Банк России

СТАНДАРТ БАНКА РОССИИ

СТО БР ФАПИ.СЕК-1.6-2020

БЕЗОПАСНОСТЬ ФИНАНСОВЫХ (БАНКОВСКИХ) ОПЕРАЦИЙ

ПРИКЛАДНЫЕ ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ФИНАНСОВЫХ СЕРВИСОВ НА ОСНОВЕ ПРОТОКОЛА OPENID

ТРЕБОВАНИЯ

МОСКВА
2020

ПРЕДИСЛОВИЕ

1. РАЗРАБОТАН Ассоциацией развития финансовых технологий (Ассоциация ФинТех) и открытым акционерным обществом «Информационные технологии и коммуникационные системы» (ОАО «ИнфоТеКС») при участии Банка России.

2. ПРИНЯТ И ВВЕДЕН В ДЕЙСТВИЕ приказом Банка России
от «09» октября 2020 года № ОД-1650

3. ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования – на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.gost.ru).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящий стандарт разработан на основе спецификаций организации OpenID Foundation (OIDF), которая продвигает и поддерживает сообщество и технологии OpenID (OpenID Connect Core (OIDC), OpenID Connect Discovery (OIDD) и другие), и определяет порядок использования модели прикладных программных интерфейсов (application programming interface, API) со структурированными данными и модель токена для повышения безопасности финансовых технологий.

СТАНДАРТ БАНКА РОССИИ

БЕЗОПАСНОСТЬ ФИНАНСОВЫХ (БАНКОВСКИХ) ОПЕРАЦИЙ

ПРИКЛАДНЫЕ ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ФИНАНСОВЫХ СЕРВИСОВ НА ОСНОВЕ ПРОТОКОЛА OPENID

ТРЕБОВАНИЯ

Financial-grade API Security. Requirements

Дата введения – 2020-XX-XX

1. ОБЛАСТЬ ПРИМЕНЕНИЯ

Настоящий стандарт рекомендован к использованию при создании и оценке соответствия программных средств, предназначенных для безопасного обмена финансовыми сообщениями, связанными:

– с получением информации о банковском счете;
– с переводом денежных средств в валюте Российской Федерации.

Настоящий стандарт предназначен для организаций:

– участников получения информации о банковском счете (банки и их клиенты, а также сторонние поставщики[1]);
– участников перевода денежных средств (банки и их клиенты, а также сторонние поставщики[2]);
– разработчиков информационного и программного обеспечения, информационных систем.

Положения настоящего стандарта применяются на добровольной основе, если только в отношении конкретных положений обязательность их применения не установлена нормативными актами Банка России или условиями договоров.

Положения настоящего стандарта применяются совместно с документом Технического комитета ТК26 «Криптографическая защита информации» «Использование российских криптографических алгоритмов в протоколах OpenID Connect».

2. НОРМАТИВНЫЕ ССЫЛКИ

В настоящем стандарте использованы ссылки на следующие документы:

– ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования»;
– Р 1323565.1.020-2018 «Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.2)»;
– ГОСТ Р ИСО/МЭК 8825-1-2003 «Информационная технология. Правила кодирования АСН.1. Часть 1. Специфика базовых (BER), канонических (CER) и отличительных (DER) правил кодирования».

Примечание. При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов и классификаторов в информационной системе общего пользования – на официальном сайте федерального органа исполнительной власти в сфере стандартизации в сети Интернет или по ежегодно издаваемому информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по опубликованным в текущем году выпускам ежемесячно издаваемого информационного указателя «Национальные стандарты». Если заменен ссылочный до-

---

[1] *Получают доступ к информации с согласия владельца счета.*

[2] *Получают доступ к счету с согласия владельца счета.*

кумент, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого документа с учетом всех внесенных в данную версию изменений. Если заменен ссылочный документ, на который дана датированная ссылка, то рекомендуется использовать версию этого документа с указанным выше годом утверждения (принятия). Если после утверждения настоящего стандарта в ссылочный документ, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный документ отменен без замены, то положение, в котором дана ссылка на него, рекомендуется применять в части, не затрагивающей эту ссылку.

3. ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

3.1

Атака (attack): попытка уничтожения, раскрытия, изменения, блокирования, кражи, получения несанкционированного доступа к активу или его несанкционированного использования.

[ГОСТ Р ИСО/МЭК 27000–2012, пункт 2.4]

3.2

Аутентификация: действия по проверке подлинности субъекта доступа и/или объекта доступа, а также по проверке принадлежности субъекту доступа и/или объекту доступа предъявленного идентификатора доступа и аутентификационной информации.

Примечание. Аутентификация рассматривается применительно к конкретному субъекту доступа и/или конкретному объекту доступа.

[ГОСТ Р 58833–2020, пункт 3.4]

3.3

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.

[ГОСТ Р 58833–2020, пункт 3.36]

3.4

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.

[ГОСТ Р 58833–2020, пункт 3.10]

3.5

Протокол аутентификации: протокол, позволяющий участникам процесса аутентификации осуществить аутентификацию.

Примечание. Протокол реализует алгоритм (правила), в рамках которого субъект доступа и объект доступа последовательно выполняют определенные действия и обмениваются сообщениями.

[ГОСТ Р 58833–2020, пункт 3.47]

3.6

Идентификация (identification): действия по присвоению субъектам и объектам доступа идентификаторов и (или) по сравнению предъявляемого идентификатора с перечнем присвоенных идентификаторов.

[Р 50.1.053–2005, пункт 3.3.9]

3.7

Авторизация (authorization): проверка, подтверждение и предоставление прав логического доступа при осуществлении субъектами доступа логического доступа.

[ГОСТ Р 57580.1–2017, пункт 3.15]

3.8

Доступ: получение одной стороной информационного взаимодействия возможности использования ресурсов другой стороны.

Примечания

1. В качестве ресурсов стороны информационного взаимодействия, которые может использовать другая сторона информационного взаимодействия, рассматриваются информационные ресурсы, вычислительные ресурсы средств вычислительной техники и ресурсы автоматизированных (информационных) систем, а также средства вычислительной техники и автоматизированные (информационные) системы в целом.

2. Доступ к информации – возможность получения информации и ее использования.

[ГОСТ Р 58833–2020, пункт 3.17]

3.9

[Криптографический] ключ (key): изменяемый параметр в виде последовательности символов, определяющий криптографическое преобразование.

[ГОСТ Р 34.12–2015, пункт 2.1.8]

3.10

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

1) по данному значению функции сложно вычислить исходные данные, отображаемые в это значение;

2) для заданных исходных данных сложно вычислить другие исходные данные, отображаемые в то же значение функции;

3) сложно вычислить какую-либо пару исходных данных, отображаемых в одно и то же значение.

[ГОСТ Р 34.10, пункт 3.1.14]

3.11

Хэш-код (hash-code): строка бит, являющаяся выходным результатом хэш-функции.

[ГОСТ Р 34.10, пункт 3.1.13]

3.12

Код аутентификации [сообщения] (message authentication code): специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности и аутентификации источника данных. [7]

3.13

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.

[Р 50.1.113–2016, пункт 3.1.1]

3.14

[Электронная цифровая] подпись (signature): строка бит, полученная в результате процесса формирования подписи.

[ГОСТ Р 34.10, пункт 3.1.15]

3.15

Ключ подписи (signature key): элемент конфиденциальных данных, специфичный для субъекта и используемый только данным субъектом в процессе формирования цифровой подписи.

[ГОСТ Р 34.10, пункт 3.1.2]

3.16

Ключ проверки подписи (verification key): элемент данных, математически связанный с ключом подписи и используемый проверяющей стороной в процессе проверки цифровой подписи.

[ГОСТ Р 34.10, пункт 3.1.3]

3.17

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

[ГОСТ Р 34.10, пункт 3.1.8]

3.18

Процесс формирования подписи (signature process): процесс, в качестве исходных данных которого используются сообщение, ключ подписи и параметры схемы ЭЦП, а в результате формируется цифровая подпись.

[ГОСТ Р 34.10, пункт 3.1.9]

3.19

Прикладной программный интерфейс (application program interface, API): интерфейс между прикладным программным средством и прикладной платформой, через который обеспечивается доступ ко всем необходимым службам (услугам).

[Р 50.1.041–2002, пункт 3.1.5]

3.20

Клиент (client): приложение, выполняющее запросы защищенных ресурсов от имени владельца ресурса и с его авторизацией. [10]

Примечание. Термин «клиент» не подразумевает каких-либо конкретных характеристик реализации (например, выполняется ли приложение на сервере, настольном компьютере или других устройствах).

3.21

Конфиденциальный клиент (confidential client): клиент, который может обеспечить конфиденциальность своих учетных данных (например, клиент, реализованный на защищенном сервере с ограниченным доступом к учетным данным клиента) или может выполнить безопасную аутентификацию клиента с использованием других средств. [10]

3.22

Публичный клиент (public client): клиент, который не может обеспечить конфиденциальность своих учетных данных (например, клиент, выполняющийся на устройстве, используемом владельцем ресурса, таком как установленное нативное приложение или приложение на основе веб-браузера) и не может выполнить безопасную аутентификацию клиента с помощью других средств. [10]

3.23

Нативное приложение (native application): приложение, которое пользователь устанавливает на свое устройство, в отличие от веб-приложения, которое работает только в контексте браузера. [10]

Примечание. Приложения, реализованные с использованием веб-технологий, но распространяемые как нативные приложения, так называемые «гибридные приложения», считаются эквивалентными нативным приложениям для целей данных требований.

3.24

Агент пользователя (user agent): клиентское приложение, использующее определенный сетевой протокол для доступа к серверу. Термин обычно используется для приложений, таких как браузеры, поисковые роботы, почтовые клиенты.

3.25

Владелец ресурса (resource owner): субъект, способный предоставить доступ к защищенному ресурсу. [10]

3.26

Конечный пользователь (end user): владелец ресурса в случае, если он является человеком. [10]

3.27

Защищенный ресурс (protected resource): ресурс с ограниченным доступом. [10]

3.28

Сервер ресурсов (resource server): сервер, на котором размещены защищенные ресурсы, способный принимать и отвечать на запросы защищенных ресурсов с использованием токенов доступа. [10]

3.29

Сервер авторизации (authorization server): сервер, выдающий клиенту токены доступа после успешной аутентификации владельца ресурса и получения авторизации. [10]

3.30

Разрешение на доступ (authorization grant): свидетельство, представляющее авторизацию владельца ресурса (для доступа к его защищенным ресурсам), которое далее используется клиентом для получения токена доступа. [10]

3.31

Конечная точка (endpoint): адрес ресурса, точка входа интерфейса. [10]

Примечание. В процессе авторизации технология OAuth 2.0 использует две конечные точки сервера авторизации (ресурсы HTTP) – конечную точку авторизации и конечную точку токена, а также конечную точку клиента.

