

1 Общие сведения о структуре JWE и алгоритмах работы с ней приведены в RFC 7516 (см. раздел «Библиография», пункт [21]).

2 Выбор используемых криптографических алгоритмов производится на этапе разработки сервера авторизации и клиента.

3. JSON структура ключа

3.1. JSON Web Key (JWK) – это структура данных в формате JSON, представляющая криптографический ключ. Параметры этой структуры представляют свойства ключа, в том числе и значение ключа.

Ниже приведена типовая структура ключа, независимо от его типа. Кроме того, в состав структуры JWK могут входить параметры, специфические для конкретных криптографических алгоритмов, использующих ключ.

3.2. В состав JWK входят следующие параметры:

1)<kty>: (обязательный) тип ключа; определены следующие значения:

- «EC»: ключ для криптографического алгоритма на базе эллиптической кривой;
- «oct»: последовательность октетов, используемая для представления симметричного ключа;

2)<use>: (опциональный) предполагаемое использование открытого ключа; допускаются следующие значения:

- «sig» – подпись,
- «enc» – шифрование;

3)<key_ops>: (опциональный) идентифицирует операции, для которых предполагается использовать ключ; определены следующие значения этого параметра:

- «sign» – вычисление цифровой подписи или кода аутентификации сообщений;
- «verify» – проверка цифровой подписи или кода аутентификации сообщений;
- «encrypt» – шифрование контента;
- «decrypt» – расшифрование контента и проверка расшифрованного, если

возможно;

- «wrapKey» – шифрование ключа;
- «unwrapKey» – расшифрование ключа и проверка расшифрованного, если возможно;
- «deriveKey» – вычисление производного ключа;
- «deriveBits» – вычисление производных битов не для использования в качестве ключа;

4)<alg>: (опциональный) идентифицирует криптографический алгоритм, в котором предполагается использование ключа;

5)<kid>: (опциональный) идентификатор ключа; используется для того, чтобы выбрать нужный ключ в документе JWK Set;

6)<x5u>, <x5c>, <x5t>, <x5t#S256>: (опциональные) как в JWS (см. Приложение 5, пункт 1).

Примечание - В зависимости от типа ключа <kty> в состав структуры JWK могут включаться также другие специфические параметры.

3.3. JSON-структура набора ключей JWK Set состоит из одного параметра <keys>, который является JSON-массивом ключей в формате JWK.

3.4. Документ Key Set, на который указывает значение параметра <jwks_uri> документа OpenID Connect Discovery сервера авторизации (см. Приложение 3, пункт 3), содержит ключ(и) проверки подписи сервера авторизации. Он также может содержать открытые ключи, для которых вырабатывается общий ключ шифрования запросов к серверу авторизации. Набор Key Set, на который указывает значение параметра <jwks_uri> метаданных клиента (см. Приложение 3, пункт 5), содержит ключ(и) проверки подписи клиента. Он также может содержать открытые ключи, для которых вырабатывается общий ключ шифрования запросов к клиенту.

Если в наборе Key Set присутствуют как ключи проверки подписи, так и ключи шифрования, в структуре JWK каждого ключа должно быть указано значение параметра <use>. Не допускается использование одного и того же ключа как для целей подписи, так и для шифрования. Параметр JWK <x5c> может использоваться для предоставления ключей в формате сертификата X.509. В этом случае значение открытого ключа также должно присутствовать в составе JWK и совпадать со значением в сертификате.

Примечания

1 Общие требования к структурам JWK и JWK Set приведены в RFC 7517 (см. раздел «Библиография», пункт [22]).

2 Структуры JWK и JWK Set определяются на этапе разработки сервера авторизации и клиента.

4. Структура JSON веб-токена

4.1. JSON веб-токен (JWT) – токен доступа, основанный на формате JSON. Концептуально JWT представляет собой набор параметров (claims) в формате JSON-объекта, закодированных в виде структуры JWS или JWE с цифровой подписью, кодом аутентификации и/или шифрованием.

Наименование параметров JWT:

- <iss>: (опциональный) идентификатор эмитента JWT;
- <sub>: (опциональный) идентификатор субъекта JWT;
- <aud>: (опциональный) перечень идентификаторов получателей, которым предназначен JWT;
- <exp>: (опциональный) время завершения срока действия JWT;
- <nbf>: (опциональный) время, до которого JWT не должен приниматься к обработке;
- <iat>: (опциональный) время выпуска JWT;
- <jti>: (опциональный) идентификатор JWT.

Эти параметры JWT помещаются в компоненте JWS Payload или в зашифрованном виде – в содержимое JWE Ciphertext.