3.32

Конечная точка авторизации (authorization endpoint): конечная точка, используемая клиентом для получения авторизации от владельца ресурса посредством перенаправления агента пользователя. [10]

3.33

Конечная точка токена (token endpoint): конечная точка, используемая клиентом для обмена разрешения на доступ на токен доступа, обычно с аутентификацией клиента. [10]

3.34

Конечная точка клиента (client endpoint): конечная точка, на которую сервер авторизации возвращает ответы клиенту посредством агента пользователя. [10]

3.35

Конечная точка UserInfo (UserInfo endpoint): защищенный ресурс, который при предоставлении клиентом токена доступа возвращает авторизованную информацию о конечном пользователе. [11]

3.36

Параметр, заявленное свойство (claim): часть информации, заявленной о субъекте. Заявленное свойство представлено в виде пары имя/значение, состоящей из имени заявленного свойства (параметра) и значения заявленного свойства (параметра). [12]

3.37

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. [12]

3.38

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. [10]

3.39

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 (токены доступа могут иметь более короткий срок службы и меньше разрешений на доступ, чем разрешено владельцем ресурса). [10]

3.40

Токен идентификации, ID токен (ID Token): JSON веб-токен, который содержит параметры события аутентификации. Может содержать также другие параметры. [12]

3.41

Токен на предъявителя (bearer token): токен доступа с тем свойством, что любая сторона, владеющая токеном (предъявитель), может использовать токен по назначению, не доказывая владение соответствующим криптографическим ключом. [13]

3.42

Объект запроса (request object): токен JWT, который содержит набор параметров запроса в качестве своих заявленных свойств. [11]

3.43

Аутентифицированное шифрование с присоединенными данными (Authenticated Encryption with Associated Data): режим работы блочного шифра, который обеспечивает шифрование и имитозащиту открытого текста. При этом часть открытого текста (дополнительные аутентифицированные данные, AAD) не шифруется.

3.44

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. [14]

3.45

Base64url кодирование: Base64 кодирование строки октетов с использованием набора символов, допускающих использование результата кодирования в качестве имени файла или URL. [14]

4. ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

API (Application Program Interface) – прикладной программный интерфейс

ASCII (American Standard Code for Information Interchange) – стандарт 8-битного кодирования некоторого набора печатных и непечатных символов

ASCII (STRING) – октеты ASCII представления последовательности ASCII символов STRING

BASE64URL (OCTETS) – Base64url кодирование строки октетов OCTETS

CSRF (Cross Site Request Forgery) – межсайтовая подделка запроса

DER (Distinguished Encoding Rules) – отличительные правила кодирования структур ASN.1

FAPI (Financial-grade API) – API обеспечения безопасности финансовых сервисов

HTTP (Hypertext Transfer Protocol) – протокол передачи гипертекстовых сообщений

HTTPS (Hypertext Transfer Protocol Secure) – протокол защищенной передачи гипертекстовых сообщений

HMAC (Hash-based Message Authentication Code) – код аутентификации сообщения на основе хэш-функции

IANA (Internet Assigned Numbers Authority) – Агентство по выделению имен и уникальных параметров протоколов Internet (Администрация адресного пространства Интернет)

ID (identifier) – идентификатор, идентификационный номер

IESG (Internet Engineering Steering Group) **–** группа технического управления Internet **(**исполнительный комитет IETF)

JARM (JWT Secured Authorization Response Mode for OAuth 2.0) – защищенный с использованием токена JWT режим ответа на запрос авторизации в OAuth 2.0

JOSE (JavaScript Object Signing and Encryption) – технология подписи и шифрования объектов JavaScript

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript (нотация объектов JavaScript)

JWA (JSON Web Algorithmes) – набор криптографических алгоритмов и идентификаторов для использования со спецификациями JWS и JWE

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

JWK (JSON Web Key) – структура данных в формате JSON, представляющая криптографический ключ

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

JWT (JSON Web Token) – токен доступа, основанный на формате JSON

MAC (Message Authentication Code) – код аутентификации сообщения

MTLS (Mutual TLS) – процесс, в соответствии с которым при согласовании сеанса TLS выполняется взаимная аутентификация сервера TLS и клиента, а также подтверждение владения соответствующими ключами

Nested JWT – JWT, в котором используются вложенные подпись и/или шифрование. В Nested JWT вложенная структура JWT используется в качестве полезной нагрузки или значения открытого текста вмещающей ее структуры JWS или JWE соответственно

OAuth – открытый протокол (схема) авторизации

OIDC (OpenID Connect Core) – 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

OIDF (OpenID Foundation) – некоммерческая организация, созданная для управления авторскими правами, товарными знаками, маркетинговыми компаниями и другой деятельностью, связанной с сообществом OpenID

OpenID – открытый стандарт децентрализованной системы аутентификации

PKI (Public Key Infrastructure) – инфраструктура открытых ключей

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной гипермедиасистемы. Для веб-служб, построенных с учетом REST, применяют термин «RESTful»

RFC (Request for Comments) – предложения для обсуждения; серия нормативных документов, стандартизирующих протоколы Internet

TLS (Transport Layer Security) – протокол защиты транспортного уровня

URI (Uniform Resource Identifier) – унифицированный идентификатор ресурса

URL (Uniform Resource Locator) – унифицированный адрес ресурса (единый указатель ресурса)

URN (Uniform Resource Name) – унифицированное имя ресурса

UTF-8 – стандарт кодирования символов, позволяющий компактно хранить и передавать символы Unicode, используя переменное количество байтов (от 1 до 4), и обеспечивающий полную обратную совместимость с кодировкой ASCII

UTF8 (STRING) – октеты UTF-8 представления последовательности Unicode символов STRING

A||B – операция конкатенации символьных строк A и B

<name> – параметр (claim) некоторого субъекта с именем name

5. ОБЩИЕ ПОЛОЖЕНИЯ

5.1. СТРУКТУРА СТАНДАРТА

5.1.1. Настоящий стандарт (далее – ФАПИ.СЕК) предоставляет требования и рекомендации для обеспечения безопасного доступа к финансовым данным в финансовых сервисах реального времени с использованием модели обмена данными REST/JSON, защищенной технологией OAuth, включая профилирующий ее протокол OpenID Connect.

ФАПИ.СЕК состоит из следующих частей:

– профиль безопасности OpenID API для доступа к сервисам в режиме только для чтения;

– профиль безопасности OpenID API для доступа к сервисам в режиме чтения и записи;

– защищенный с использованием JWT режим ответа на запрос авторизации OAuth 2.0 (JARM).

В разделе 5 настоящего стандарта приведены нормативные требования (см. подраздел 5.2), а также общие сведения о протоколах авторизации OAuth 2.0 и аутентификации OpenID Connect (см. подразделы 5.3–5.8).

Первая часть ФАПИ.СЕК (раздел 6) основана на документе [15], выпущенном OpenID Foundation. В этой части определяются требования к системным (безопасность на уровне транспорта и алгоритмов преобразования данных) и прикладным параметрам протокола OpenID Connect, выполнение которых является обязательным для обеспечения безопасного доступа к конфиденциальной финансовой информации/данным в режиме только для чтения.

Вторая часть (раздел 7), источник [16], представляет профиль безопасности более высокого уровня – профиль API для чтения и записи финансовых данных и других аналогичных ситуаций с повышенным риском несанкционированной модификации данных. В ней определяются меры защиты от таких атак, как фальсификация запроса авторизации, фальсификация ответа на запрос авторизации, включая внедрение кода, внедрение состояния и фишинг запроса токена.

Третья часть настоящего стандарта – JARM (раздел 8) – основана на документе [17] и определяет использующий токены доступа JWT-формат представления ответов на запросы авторизации. Клиентам дается возможность запрашивать передачу параметров ответа на запрос авторизации вместе с допол-

нительными данными в JWT-формате. Этот механизм повышает безопасность стандартного ответа на запрос авторизации, поскольку добавлены поддержка подписи и шифрования, аутентификация отправителя, ограничение аудитории, а также защита от повторного воспроизведения, утечки учетных данных и атак путем подмены/перепутывания. Он может быть объединен с любым типом ответа.

5.2. НОРМАТИВНЫЕ ТРЕБОВАНИЯ

5.2.1. Вовлеченные стороны должны следовать требованиям обеспечения безопасности персональных данных, определенным нормативными правовыми актами Российской Федерации, в частности:

- Федеральным законом от 27 июля 2006 года № 152-ФЗ «О персональных данных» [18];
- постановлением Правительства Российской Федерации от 1 ноября 2012 г. № 1119 «Об утверждении требований к защите персональных данных при их обработке в информационных системах персональных данных» [19];
- приказом Федеральной службы безопасности (ФСБ России) от 10 июля 2014 г. № 378 «Об утверждении Состава и содержания организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных с использованием средств криптографической защиты информации, необходимых для выполнения установленных Правительством Российской Федерации требований к защите персональных данных для каждого из уровней защищенности» [20];
- приказом Федеральной службы по техническому и экспортному контролю (ФСТЭК России) от 18 февраля 2013 г. № 21 «Об утверждении Состава и содержания организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных» [21].

5.2.2. Прикладное программное обеспечение, реализующее требования настоящего стандарта, должно отвечать требованиям Положений Банка России от 9 июня 2012 года № 382-П [22], от 17 апреля 2019 года № 683-П [23], от 17 апреля 2019 года № 684-П [24] в части, предъявляемой к анализу уязвимостей в прикладном ПО как на этапе его разработки, так и при его использовании в составе конечных целевых систем.

Разработка программного обеспечения, реализующего требования настоящего стандарта, должна вестись в соответствии с ГОСТ Р 56939. В том числе должен проводиться регулярный поиск информации, связанной с уязвимостями программ, в общедоступных источниках, в том числе с использованием банка данных угроз безопасности информации ФСТЭК России.

5.2.3. Оценка соответствия и сертификация реализаций, отвечающих требованиям настоящего стандарта, производится в соответствии с законодательством Российской Федерации.

5.3. ТЕХНОЛОГИЯ АВТОРИЗАЦИИ OAUTH 2.0

5.3.1 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 [10] и RFC 6750 [13].

5.3.2. Схематично сценарий протокола OAuth 2.0 представлен на рисунке 1. Он описывает взаимодействие между упомянутыми выше четырьмя сторонамии включает следующие шаги.

1 Клиент запрашивает авторизацию у владельца ресурса. Запрос авторизации может быть направлен владельцу ресурса напрямую, как показано на рисунке 1, 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 до-

ступ 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.

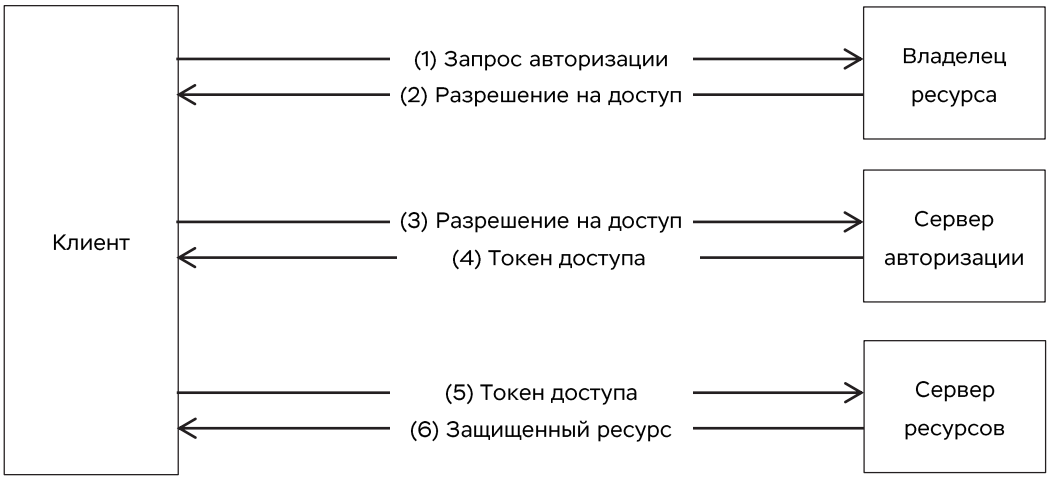
3 Клиент запрашивает токен доступа посредством аутентификации на сервере авторизации и предоставления разрешения на доступ.

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.

СЦЕНАРИЙ ПРОТОКОЛА OAUTH 2.0 *Рис. 1*



5.3.3. 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.

5.3.4. 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.

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

5.4. OPENID CONNECT

5.4.1. Протокол OpenID Connect

5.4.1.1. OIDC – 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. Это этапы (1) – (4) протокола OAuth 2.0 (см. рисунок 1).

5.4.1.2. Схематично протокол OpenID Connect выглядит следующим образом (см. рисунок 2):

1 Клиент отправляет серверу авторизации запрос аутентификации.

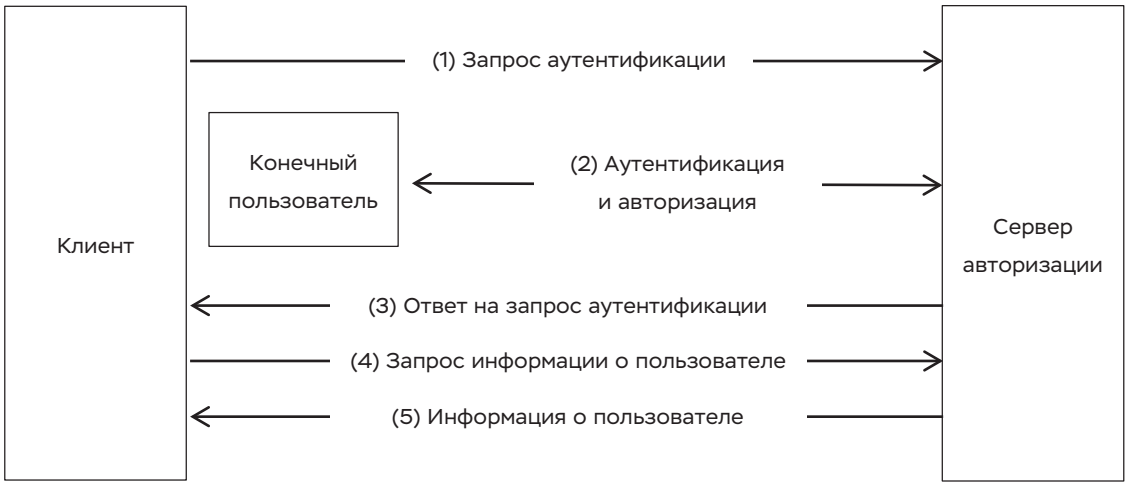
2 Сервер авторизации аутентифицирует конечного пользователя и получает согласие пользователя на доступ клиента к запрошенному ресурсу.

3 Сервер авторизации отвечает клиенту ID 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.

СЦЕНАРИЙ АУТЕНТИФИКАЦИИ OPENID CONNECT *Рис. 2*



5.4.1.3. Сервер авторизации OpenID Connect поддерживает три сценария аутентификации, реализующие этот сценарий: с генерацией кода авторизации (Authorization Code Flow), неявный сценарий (Implicit Flow), гибридный сценарий (Hybrid Flow). Настоящий стандарт не предполагает использование неявного сценария.

5.4.1.4. Передача сообщений между клиентом и сервером авторизации (на конечных точках авторизации, токена и UserInfo) должна производиться с использованием протокола TLS.

Примечание. Дополнительные сведения об OIDC приведены в документе [11].

5.4.2. Аутентификация с генерацией кода авторизации

5.4.2.1. 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 ID токен и токен доступа, используя код авторизации на конечной точке токена.
7 Клиент получает ответ, содержащий ID токен и токен доступа.
8 Клиент проверяет ID токен (см. подпункт 5.4.2.17) и получает идентификатор конечного пользователя.

5.4.2.2. В состав *запроса аутентификации* клиент может включать следующие параметры:

– `<scope>`: (обязательный) область действия; определяет свойства защищаемых данных конечного пользователя, к которым запрошен доступ; в случае использования протокола OpenID Connect параметр <scope> должен содержать строку “`openid`”;
– `<response_type>`: (обязательный) тип ответа и сценарий протокола авторизации; в данном сценарии используется следующее значение:
  – “`code`”: возвращает код авторизации; используется в сценарии аутентификации с генерацией кода авторизации;
– `<client_id>`: (обязательный) идентификатор клиента OAuth 2.0, полученный при регистрации на сервере авторизации (см. подпункт 5.4.4.5);

– <redirect_uri>: (обязательный) URI переадресации, на который будет отправлен ответ; должен быть предварительно зарегистрирован на сервере авторизации (см. подпункт 5.4.4.5);
– <state>: (рекомендуемый) значение, используемое для синхронизации состояния между запросом и обратным вызовом; используется для защиты от атак межсайтовых запросов (CSRF);
– <nonce>: (опциональный) случайное строковое значение, используемое для привязки сеанса клиента к ID токену и для защиты от атак повторного воспроизведения;
– <prompt>: (опциональный) список строк, которые указывают, должен ли сервер авторизации запрашивать у конечного пользователя повторную аутентификацию и согласие на доступ клиента к ресурсу. Определены значения:
  – "none" – не требуется интерфейс пользователя,
  – "login" – серверу авторизации рекомендуется запросить повторную аутентификацию,
  – "consent" – серверу авторизации рекомендуется запросить у пользователя согласие на доступ к ресурсу,
  – "select_account" – серверу авторизации рекомендуется запросить у пользователя выбор учетной записи;
– <max_age>: (опциональный) максимальный срок аутентификации. Определяет допустимое время в секундах, прошедшее с момента последней активной аутентификации конечного пользователя сервером авторизации. Если истекшее время больше этого значения, сервер авторизации должен пытаться активно повторно аутентифицировать конечного пользователя. Если в запросе аутентификации присутствует параметр <max_age>, возвращаемый ID токен должен включать значение параметра <auth_time>.

Примечание. Дополнительные сведения по параметрам запроса аутентификации приведены в RFC 6749 [10] (пункт 4.1.1) и в OIDC [11] (подпункт 3.1.2.1).

5.4.2.3. 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. [11], подраздел 13.1). При использовании метода HTTP POST параметры запроса сериализуются с использованием сериализации форм (Form Serialization) (см. [11], подраздел 13.2).

5.4.2.4. 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 (см. [14], раздел 5) 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 [11] (пункт 3.1.2 и раздел 6).

5.4.2.5. Сервер авторизации должен проверить полученный запрос следующим образом:

1 Проверить наличие обязательных параметров запроса и их соответствие спецификациям OAuth 2.0 и OIDC.
2 Убедиться, что параметр <scope> содержит значение области действия "openid".
3 Проверить, что все обязательные параметры присутствуют и их использование соответствует данной спецификации.

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 подпунктом 5.4.2.9.

5.4.2.6. Если запрос аутентификации действителен, сервер авторизации пытается аутентифицировать конечного пользователя или определяет, аутентифицирован ли конечный пользователь. Методы, используемые сервером авторизации для аутентификации конечного пользователя (например, имя пользователя и пароль, файлы cookie сеанса и т. д.), не регламентируются настоящим стандартом.

Сервер авторизации должен запрашивать аутентификацию конечного пользователя в следующих случаях:

– конечный пользователь еще не аутентифицирован;

– 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.

При взаимодействии с конечным пользователем сервер авторизации должен использовать соответствующие меры против подделки межсайтовых запросов (CSRF) и перехвата кликов (Clickjacking).

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

– через интерактивный диалог с конечным пользователем, в ходе которого сервер авторизации отображает запрашиваемую клиентом область действия и запрашивает у конечного пользователя согласие на доступ;
– путем установления согласия с помощью условий обработки запроса;
– другими способами (например, с помощью предыдущего административного согласования).

5.4.2.8. 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.

5.4.2.9. В случае ошибки при проверке запроса аутентификации либо в процессе аутентификации конечного пользователя сервер авторизации отвечает клиенту с указанием информации об ошибке.

Параметры *сообщения об ошибке*:

– <error>: (обязательный) код ошибки;
– <error_description>: (опциональный) текстовое описание ошибки;
– <error_uri>: (опциональный) URI веб-страницы с дополнительной информацией об ошибке;
– <state>: (обязательный, если параметр <state> присутствует в запросе аутентификации) значение параметра <state> запроса аутентификации, полученного от клиента.

Возможные значения кода ошибки:

– “invalid_request”: в запросе отсутствует обязательный параметр, или он содержит недопустимое значение параметра, или содержит параметр более одного раза, или имеет иные неправильные значения параметров;
– “unauthorized_client”: клиент не авторизован для запроса кода авторизации с использованием данного метода;
– “access_denied”: владелец ресурса или сервер авторизации отклонил запрос;
– “unsupported_response_type”: сервер авторизации не поддерживает выдачу кода авторизации с использованием данного метода;
– “invalid_scope”: запрошенная область действия недействительна, неизвестна или имеет неправильный формат;
– “server_error”: сервер авторизации обнаружил непредвиденное состояние, которое не позволило ему выполнить запрос;
– “temporarily_unavailable”: сервер авторизации в настоящее время не может обработать запрос из-за временной перегрузки или обслуживания сервера;
– “interaction_required”: для авторизации на сервере авторизации требуется взаимодействие с конечным пользователем (значение параметра <prompt> запроса аутентификации равно “none”,

но запрос аутентификации не может быть выполнен без отображения пользовательского интерфейса для взаимодействия с конечным пользователем);

– “login_required”: сервер авторизации требует аутентификации конечного пользователя (значение параметра <prompt> запроса аутентификации равно “none”, но запрос аутентификации не может быть выполнен без отображения пользовательского интерфейса для аутентификации конечного пользователя);
– “account_selection_required”: конечный пользователь должен выбрать сеанс на сервере авторизации (значение параметра <prompt> запроса аутентификации равно “none”, но запрос аутентификации не может быть выполнен без отображения пользовательского интерфейса, запрашивающего сеанс);
– “consent_required”: серверу авторизации требуется согласие конечного пользователя (значение параметра <prompt> запроса аутентификации равно “none”, но запрос аутентификации не может быть выполнен без отображения пользовательского интерфейса для получения согласия конечного пользователя);
– “invalid_request_uri”: попытка доступа по адресу <request_uri> в запросе авторизации возвращает ошибку или содержит неверные данные;
– “invalid_request_object”: параметр <request> содержит недопустимый объект запроса;
– “request_not_supported”: сервер авторизации не поддерживает использование параметра <request>;
– “request_uri_not_supported”: сервер авторизации не поддерживает использование параметра <request_uri>;
– “registration_not_supported”: сервер авторизации не поддерживает использование параметра <registration>.

5.4.2.10. 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.

5.4.2.11. В случае успешной проверки ответа сервера авторизации клиент формирует и отправляет на адрес конечной точки токена *запрос токена*. Параметры запроса токена:

– <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 [10] (пункт 4.1.3) и [11] (подпункт 3.1.3.1).

5.4.2.12. Сервер авторизации должен проверить запрос токена следующим образом:

– аутентифицировать клиента в соответствии с подразделом 5.5;
– 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.

5.4.2.13. 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.

Примечание. Время действия кода авторизации определяется на этапе разработки сервера авторизации.

Сервер авторизации должен аутентифицировать клиента (см. подраздел 5.5) и гарантировать, что код авторизации был выдан клиенту, запросившему его.

5.4.2.14. Получив и успешно проверив запрос токена, сервер авторизации генерирует ID токен (см. подпункт 5.4.2.16), токен доступа (см. подпункт 5.4.2.18), а также, если необходимо, токен обновления (см. подпункт 5.4.2.19) и возвращает их клиенту по адресу <redirect_uri> в теле HTTP ответа с кодом состояния 200 (OK). В ответе используется тип формата "application/json". Ответ должен включать следующие поля и значения заголовка HTTP ответа:

```
Cache-Control: no-store
Pragma: no-cache
```

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) ID 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 подпункте 5.4.2.9. 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 [11] (подпункты 3.1.3.3 и 3.1.3.4).