4.2. JWT может быть либо подписан, либо подписан и зашифрован. Если JWT подписан и зашифрован, JSON-документ будет подписан, затем зашифрован, а результатом будет структура вложенного JWT – Nested JWT.

4.3. В JOSE-заголовке JWT независимо от типа (JWS или JWE) используются два параметра: <typ> (рекомендуется использовать строку «JWT») и <cty> (при наличии вложенной подписи или шифрования его значением должна быть строка «JWT»).

Примечание - Дополнительные сведения по структурам JWT и Nested JWT приведены в RFC 7519 (см. раздел «Библиография», пункт [23]).

Библиография

[1] Financial-grade API – Part 1: Read Only API Security Profile (Implementer's Draft). OpenID Foundation, Financial-grade API (FAPI) WG, 2019. https://openid.net/specs/openid-financial-api-part-1-1_0-final.html.

[2] Financial-grade API – JWT Secured Authorization Response Mode for OAuth 2.0 (JARM) (Implementer's Draft). OpenID Foundation, Financial-grade API (FAPI) WG, 2019. https://openid.net/specs/openid-financial-api-jarm-ID1.html.

[3] Sakimura N., Bradley J., Jones M., de Medeiros B., Mortimore C. OpenID Connect Core 1.0 incorporating errata set 1. November 8, 2014. https://openid.net/specs/openid-connect-core-1_0.html.

[4] СТ РК ИСО/МЭК 10181-4-2008 Информационная технология. Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 4. Основы неотказуемости

[5] СТ РК ИСО/МЭК 10181-6-2008 Информационная технология. Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 6. Основы целостности

[6] СТ РК ИСО/МЭК 10181-2-2008 Информационная технология. Взаимодействие открытых систем. Основы безопасности для открытых систем. Часть 2. Основы аутентификации

[7] Hardt D. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012. https://tools.ietf.org/html/rfc6749.

[8] ГОСТ Р ИСО/МЭК 8825-1-2003 «Информационная технология. Правила кодирования АСН.1. Часть 1. Спецификация базовых (BER), канонических (CER) и отличительных (DER) правил кодирования».

[9] The Base16, Base32, and Base64 Data Encodings. RFC 4648. https://tools.ietf.org/html/rfc4648.

[10] A UniveRSAlly Unique IDentifier (UUID) URN Namespace. RFC 4122. https://tools.ietf.org/html/rfc4122.

[11] Sakimura N., Bradley J., Jones M., Jay E. OpenID Connect Discovery 1.0 incorporating errata set 1. November 8, 2014. https://openid.net/specs/openid-connect-discovery-1_0.html.

[12] OAuth 2.0 Authorization Server Metadata. RFC 8414. https://tools.ietf.org/html/rfc8414.

[13] Sakimura N., Bradley J., Jones M. OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1. November 8, 2014. https://openid.net/specs/openid-connect-registration-1_0.html.

[14] OAuth 2.0 Dynamic Client Registration Protocol. RFC 7591. https://tools.ietf.org/html/RFC7591.

[15] Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7521. May 2015. https://tools.ietf.org/html/rfc7521.
[16] JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7523. May 2015. https://tools.ietf.org/html/rfc7523.
[17] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. May 2008. https://tools.ietf.org/html/rfc5280.
[18] HTTP Authentication: Basic and Digest Access Authentication. RFC 2617. June 1999. https://tools.ietf.org/html/rfc2617.
[19] Jones M., Hardt D. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, October 2012. https://tools.ietf.org/html/rfc6750.
[20] JSON Web Signature (JWS). RFC 7515. https://tools.ietf.org/html/rfc7515.
[21] JSON Web Encryption (JWE). RFC 7516. https://tools.ietf.org/html/rfc7516.
[22] JSON Web Key (JWK). RFC 7517. https://tools.ietf.org/html/rfc7517.
[23] JSON Web Token (JWT). RFC 7519. https://tools.ietf.org/html/rfc7519.
[24] OAuth 2.0 Form Post Response Mode: https://openid.net/specs/oauth-v2-form-post-responsemode-1_0.html
[25] OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens. RFC 8705. https://www.rfc-editor.org/rfc/rfc8705.html
[26] OAuth 2.0 Token Introspection. RFC 7662
[27] OpenID Connect Session Management 1.0: https://openid.net/specs/openid-connect-session-1_0.html
[28] СТ РК ИСО/МЭК 15816-2009 Информационная технология. Методы защиты. Объекты информационной защиты по управлению доступом.
[29] СТ РК ISO/IEC 9797-1-2017 Информационные технологии. Методы и средства обеспечения безопасности. Коды аутентификации сообщений (MAC). Часть 1 Механизмы, использующие блочный шифр