5.4.2.15. 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 ID токена согласно подпункту 5.4.2.17.

5.4.2.16. *ID токен* – это токен безопасности, который содержит параметры аутентификации конечного пользователя сервером авторизации. ID токен представляется в виде структуры JWT (см. подраздел 5.7).

ID токен включает следующие параметры:

- <iss>: (обязательный) идентификатор эмитента (сервера авторизации), источника ответа; регистрозависимый URL-адрес, использующий схему https; содержит схему, хост и опционально компоненты номера порта и пути, но не компоненты запроса или фрагмента;
- <sub>: (обязательный) уникальный идентификатор субъекта, выданный сервером авторизации конечному пользователю; регистрозависимая строка длиной не более 255 символов ASCII;
- <aud>: (обязательный) аудитория, для которой предназначен данный ID токен; должна содержать идентификатор <client_id>; в общем случае значение <aud> представляет собой массив чувствительных к регистру строк; если есть только один элемент, значением <aud> может быть единственная строка;
- <exp>: (обязательный) время завершения срока действия, при наступлении и после наступления которого ID токен не должен приниматься к обработке; значением является число в формате JSON, представляющее количество секунд от 1970-01-01T0:0:0Z UTC до соответствующей даты/времени;
- <iat>: (обязательный) время, когда был выпущен токен; значением является число в формате JSON, представляющее количество секунд от 1970-01-01T0:0:0Z UTC до соответствующей даты/времени;

– <auth_time>: (обязательный, если в запросе аутентификации присутствует значение параметра <max_age>) время, когда выполнена аутентификация конечного пользователя; значением является число в формате JSON, представляющее количество секунд от 1970-01-01T0:0:0Z UTC до соответствующей даты/времени;
– <nonce>: строковое значение; равно значению параметра <nonce>, в запросе аутентификации; серверу авторизации следует включать этот параметр, если значение параметра <nonce> присутствует в соответствующем запросе аутентификации;
– <azp>: (опциональный) идентификатор клиента стороны, для которого был выпущен ID токен.

ID токен должен быть подписан с использованием JWS (см. пункт 5.7.1). Он может быть подписан с использованием JWS и затем зашифрован с использованием JWE (см. пункт 5.7.2), как структура Nested JWT (см. подраздел 5.7). При этом в ID токене не должна использоваться строка "none" в качестве значения параметра <alg>.

5.4.2.17. Проверка ID токена клиентом должна выполняться следующим образом:

1 Если ID токен зашифрован, следует расшифровать его, используя ключи и алгоритмы, которые сервер авторизации должен был использовать для шифрования ID токена как JWE (см. 5.8.1 и 5.7.2).
2 Идентификатор эмитента (сервера авторизации), полученный при регистрации клиента, должен точно соответствовать значению параметра <iss> ID токена.
3 Клиент должен убедиться, что значение параметра <aud> содержит значение <client_id>, полученное клиентом от сервера авторизации при регистрации. ID токен должен быть отклонен, если <aud> не содержит значение <client_id> клиента.
4 Если ID 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 ID токена, применяя алгоритм, указанный в параметре <alg>, используя ключ, предоставленный сервером авторизации. Если ID 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 ID 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 ID токена см. в разделе 2 [11].

5.4.2.18. *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. Данная строка является случайной последовательностью символов и не несет смысловой нагрузки для клиента.

Рекомендуется время жизни токена ограничить однократным использованием или очень 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 ресур-

су. Серверу авторизации следует предоставить конечному пользователю механизм отзыва токенов доступа и токенов обновления, предоставленных клиенту.

Примечание. Рекомендации по использованию токенов доступа см. в [11] (подраздел 16.18).

5.4.2.19. *Токен обновления* (refresh token) – это 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 (токены доступа могут иметь более короткий срок службы и меньше разрешений на доступ, чем разрешено владельцем ресурса). 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.

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.

5.4.2.20. Клиент может получить информацию о конечном пользователе, либо извлекая ее из параметров ID токена, либо с помощью запроса к конечной точке UserInfo сервера авторизации.

Чтобы получить значение параметра информации о конечном пользователе, клиент отправляет запрос конечной точке UserInfo, используя токен доступа, полученный при аутентификации на сервере авторизации. Значения параметров в ответ возвращаются в виде JSON объекта, который содержит коллекцию пар «имя и значение параметра». Конечная точка UserInfo должна принимать токены доступа на предъявителя (“Bearer”).

Конечная точка UserInfo должна поддерживать использование методов HTTP GET и HTTP POST. Взаимодействие между клиентом и сервером авторизации при обращении клиента на точку UserInfo должно быть защищено протоколом TLS.

Примечание. Сведения об алгоритмах работы в конечной точке UserInfo приведены в [11] (подраздел 5.3).

5.4.3. Гибридный сценарий аутентификации

5.4.3.1. 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 ID токен и токен доступа.
8 Клиент проверяет ID токен и извлекает идентификатор субъекта конечного пользователя (значение параметра <sub>).

5.4.3.2. 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 (см. подпункты 5.4.2.2–5.4.2.7). При этом параметр <response_type> запроса аутентификации должен иметь одно из следующих значений: “code id_token”, либо “code token”, либо “code id_token token”.

5.4.3.3. В случае ошибки проверки запроса аутентификации ответ возвращается клиенту, как в подпункте 5.4.2.9. 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>: ID 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 (см. подпункт 5.4.2.2); 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.

5.4.3.4. Формат ID токена в данном сценарии аналогичен подпункту 5.4.2.16 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 хэш-кода 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>.

5.4.3.5. Клиент должен убедиться в правильности ответа аутентификации:

– следуя правилам 5.4.2.15;
– 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>.

5.4.3.6. При использовании гибридного сценария запрос токена клиентом, проверка запроса токена, генерация токена доступа и ID 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. подпункты 5.4.2.11 – 5.4.2.15).

Если ID 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 ID токенах. Все значения параметров события аутентификации, присутствующие в любом из них, должны присутствовать в обоих ID токенах. Если какой-либо ID токен содержит параметр конечного пользователя, присутствующий в обоих ID токенах, он должен иметь одинаковые значения в обоих ID 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 [11] (подраздел 3.3).

5.4.4. Регистрация сервера авторизации (Discovery) и динамическая регистрация клиента

5.4.4.1. 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*.

5.4.4.2. Настоящий стандарт не регламентирует способ получения адреса сервера авторизации. Этот адрес может предоставляться клиенту, например указанием его в документации на систему. Сервер авторизации может также запускать сервис, поддерживающий технологию WebFinger [25].

5.4.4.3. 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, в конфигурации перечислены значения таких параметров:

- <issuer>: (обязательный) https URL адрес, являющийся идентификатором эмитента (Issuer Identifier); далее это значение должно быть указано в качестве параметра <iss> в ID токенах; получив ответ, клиент должен проверить совпадение значения параметра <issuer> с адресом сервера авторизации, на который он делал запрос;
- <authorization_endpoint>: (обязательный) URI конечной точки авторизации (см. подпункты 5.4.2.2–5.4.2.9); может включать в себя компоненту запроса в формате "application/x-www-form-urlencoded"; не должен включать компоненту фрагмента;
- <token_endpoint>: (обязательный кроме неявного сценария) URI конечной точки токена (см. подпункты 5.4.2.10–5.4.2.14); может включать в себя компоненту запроса в формате "application/x-www-form-urlencoded"; не должен включать компоненту фрагмента;
- <userinfo_endpoint>: (рекомендуемый) URI конечной точки UserInfo, предназначенный для запроса информации об аутентифицированном конечном пользователе (см. подпункт 5.4.2.20);
- <registration_endpoint>: (рекомендуемый) URI конечной точки динамической регистрации клиентов сервера авторизации (см. подпункт 5.4.4.5);
- <jwks_uri>: (обязательный) URL документа Key Set (см. подпункт 5.7.3.3) сервера авторизации, где публикуются его ключи в формате JWK; публикуемые сертификаты должны передаваться доверенным образом;
- <grant_types_supported>: (опциональный) перечень поддерживаемых сервером авторизации типов разрешений на доступ (grants); сервера авторизации OIDC должны поддерживать значения "authorization_code" и "implicit"; по умолчанию используется значение ["authorization_code", "implicit"];
- <scopes_supported>: (рекомендуемый) JSON массив значений параметра <scope>, которые поддерживает сервер авторизации;
- <response_types_supported>: (обязательный) JSON массив с перечнем поддерживаемых значений параметра <response_type>; сервера авторизации OIDC должны поддерживать значения "code", "id_token" и "token id_token";
- <response_modes_supported>: (опциональный) JSON массив с перечнем поддерживаемых значений параметра <response_mode>; по умолчанию [«query», «fragment»]; также в подразделе 8.3 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 читаемую 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 8.3 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 Discovery представлены в документах [26] и RFC 8414 [27].

5.4.4.4. Передача сообщений Discovery, в том числе и сообщений сервиса WebFinger, между клиентом и сервером авторизации должна быть защищена с помощью протокола TLS с обеспечением конфиденциальности и целостности. При этом должна выполняться аутентификация источника информации Discovery.

5.4.4.5. Процедура *динамической регистрации клиента* выполняется после того, как выполнена регистрация сервера авторизации, описанная выше.

Для регистрации клиент посылает HTTP POST запрос серверу авторизации на конечную точку регистрации клиентов (параметр <registration_endpoint> документа Discovery, см. подпункт 5.4.4.1) 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 (см. подпункт 5.4.2.2); 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";
- <application_type>: (опциональный) тип клиента ("native" или "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. подпункт 5.7.3.3), содержащего ключи клиента в формате JWK; публикуемые сертификаты должны передаваться доверенным образом;
- <jwks>: (опциональный) документ JWK Set, передаваемый по значению; параметры <jwks_uri> и <jwks> не должны одновременно присутствовать в метаданных клиента; сертификаты открытых ключей должны передаваться доверенным образом;
- <default_max_age>: (опциональный) максимальный возраст 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 ID 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 5.5).

5.4.4.6. Сервер авторизации может игнорировать значения, предоставленные клиентом, и должен игнорировать любые присланные клиентом метаданные, которые он не понимает. Он должен проверить все заявленные значения метаданных клиента на их совместимость со спецификациями OIDC, 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 OIDC;

– <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.

5.4.4.7. *При ошибке обработки запроса регистрации* (см. подпункт 5.4.4.6*)* 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 подпункта 5.4.2.9. 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 [28] и [29].

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

5.5. АУТЕНТИФИКАЦИЯ КЛИЕНТА

5.5.1. При динамической регистрации (см. подпункт 5.4.4.5) клиент может зарегистрировать метод аутентификации клиента, указав его идентификатор в качестве значения параметра <token_endpoint_auth_method>. Если этот метод не зарегистрирован, по умолчанию используется метод “client_secret_basic”. Можно использовать следующие варианты механизмов аутентификации:

– “client_secret_basic”: аутентификация с использованием базовой схемы аутентификации HTTP и <client_secret>;
– “client_secret_post”: аутентификация с включением учетных данных клиента в тело запроса и с использованием <client_secret>;

Примечание. Спецификации механизмов “client_secret_basic” и “client_secret_post” представлены в [10] (пункт 2.3.1).

– “client_secret_jwt”: аутентификация с использованием структуры JWT на основе кода аутентификации HMAC и <client_secret> (см. пункт 5.5.2);
– “private_key_jwt”: аутентификация с использованием структуры JWT на основе цифровой подписи (см. пункт 5.5.3);
– “tls_client_auth”: аутентификация с использованием TLS с взаимной аутентификацией и PKI для связывания сертификата с клиентом (см. пункт 5.5.4);
– “self_signed_tls_client_auth”: аутентификация с использованием TLS с взаимной аутентификацией и самоподписанного сертификата клиента (см. пункт 5.5.5);
– “none”: клиент не аутентифицирует себя в конечной точке токена либо потому, что он использует только неявный сценарий аутентификации (и поэтому не использует конечную точку токена), либо потому, что он является публичным клиентом без <client_secret>, либо используется другой механизм аутентификации.

В реализациях настоящего стандарта допускается использование механизмов аутентификации клиента “client_secret_basic”, “client_secret_post” и “none” только в тестовом режиме.

5.5.2. Механизм аутентификации «client_secret_jwt»

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. пункт 5.7.4):

- <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) время, по истечении которого токен не должен быть принят к обработке;
- <iat>: (опциональный) время выпуска JWT.

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".

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 отслеживаться. Сервер авторизации должен отклонить любой JWT с истекшим временем действия при условии допустимого расхождения часов. 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 необоснованно далеко от настоящего.

7 JWT может содержать значение параметра <jti> (JWT ID), которое предоставляет уникальный идентификатор токена. Сервер авторизации может гарантировать, что JWT не будут воспроизведены, поддерживая набор используемых значений <jti> в течение определенного промежутка времени, в течение которого JWT будет считаться действительным, на основе применимого момента <exp>.

8 JWT может содержать другие параметры.

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.

В случае ошибки проверки JWT сервер авторизации должен ответить кодом ошибки "invalid_client" в качестве значения параметра <error> (см. подпункт 5.4.2.9). В случае успешной проверки JWT клиент считается аутентифицированным.

Примечание. Дополнительные сведения о данном методе аутентификации приведены в RFC 7521 [30] и RFC 7523 [31].

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

5.5.3. *Механизм аутентификации “private_key_jwt”*

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 5.5.2.

5.5.4. *Механизм аутентификации с использованием TLS с взаимной аутентификацией и PKI*

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

1 В ходе динамической регистрации (см. подпункт 5.4.4.5) 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 [32].

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`”.

Такой механизм аутентификации позволяет просто выполнять обновление ключей и сертификата клиента в соответствии со штатной процедурой PKI.

5.5.5. *Механизм аутентификации с использованием* TLS с взаимной аутентификацией *и самоподписанного сертификата клиента*

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. подпункт 5.7.3.3), 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. пункт 5.5.4) 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).

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.

5.6. ДОСТУП К ЗАЩИЩЕННОМУ РЕСУРСУ

5.6.1. 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. При этом предполагается использование связи сервера ресурса с сервером авторизации.

5.6.2. 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 [33] (раздел 2).

Клиенту рекомендуется использовать этот метод авторизации. Сервер ресурса должен его поддерживать.

5.6.3. Альтернативный вариант передачи токена доступа от клиента серверу ресурса – в теле HTTP-запроса как значение параметра <access_token>. При этом должны быть выполнены следующие условия:

- поле <Content-Type> заголовка HTTP-запроса имеет значение “application/x-www-form-urlencoded”;
- тело запроса соответствует требованиям кодирования контента “application/x-www-form-urlencoded”;
- тело HTTP-запроса состоит из одной части, то есть запрос не является запросом типа “multipart/form-data”;
- содержимое тела запроса должно состоять исключительно из ASCII-символов;
- должен использоваться HTTP метод, семантика тела запроса которого строго определена. В частности, это означает, что HTTP метод GET не должен использоваться.

Клиенту не рекомендуется использовать такой метод авторизации, кроме случаев, когда браузер не имеет доступа к <Authorization> заголовку HTTP-запроса. Сервер ресурсов может поддерживать этот метод.

Примечание. Дополнительные сведения о протоколе доступа клиента к защищенному ресурсу с использованием токена доступа приведены в RFC 6750 [13].

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

5.7. JWT

5.7.1. Цифровая подпись в формате JSON

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

5.7.1.2. JWS состоит из трех частей:

- <JOSE Header>: JOSE заголовок – JSON-объект, содержащий параметры, описывающие криптографические операции и их параметры (см. подпункт 5.7.1.4);
- <JWS Payload>: функциональное содержимое JWS (полезная нагрузка) – последовательность октетов, подлежащих защите, другими словами – сообщение; функциональное содержимое может состоять из произвольной последовательности октетов;
- <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:

– <JWS Protected Header>: защищенный заголовок JWS – JSON-объект, содержащий параметры заголовка, целостность которых защищена цифровой подписью JWS или операцией MAC;
– <JWS Unprotected Header>: незащищенный заголовок JWS – JSON-объект, который содержит параметры заголовка, целостность которых не обеспечивается.

5.7.1.3. Для передачи структуры JWS может использоваться один из двух типов сериализации:
– компактная сериализация – представление JWS в виде компактной URL-строки,
– JSON-сериализация – представление JWS в виде объекта JSON.

При использовании компактной сериализации JWS представляется как строка:

BASE64URL(UTF8(<JWS Protected Header>) ‖ ‘.’ ‖ BASE64URL
(<JWS Payload>) ‖ ‘.’ ‖ BASE64URL(<JWS Signature>)

Порядок элементов строки критичен.

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>).

5.7.1.4. 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. Реализации настоящего стандарта должны понимать параметры <JOSE Header>, обозначенные в стандарте как «должны быть поняты», и обрабатывать их так, как определено в стандарте. Все остальные параметры <JOSE Header>, определенные в настоящем стандарте, но не обозначенные таким образом, должны игнорироваться, если они не поняты.

<JOSE Header> может включить 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. пункт 5.7.3), который используется для проверки цифровой подписи JWS;
– <jwk>: (опциональный) ключ проверки подписи в представлении JWK (см. пункт 5.7.3), 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 [32] 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 [32], 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 [14], не Base64url кодирование) значение 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 [32], считать сертификат или цепочку сертификатов и подпись JWS недействительными в случае отрицательного результата проверки;

– <typ> (Type): (опциональный) тип сериализации (“JOSE” – компактная сериализация, “JOSE+-JSON” – JSON-сериализация);
– <cty> (Content Type): (опциональный) тип (media type) функционального содержимого JWS; как правило, 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 понят и обработан получателем.

5.7.1.5. 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>)) ‖ ‘.’ ‖ 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 [34].
2 Выбор используемых криптографических алгоритмов производится на этапе разработки сервера авторизации и клиента.

5.7.2. JSON-структура шифрованного текста

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

Структура JWE:

– <JOSE Header>: JOSE заголовок – JSON-объект, содержащий параметры, описывающие криптографические операции и их параметры (см. подпункт 5.7.1.4);
– <JWE Encrypted Key>: зашифрованный ключ защиты содержимого JWE для каждого получателя;
– <JWE Initialization Vector>: синхропосылка (вектор инициализации) JWE;
– <JWE AAD>: дополнительные аутентифицируемые, не шифруемые данные JWE;
– <JWE Ciphertext>: шифротекст JWE;
– <JWE Authentication Tag>: имитовставка JWE.

В состав <JOSE Header> входят следующие компоненты:

– <JWE Protected Header>: защищенный заголовок JWE, целостность значения которого будет защищаться JWE;
– <JWE Shared Unprotected Header>: общий незащищенный заголовок JWE;
– <JWE Per-Recipient Unprotected Header>: незащищенный заголовок для каждого получателя JWE.

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

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

BASE64URL(UTF8(<JWE Protected Header>)) ‖ ‘.’ ‖ BASE64URL
(<JWE Encrypted Key>) ‖ ‘.’ ‖ BASE64URL(<JWE Initialization Vector>) ‖ ‘.’ ‖ BASE64URL
(<JWE Ciphertext>) ‖ ‘.’ ‖ 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:

– <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;

– <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;
– <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;
– <iv>: если значение <JWE Initialization Vector> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE Initialization Vector>); иначе этот параметр должен быть опущен;
– <aad>: если значение <JWE AAD> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE AAD>); иначе этот параметр должен быть опущен;
– <ciphertext>: (обязательный) шифротекст со значением BASE64URL(<JWE Ciphertext>);
– <tag>: если значение <JWE Authentication Tag> не пустое, этот параметр должен присутствовать со значением BASE64URL(<JWE Authentication Tag>); иначе этот параметр должен быть опущен.

5.7.2.3. Параметры JOSE заголовка:

– <enc>: (обязательный) идентификатор алгоритма шифрования, используемого для выполнения аутентифицированного шифрования открытого текста; этот параметр должен быть понят и 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. пункт 5.7.1); 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.

5.7.2.4. Шифрование выполняется следующим образом:

– в соответствии с идентификатором алгоритма шифрования вычислить ключ шифрования содержимого CEK;
– если используется алгоритм с шифрованием ключа, зашифровать ключ CEK, положив значением <JWE Encrypted Key> результат шифрования; если алгоритм шифрования не предполагает шифрования ключа, положить значение <JWE Encrypted Key> пустым;
– если требуется для алгоритма шифрования, сгенерировать значение случайной синхропосылки <JWE Initialization Vector> необходимой длины; иначе значение <JWE Initialization Vector> должно быть пустое;
– если указано значение параметра "zip", положить M – последовательность октетов, представляющих сжатый открытый текст; иначе M – последовательность октетов, представляющих открытый текст;
– зашифровать M с помощью алгоритма, который идентифицирован параметрами <alg> и <enc>, с использованием значений ключа CEK, синхропосылки <JWE Initialization Vector> и <JWE AAD>; результат – 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 [35].

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

5.7.3. JSON структура ключа

5.7.3.1. 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.

5.7.3.2. В состав JWK входят следующие параметры:

- <kty>: (обязательный) тип ключа; определены следующие значения:
  - "EC": ключ для криптографического алгоритма на базе эллиптической кривой,
  - "oct": последовательность октетов, используемая для представления симметричного ключа;
- <use>: (опциональный) предполагаемое использование открытого ключа; определены следующие значения:
  - "sig" – подпись,
  - "enc" – шифрование;
- <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;
- <alg> – (опциональный) идентифицирует криптографический алгоритм, в котором предполагается использование ключа;
- <kid> – (опциональный) идентификатор ключа; используется для того, чтобы выбрать нужный ключ в документе JWK Set;
- <x5u>, <x5c>, <x5t>, <x5t#S256>: (опциональные) как в JWS (см. пункт 5.7.1).

В зависимости от типа ключа <kty> в состав структуры JWK включаются также другие специфические параметры.

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

5.7.3.4. Документ Key Set, на который указывает значение параметра <jwks_uri> документа Discovery сервера авторизации (см. подпункт 5.4.4.1), 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. подпункт 5.4.4.5), 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 [36].

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

5.7.4. Структура JSON веб-токена

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

Имена зарегистрированных параметров JWT:

– <iss>: (опциональный) идентификатор эмитента JWT (сервера авторизации, выдавшего его);
– <sub>: (опциональный) идентификатор субъекта JWT;
– <aud>: (опциональный) перечень идентификаторов получателей, которым предназначен JWT;
– <exp>: (опциональный) время завершения срока действия JWT в секундах;
– <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.

5.7.4.2. 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.

5.7.4.3. 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 [12].

5.8. МЕХАНИЗМЫ ЗАЩИТЫ

5.8.1. Цифровая подпись и шифрование

5.8.1.1. В зависимости от транспорта, с помощью которого отправляются сообщения, целостность сообщения может не гарантироваться, а отправитель сообщения может не проходить проверку подлинности. Чтобы уменьшить эти риски, при обработке значений ID токена, ответа UserInfo, объекта запроса и 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.

Сервер авторизации может объявлять о поддерживаемых им алгоритмах подписи и шифрования в своем документе Discovery (см. подпункт 5.4.4.1) или предоставлять эту информацию другими способами. Клиент может декларировать свои алгоритмы цифровой подписи и шифрования в своем запросе динамической регистрации (см. подпункт 5.4.4.5) или передавать эту информацию другими способами.

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

5.8.1.2. *Цифровая подпись*

Подписывающая сторона должна выбрать алгоритм цифровой подписи на основе алгоритмов, поддерживаемых получателем.

Ключ асимметричной цифровой подписи

Ключ 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 есть несколько ключей, в JOSE-заголовке должно присутствовать значение параметра <kid>. Параметр <use> соответствующего ключа должен указывать на то, что этот ключ поддерживает алгоритм цифровой подписи ("sig").

Ключ кода аутентификации сообщения

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>. Требования к энтропии значений <client_secret> см. в пункте 5.8.2. Симметричная подпись не должна использоваться публичными клиентами из-за их неспособности сохранять ключ клиента.

5.8.1.3. Смена асимметричных ключей подписи

Смена асимметричных ключей цифровой подписи может быть выполнена с помощью следующего подхода. Подписывающий публикует свои ключи в документе JWK Set, местоположение которого ука-

зывает в качестве значения параметра <jwks_uri> документа Discovery (см. подпункт 5.4.4.1), 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. Ключи можно обновлять, периодически добавляя новые ключи проверки цифровой подписи в JWK Set по адресу <jwks_uri>. Подписывающий может начать использовать новый ключ цифровой подписи по своему усмотрению, сигнализируя об этом проверяющему, использовав новое значение параметра <kid>. Проверяющий понимает, что, обнаружив незнакомое значение параметра <kid>, он 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, соответствующего требованиям к СКЗИ.

5.8.1.4. Шифрование

Отправитель, выполняющий шифрование передаваемой информации, должен выбрать алгоритм шифрования из тех алгоритмов, которые поддерживает получатель. Перечень этих алгоритмов указан в конфигурации сервера авторизации (см. подпункт 5.4.4.1) и клиента (см. подпункт 5.4.4.5).

Асимметричное шифрование на основе арифметики эллиптической кривой

Генерируется эфемерный открытый ключ с использованием арифметики эллиптической кривой в качестве элемента <epk> JOSE-заголовка JWE. Второй открытый ключ, используемый для ключевого соглашения (key agreement), должен быть открытым ключом, опубликованным получателем в его документе JWK Set. Если в указанном документе JWK Set есть несколько ключей, в JOSE-заголовке JWE должно быть указано значение параметра <kid>. Для согласования ключа шифрования контента, который будет использован для шифрования подписанного JWT, следует использовать алгоритм ключевого соглашения с использованием арифметики эллиптической кривой. Параметр <use> соответствующего ключа должен включать шифрование (“enc”).

Симметричное шифрование

Симметричный ключ шифрования вычисляется 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.

5.8.1.5. Смена асимметричных ключей шифрования

При смене 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. Шифрующий 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.

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 пункта 10.2.1 [11]. 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. Продолжительность кэша следует также координировать с выдачей новых ключей подписи, как описано в подпункте *5.8.1.3*.

5.8.2. Требования к энтропии симметричных ключей

Ключи формирования MAC JWS и симметричного шифрования JWE вычисляются на основе значения ключа клиента <client_secret> (см. подпункт 5.4.4.5). Таким образом, при использовании с симметричными операциями подписи (MAC) или шифрования значения <client_secret> должны содержать достаточную энтропию для генерации криптографически стойких ключей.

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

Кроме того, длина значения <client_secret> должна быть не менее той, которая требуется для ключей MAC для конкретного используемого алгоритма. 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.

5.8.3. Требования к TLS

5.8.3.1. Приложения, соответствующие настоящему стандарту, должны выполнять следующие требования к использованию протокола TLS.

Реализация протокола TLS с использованием криптографических алгоритмов Российской Федерации должна выполняться в соответствии с положениями рекомендаций по стандартизации Р 1323565.1.020. Допускается использование сертифицированных федеральным органом исполнительной власти в области обеспечения безопасности СКЗИ, реализующих TLS в соответствии с требованиями МР 26.2.001-2013 [37], до окончания срока действия сертификата на использование СКЗИ.

5.8.3.2. Все взаимодействия между клиентом и сервером авторизации, между клиентом и сервером ресурсов, между сервером авторизации и сервером ресурсов должны быть защищены с использованием TLS (HTTPS). Данное требование относится к взаимодействиям в рамках всех указанных в стандарте протоколов (регистрация клиентов, взаимодействие по протоколам OIDC), ко всем сценариям (с кодом авторизации и гибридный), ко всем режимам доступа к защищаемому ресурсу (только чтение и чтение/запись).

5.8.3.3. Для всех коммуникаций должен использоваться TLS версии 1.2 или более поздней.

5.8.3.4. Должна осуществляться проверка сертификата TLS-сервера.

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

5.8.3.6. Должны использоваться только следующие криптонаборы:

– `TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC` (Р 1323565.1.020),
– `TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC` (Р 1323565.1.020),
– `TLS_GOSTR341112_256_WITH_28147_CNT_IMIT` [37].

5.8.3.7. Сервер авторизации, сервер ресурсов не должны быть доступны без использования TLS. В случае обращения клиента без использования TLS сервер авторизации, сервер ресурсов должны отказать в соединении или возвратить сообщение со статусом HTTP 301 и кодом ошибки “`Moved Permanently`”.

5.8.4. Токен доступа, связанный с MTLS-сертификатом клиента

5.8.4.1. Если клиент использует взаимную аутентификацию по протоколу TLS 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 (см. подпункт *5.8.4.2*) или посредством интроспекции токена (запроса информации о токене) у сервера авторизации (см. подпункт *5.8.4.3*). Эта привязка может выполняться как совместно с аутентификацией клиента по сертификату MTLS (см. пункты 5.5.4 и 5.5.5), так и отдельно от аутентификации клиента сервером авторизации, что позволяет MTLS во время защищенного доступа к ресурсам служить исключительно механизмом подтверждения владения закрытым ключом.

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.

В процессе доступа к ресурсам, защищенным протоколом TLS с взаимной аутентификацией сторон, клиент может выполнять запросы к защищенным ресурсам, как описано в подразделе 5.6, 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.

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`”.

Метаданные, необходимые для того, чтобы сервер и клиент сигнализировали о желании использовать токены доступа с привязкой к MTLS-сертификату клиента, определены в подпункте *5.8.4.4*.

5.8.4.2. Метод подтверждения отпечатка сертификата с использованием JWT

Чтобы представить хэш-код X.509 сертификата в формате JWT (5.7), в качестве значения параметра токена <cnf> используется строка, идентифицирующая метод подтверждения на основе хэш-функции с длиной образа не менее 256 битов, и значение хэш-кода, которое формируется как Base64url кодирование значения хэш-функции, вычисленное от DER-представления (по ГОСТ Р ИСО/МЭК 8825-1) сертификата формата X.509.

Примечание. Дополнительные сведения о методе подтверждения отпечатка сертификата с использованием JWT приведены в [39] (подраздел 3.1).

Ниже приведен пример функционального содержимого JWT, содержащего подтверждение отпечатка сертификата.

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

5.8.4.3. Метод подтверждения отпечатка сертификата с использованием интроспекции токена

Интроспекция токена OAuth 2.0 определяет способ, с помощью которого сервер ресурсов может запрашивать у сервера авторизации информацию о состоянии активности токена доступа, а также другую метаинформацию о токене доступа.

Для токена доступа, связанного с сертификатом MTLS, хэш-код сертификата, с которым связан токен, передается серверу защищенного ресурса в виде метаинформации в составе ответа интроспекции токена. Хэш-код передается 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 *5.8.4.2*, в качестве параметра верхнего уровня JSON-ответа интроспекции. 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 интроспекции токена доступа представлены в RFC 7662 [38].

2 Дополнительные сведения о методе подтверждения отпечатка сертификата с использованием интроспекции токена приедены в [39] (подраздел 3.1).

Ниже приведен пример ответа интроспекции для активного токена с подтверждением отпечатка сертификата.

```
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"
    }
}
```

5.8.4.4. Метаданные сервера авторизации и клиента для подтверждения отпечатка MTLS-сертификата

Следующий параметр метаданных сервера авторизации, публикуемый в соответствии с требованиями подпункта 5.4.4.1, указывает на способность сервера авторизации выдавать токены доступа с привязкой к сертификату:

– `<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`".

Следующий параметр метаданных клиента, публикуемый в соответствии с требованиями подпункта 5.4.4.5, позволяет сигнализировать о намерении клиента 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`".

Если клиент, который указал намерение использовать токены, привязанные к MTLS-сертификату клиента, отправляет запрос на конечную точку токена по соединению, не являющемуся MTLS, на усмотрение сервера авторизации остается принятие решения относительно того, следует ли возвращать ошибку или выдать токен, не связанный с сертификатом.

6. ПРОФИЛЬ БЕЗОПАСНОСТИ OPENID API ДЛЯ ДОСТУПА К СЕРВИСАМ В РЕЖИМЕ ТОЛЬКО ДЛЯ ЧТЕНИЯ

6.1. ВВОДНАЯ ИНФОРМАЦИЯ

Раздел 6 определяет требования по использованию OIDC, обеспечивающие приложению:

– получение токенов доступа и в некоторых случаях токенов обновления OAuth для доступа к защищенным данным только для чтения;
– использование OIDC для идентификации клиента (пользователя);
– использование токенов для чтения защищенных данных с конечных точек REST.

В разделе приведены спецификации ФАПИ.СЕК, которые являются реализацией REST-стиля построения API с использованием форматов JSON. Эти API защищены технологиями авторизации OAuth 2.0 (см. подраздел 5.3) и OpenID Connect (см. подраздел 5.4).

6.2. ПОЛОЖЕНИЯ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ АВТОРИЗАЦИИ

6.2.1. В связи с тем, что на основании стандарта ФАПИ.СЕК может предоставляться потенциально чувствительная информация, она требует более высокого уровня защиты, чем базовые требования OAuth 2.0.

6.2.2. *Сервер авторизации*:

1 должен поддерживать конфиденциальных клиентов;

2 не должен поддерживать доступ в режиме чтения со стороны публичных клиентов;

3 в случае использования симметричного ключа должен предоставлять ключ клиента `<client_secret>`, соответствующий требованиям пункта 5.8.2;

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

– TLS с взаимной аутентификацией сторон взаимодействия протокола OAuth 2.0 и его профилей, включая OIDC, как определено в пунктах 5.5.4 или 5.5.5;

<client_secret_JWT> или <private_key_JWT> , как определено в пунктах 5.5.2 и 5.5.3;

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

6 должен требовать предварительной регистрации URI переадресации клиента;

7 должен требовать наличия параметра `<redirect_URI>` в запросе аутентификации (см. подпункт 5.4.2.2);

8 должен требовать, чтобы значение параметра `<redirect_URI>` точно соответствовало одному из предварительно зарегистрированных URI переадресации клиента (см. подпункт 5.4.4.5);

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

10 должен исключать повторное использование кодов авторизации в соответствии с рекомендациями подпункта 5.4.2.13;

11 должен возвращать ответ с токеном доступа и, если это установлено при разработке сервера авторизации, с токеном обновления в соответствии с подпунктом 5.4.2.10;

12 должен возвращать список предоставленных областей действия (параметр <scope>) по выданному токену доступа (см. подпункт 5.4.2.20);

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

Примечание. Здесь и далее значения токенов доступа и параметров <nonce> должны генерироваться с помощью СКЗИ, используемого при реализации криптографической защиты информации.

14 рекомендуется уведомлять владельца ресурса о том, что клиент запрашивает долгосрочное разрешение на доступ (см. подпункт 5.4.2.19);

15 рекомендуется предоставлять конечному пользователю механизм аннулирования токенов доступа и токенов обновления, выданных клиенту (см. подпункт 5.4.2.19);

16 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”;

17 должен требовать, чтобы сервисы по адресу URI переадресации использовали протокол HTTPS.

6.2.3. При предоставлении *клиенту в ответе на запрос токена доступа* идентификатор аутентифицированного пользователя сервер авторизации:

1 должен поддерживать запрос аутентификации в соответствии с подпунктом 5.4.2.2;

2 должен осуществлять проверку запроса аутентификации в соответствии с подпунктом 5.4.2.5;

3 должен предоставлять ответ на запрос аутентификации в соответствии с подпунктами 5.4.2.7 и 5.4.2.8 в зависимости от результата аутентификации;

4 должен осуществлять проверку запроса токена в соответствии с подпунктом 5.4.2.12;

5 если параметр <scope> запроса аутентификации включает “openid”, должен генерировать ID токен в составе ответа на запрос токена в соответствии с подпунктом 5.4.2.14; при этом значение параметра <sub> должно соответствовать аутентифицированному пользователю.

6.2.4. *Клиент*:

1 может использовать механизмы, определенные в разделе 7;

2 должен использовать разные URI переадресации для каждого сервера авторизации, на котором зарегистрированклиент;

3 должен поддерживать следующие методы аутентификации в конечной точке токена:
– TLS с взаимной аутентификацией клиента OAuth, как определено в пунктах 5.5.4 и 5.5.5;
– <client_secret_JWT> или <private_key_JWT> , как определено в пунктах 5.5.2 и 5.5.3 ;

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

5 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:

6 должен включать строку “openid” в перечень значений параметра <scope>;

7 должен включать параметр <nonce> (см. подпункт 5.4.2.2) в состав запроса аутентификации.

Если строка “openid” не включена в перечень значений параметра <scope>, конфиденциальный клиент:

8 должен включать параметр <state> (см. подпункт 5.4.2.2).

6.3. ТРЕБОВАНИЯ К ДОСТУПУ К ЗАЩИЩЕННЫМ РЕСУРСАМ ТОЛЬКО ДЛЯ ЧТЕНИЯ

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

6.3.2. *Сервер ресурсов*, предоставляющий доступ к данным в соответствии с настоящим стандартом:

1 должен поддерживать метод HTTP GET;
2 должен принимать токены доступа в HTTP заголовке в соответствии с пунктом 5.6.2;
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 чтение ресурса, 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, иначе возвращать ошибку (см. подпункт 5.4.2.9);
8 должен кодировать ответ в UTF-8;
9 должен отправлять параметр <Content-Type> HTTP заголовка в виде “Content-Type: application/JSON; charset = UTF-8”;
10 должен отправлять текущую дату на сервере в HTTP заголовке даты;
11 должен устанавливать в качестве значения заголовка ответа <x-FAPI-interaction-id> либо значение, полученное из соответствующего заголовка запроса клиента, либо значение UUID, если заголовок запроса не предоставлялся для отслеживания взаимодействия, например, “x-FAPI-interaction-id: c770aef3-6784-41f7-8e0e-ff5f97bddb3a”;
12 должен регистрировать значение <x-FAPI-interaction-id> в записи журнала регистрации.

Примечания

1 Форматы и алгоритмы формирования UUID определены в RFC 4122 [40].
2 Для получения информации об объекте, связанном с токеном доступа и предоставленной областью действия, сервер ресурсов может использовать протокол интроспекции токена, описанный в RFC 7662 [38].

6.3.3. *Клиент*, реализующий доступ к защищенному ресурсу в соответствии с требованиями данного стандарта:

1 должен отправлять токены доступа в заголовке HTTP в соответствии с пунктом 5.6.2;
2 может отправлять последнюю дату регистрации пользователя в клиенте в заголовке <x-FAPI-auth-date>, причем значение предоставляется в формате HTTP даты; например, “x-FAPI-auth-date: Tue, 11 Sep 2012 19:43:31 GMT”;
3 может отправлять в заголовке <x-FAPI-customer-ip-address> IP-адрес пользователя, если эти данные ему доступны;например, “x-FAPI-customer-ip-address: 198.51.100.119”;
4 для синхронизации записей журналов регистрации клиента и сервера ресурсов может отправлять серверу заголовок <x-FAPI-interaction-id> запроса, значением которого является UUID; например, “x-FAPI-interaction-id: c770aef3-6784-41f7-8e0e-ff5f97bddb3a”.

7. ПРОФИЛЬ БЕЗОПАСНОСТИ OPENID API ДЛЯ ДОСТУПА К СЕРВИСАМ В РЕЖИМЕ ЧТЕНИЯ И ЗАПИСИ

7.1. ВВОДНАЯ ИНФОРМАЦИЯ

7.1.1. В данном разделе определяются требования к тому:

– как приложение может получить токен OAuth для доступа к данным с повышенным уровнем риска,
– как приложение может использовать OpenID Connect для идентификации пользователей;
– как использовать токены для взаимодействия с конечными точками REST, предоставляющими защищенные данные.

В данном разделе определены спецификации ФАПИ.СЕК, являющиеся интерфейсами REST API. В этих спецификациях предоставлены JSON-структуры для форматов данных с повышенным риском. Эти про-

граммные интерфейсы защищены протоколами авторизации OAuth 2.0 (см. подраздел 5.3) и OpenID Connect (см. подраздел 5.4).

В рамках настоящего протокола также допустимо включение в качестве одного из параметров ID токена временного идентификатора субъекта – владельца ресурса. В этом случае ID токен выполняет функции *отдельной подписи* (detached signature) ответа на запрос авторизации.

В данном разделе ответ на запрос авторизации защищается путем включения хэш-значений всех незащищенных параметров ответа, например <code> и <state>.

7.1.2. В подпункте 5.4.3.2 определен алгоритм выработки хэш-кода параметра <code> (параметр <c_hash> ID токена). Аналогично хэш-значение параметра <state> определяется следующим образом:

– <s_hash>: параметр ID токена, значение которого равно хэш-коду значения параметра <state>. Его значение вычисляется как Base64url кодирование левой половины значения хэш-функции октетов ASCII представления значения <state>. При этом используется алгоритм хэширования, указанный в параметре <alg> JOSE заголовка ID токена. Значение параметра <s_hash> вычисляет сервер авторизации, а проверяет клиент при получении ID токена.

7.2. ПОЛОЖЕНИЯ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ АВТОРИЗАЦИИ

7.2.1. Доступ для чтения и записи представляет повышенный риск; соответственно, требуемый уровень защиты выше, чем в случае доступа только для чтения.

Данный документ предписывает выполнение требований пунктов 7.2.2, 7.2.3 и 7.2.4 при использовании технологии авторизации OAuth 2.0 для защиты API чтения и записи.

7.2.2. *Сервер авторизации* должен поддерживать положения, определенные в 6.2.2.

Кроме того, для выполнения операции записи сервер авторизации:

1 должен требовать, чтобы параметр <request> или <request_URI> передавался как токен JWT (см. 5.7.4), подписанный в соответствии с JWS (см. пункт 5.7.1);

2 должен требовать в качестве значения параметра <response_type>: “code id_token token”;

3 должен возвращать ID токен как отдельную подпись (detached signature) ответа на запрос авторизации;

4 если клиент предоставил значение <state>, для защиты этого значения должен включать в ID токен значение параметра <s_hash> (см. пункт 7.1.2) – хэш-код параметра <state>. Значение <s_hash> может быть опущено в ID токене, возвращенном из конечной точки токена, когда <s_hash> присутствует в ID токене, возвращенном из конечной точки авторизации;

5 должен генерировать только код авторизации, токен доступа и токен обновления, которые связаны с владельцем ключа, чей сертификат был предъявлен серверу на этапе установления TLS-соединения;

6 должен поддерживать механизм подтверждения владения ключом с привязкой токена к MTLS сертификату (см. пункт 5.8.4) или с привязкой токенов к TLS-соединению;

Примечание. Общие сведения о технологии привязки токенов OAuth 2.0 к TLS- соединению приведены в документе «OAuth 2.0 Token Binding» [41].

7 должен поддерживать подписанные ID токены;

8 рекомендуется поддерживать подписанные и зашифрованные ID токены;

9 должен требовать, чтобы все параметры передавались в качестве атрибутов подписанного объекта запроса (см. подпункт 5.4.2.4), передаваемого в параметре <request> или <request_URI>;

10 может поддерживать конечную точку API для обработки запроса, включающего в качестве своей компоненты <payload> объект запроса, как описано в подразделе 7.4;

11 должен требовать, чтобы объект запроса (см. подпункт 5.4.2.4) содержал параметр <exp>;

12 должен аутентифицировать конфиденциального клиента в конечной точке токена, используя один из следующих методов (это отменяет перечисление 4 в пункте 6.2.2):

– TLS с взаимной аутентификацией клиента OAuth, как определено в пункте 5.5.4 или 5.5.5;

– <private_key_JWT>, как определено в пункте 5.5.3.

7.2.3. *Клиент* должен поддерживать положения пункта 6.2.4.

Кроме того, для выполнения операций записи клиент:

1 должен поддерживать механизм подтверждения владения ключом с привязкой токена к MTLS сертификату (см. пункт 5.8.4) или с привязкой токена к TLS-соединению.

Примечание. В случае конфиденциального клиента MTLS может также использоваться как механизм аутентификации клиента (см. пункты 5.5.4 или 5.5.5).

2 должен включать в запрос аутентификации параметр `<request>` или `<request_URI>` (см. подпункт 5.4.2.4);

3 должен требовать, чтобы конечные точки возвращали ID токен, подписанный с использованием JWS (см. пункт 5.7.4);

4 используя ID токен в качестве отдельной подписи, должен проверять, что ответ на запрос авторизации не был подделан;

5 при проверке ID токена должен проверять значение параметра `<state>` запроса аутентификации путем проверки значения параметра `<s_hash>` (пункт 7.1.2);

6 должен отправлять все параметры в составе подписанного объекта запроса авторизации (подпункт 5.4.2.4);

7 должен дополнительно отправлять дубликаты параметров с использованием синтаксиса запроса OAuth 2.0, когда этого требует спецификация OAuth;

8 должен требовать, чтобы конечные точки возвращали подписанные с использованием JWS (пункт 5.7.1) и зашифрованные с использованием JWE (пункт 5.7.2) ID токены, необходимые для обеспечения защиты любых персональных данных, содержащихся в ID токене, предоставленном в качестве отдельной подписи в ответе на запрос авторизации.

Примечания

1 Общие сведения об алгоритмах проверки запроса авторизации приведены в подпункте 5.4.2.5, дополнительные сведения приведены в [11] (подпункты 3.2.2.11, 3.3.2.12).

2 Дополнительные сведения о передаче объекта запроса по значению приведены в [11] (подраздел 6.1).

7.2.4. *Режим защищенного ответа на запрос авторизации на базе JWT*

В дополнение к положениям, представленным в пункте 7.2.2, сервер авторизации может обеспечивать *защиту ответов на запрос авторизации* клиентов, используя JARM (см. раздел 8).

JARM позволяет клиенту потребовать от сервера авторизации, чтобы он кодировал ответы на запросы авторизации (любого типа), используя JWT. Это альтернатива использованию ID токенов в качестве отдельных подписей для обеспечения безопасности ответов на запросы авторизации.

Серверу авторизации рекомендуется известить о поддержке режимов ответа JARM, используя параметр метаданных `<response_modes_supported>` (см. подпункт 5.4.4.1).

В случае использования JARM для защиты ответов на запросы авторизации перечисления 2, 3 и 4 из пункта 7.2.2 не применяются. Например, клиенты могут использовать JARM в сочетании с типом ответа "`code`".

7.3. ТРЕБОВАНИЯ К ДОСТУПУ К ЗАЩИЩЕННЫМ РЕСУРСАМ ДЛЯ ЧТЕНИЯ И ЗАПИСИ (С ИСПОЛЬЗОВАНИЕМ ТОКЕНОВ)

7.3.1. Конечные точки ФАПИ.СЕК – это конечные точки ресурсов, защищенных с помощью протокола OAuth 2.0, которые возвращают защищенную информацию для владельца ресурса, связанного с предоставленным токеном доступа.

7.3.2. *Сервер, обрабатывающий запросы на предоставление доступа к защищенным ресурсам*, определенным в терминах настоящего документа:

1 должен поддерживать положения, определенные в пункте 6.3.2;

2 должен поддерживать механизм подтверждения владения ключом с привязкой токена к MTLS сертификату (см. пункт 5.8.4) или с привязкой токена к TLS-соединению.

7.3.3. *Клиенты*, определенные в терминах настоящего стандарта, должны поддерживать положения, определенные в пункте 6.3.3.

7.4. КОНЕЧНАЯ ТОЧКА ОБЪЕКТА ЗАПРОСА

7.4.1. Вводная информация

В случае, когда клиент не желает отправлять объект запроса (см. подпункт 5.4.2.4) в качестве параметра с передачей его содержимого по значению либо потому что он слишком большой, либо потому что

он содержит конфиденциальную информацию, а клиент не хочет шифровать объект запроса, объект запроса может быть отправлен по ссылке с использованием параметра <request_uri>.

В общем случае <request_uri> может быть либо URL, либо URN.

Хотя URI запроса может размещаться на стороне клиента, в спецификации ФАПИ.СЕК он должен размещаться на сервере авторизации. Преимущество сервера авторизации, на котором размещается объект запроса, состоит в том, что он не должен поддерживать исходящие запросы к определенному клиентом URI запроса и не полагается на энтропию URI для обеспечения конфиденциальности объекта запроса.

В случае если объект запроса хранится на сервере авторизации, значением <request_URI> обычно является URN.

В данном подразделе определяются методы, позволяющие конечной точке объекта запроса сервера авторизации осуществлять обмен объектом запроса на URI запроса.

7.4.2. Запрос

1 Конечной точкой объекта запроса должен быть RESTful API на сервере авторизации, который принимает подписанный объект запроса как значение компоненты <payload> HTTP-запроса.

2 Объект запроса должен подписываться для аутентификации клиента и для подтверждения факта представления клиентом объекта запроса, то есть неотказуемости.

7.4.3. Успешный ответ

1 Сервер авторизации должен проверить действительность объекта запроса и то, что параметр, идентифицирующий алгоритм подписи, не является пустым, а подпись корректна, как представлено в OIDC [11] (подраздел 6.3).

2 Если проверка прошла успешно, сервер должен сгенерировать URI запроса и вернуть в составе <payload> JSON-структуры параметры <request_URI.aud.iss> и <exp> с кодом ответа HTTP “201 Created”.

3 Во избежание его угадывания значение <request_URI> должно быть сгенерировано с использованием СКЗИ. Механизм формирования <request_URI> определяется на этапе разработки сервера авторизации, исходя из его прикладных целей и задач.

4 URI запроса должен быть привязан к идентификатору клиента, который определил объект запроса в виде компоненты <payload> соответствующего запроса.

5 Поскольку URI запроса может быть повторно воспроизведен, рекомендуется сделать срок его службы коротким и однократно используемым.

6 Значения параметров компоненты <payload> JSON-структуры JWT должны быть следующими:
- <request_URI>: URI запроса, соответствующий присланному объекту запроса;
- <aud>: JSON-строка, представляющая идентификатор клиента, который послал объект запроса;
- <iss>: JSON-строка, представляющая идентификатор отправителя (issuer identifier) сервера авторизации. Если используется OAuth 2.0, значением является URI переадресации. При использовании OpenID Connect значением является значение отправителя (issuer value) сервера авторизации;
- <exp>: JSON-число, представляющее время окончания срока жизни URI запроса.

Ниже приведен пример такого ответа:

```
HTTP/1.1 201 Created
Date: Tue, 2 May 2017 15:22:31 GMT
Content-Type: application/JSON

{
    “iss”: “https://as.example.com/”,
    “aud”: “s6BhdRkqt3”,
    “request_URI”: “urn:example:MTAyODAK”,
    «exp»: 1493738581
}
```

7.4.4. Обработка ошибок

7.4.4.1. Требуется авторизация:

– если подтверждение достоверности подписи завершилось неудачей, сервер авторизации должен вернуть HTTP ответ с кодом ошибки “401 Unauthorized”.

7.4.4.2. Недействительный запрос:

– если полученный объект запроса не прошел успешно структурную/синтаксическую валидацию, сервер авторизации должен вернуть HTTP ответ с кодом ошибки “400 Bad Request”.

7.4.4.3. Метод не разрешен:

– если запрос не направлен методом POST, сервер авторизации должен вернуть HTTP ответ с кодом ошибки “405 Method Not Allowed”.

7.4.4.4. Длина запроса слишком велика:

– если размер запроса превышает верхнюю границу, допускаемую сервером авторизации, сервер авторизации должен вернуть HTTP ответ с кодом ошибки “413 Request Entity Too Large”.

7.4.4.5. Слишком много запросов:

– если число запросов от клиента за период времени превышает границу, допускаемую сервером авторизации, сервер авторизации должен вернуть HTTP ответ с кодом ошибки “429 Too Many Requests”.

7.4.5. Метаданные обнаружения поставщика OpenID

Если сервер авторизации имеет конечную точку объекта запроса и поддерживает сервис Discovery (см. подпункт 5.4.4.1), он должен включать в ответы об обнаружении следующий параметр метаданных сервера авторизации:

– <request_object_endpoint>: URL конечной точки объекта запроса, на котором клиент может осуществлять обмен объекта запроса на URI запроса.

8. ЗАЩИЩЕННЫЙ С ИСПОЛЬЗОВАНИЕМ JWT РЕЖИМ ОТВЕТА НА ЗАПРОС АВТОРИЗАЦИИ ДЛЯ OAUTH 2.0 (JARM)

8.1. РЕЖИМ ОТВЕТА НА ОСНОВЕ JWT

В данном разделе стандарта определяется основанный на токенах JWT режим кодирования параметров ответов на запросы авторизации. Все параметры ответов данного типа передаются в JWT вместе со вспомогательными полями, используемыми для дополнительной защиты передачи. Существуют разные способы кодирования самого токена JWT в ответе клиенту, а именно в качестве параметра запроса <URI>, фрагмента полного URI <fragment> либо одного из значений скрипта автоматически обрабатываемой формы <form post>, и в данном разделе определен набор значений режима ответа, соответствующих этим способам кодирования.

Примечание. Дополнительные сведения о кодировании ответа на запрос авторизации OAuth 2.0 представлены в документе [42].

8.1.1. Структура данных JWT-ответа

8.1.1.1. JWT (подраздел 5.7) содержит следующие базовые параметры, используемые для обеспечения безопасности передачи:

– <iss>: URL сервера авторизации, который создал 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. В последующих подразделах иллюстрируется шаблон с типами ответов “code” и “token”.

Примечание. В JWT также добавляются дополнительные параметры ответа конечной точки авторизации, определяемые расширениями, например <session_state> в соответствии с [43].

8.1.1.2. *Тип ответа “code”*

В случае если разрешение на авторизацию имеет тип “code”, JWT содержит параметры <code> и <state>, определенные в подпункте 5.4.2.8.

Следующий пример демонстрирует параметры компоненты <payload> токена JWT для успешного значения <code> ответа на запрос авторизации:

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

В случае ответа об ошибке JWT будет содержать параметры ответа об ошибке, <error>, <error_description>, <error_URI>, <state>, определенные в подпункте 5.4.2.9.

Следующий пример демонстрирует содержимое токена JWT для такого сообщения об ошибке:

```
{
   "error":"access_denied",
   "state":"S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw"
}
```

8.1.1.3. *Тип ответа "token"*

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.

Следующий пример показывает параметры JWT для успешного ответа типа "token" на запрос авторизации:

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

В случае ответа об ошибке JWT содержит параметры ответа об ошибке, как в случае типа ответа "code".

8.1.2. Цифровая подпись и шифрование

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

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 8.2.

Рекомендации по управлению ключами в целом и в особенности по использованию симметричных алгоритмов для подписи и шифрования на основе ключа клиента см. в пункте 5.8.1.

8.1.3. Кодирование ответа

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

- "query.jwt",
- "fragment.jwt",
- "form_post.jwt",
- "jwt".

8.1.3.1. *Режим ответа "query.jwt"* заставляет сервер авторизации отправлять ответ на запрос авторизации в виде HTTP переадресации на URI переадресации (<redirect_URI>) клиента. Сервер авторизация-

ции добавляет параметр <response>, содержащий JWT, как определено в пункте 8.1.1, 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.

8.1.3.2. *Режим ответа "fragment.jwt"* заставляет сервер авторизации 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 8.1.1, к компоненту <fragment> <redirect_URI>, используя формат "application/x-www-form-urlencoded".

8.1.3.3. *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 теле с использованием формата "application/x-www-form-urlencoded".

Примечание. Общие сведения по использованию метода POST для передачи JWT клиенту представлены в документе «OAuth 2.0 Form Post Response Mode» [44].

8.1.3.4. *Режим ответа "jwt"* является ссылкой и указывает на формат включения JWT в URI переадресации по умолчанию (<query> либо <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", – "fragment.jwt".

8.1.4. Правила обработки

Предположение: клиент хранит 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.

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 ключа клиента (см. пункт 8.1.2);

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.

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

Получение клиентом ключей для проверки подписи JWT (шаг 5) должно выполняться в соответствии с алгоритмами подпункта *5.8.1.3*.

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

8.2. МЕТАДАННЫЕ КЛИЕНТА

Названия параметров следуют шаблону, установленному динамической регистрацией клиентов OpenID Connect (см. подпункт 5.4.4.5) для конфигурирования алгоритмов подписи и шифрования JWT ответов в конечной точке UserInfo.

Определены следующие параметры метаданных клиента:

– <authorization_signed_response_alg>: алгоритм цифровой подписи JWS (см. пункт 5.7.1) типа <alg>, который требуется для формирования подписи ответов на запросы авторизации. Если этот параметр определен, ответ будет подписан в соответствии с требованиями JWS (см. пункт 5.7.1) и сконфигурированного алгоритма. Алгоритм "none" не допускается;
– <authorization_encrypted_response_alg>: алгоритм шифрования JWE (см. пункт 5.7.2) 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 и <authorization_encrypted_response_alg>.

В процессе регистрации на сервере авторизации клиенты могут давать последнему ссылки на ресурсы хранения своих открытых ключей, используя параметры метаданных <jwks_URI> или <jwks> (см. подпункт 5.4.4.5).

8.3. МЕТАДАННЫЕ СЕРВЕРА АВТОРИЗАЦИИ

Серверам авторизации рекомендуется публиковать типы поддерживаемых алгоритмов формирования цифровой подписи и шифрования JWT ответа на запрос авторизации, используя параметры метаданных сервера авторизации в соответствии с пунктом 5.4.4.1.

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.

Серверам авторизации рекомендуется публиковать значения поддерживаемых режимов ответа с использованием параметра <response_modes_supported> в соответствии с подпунктом 5.4.4.1. Настоящий стандарт представляет следующие возможные значения этого параметра:

– "query.jwt",
– "fragment.jwt",
– "form_post.jwt",
– "jwt".

БИБЛИОГРАФИЯ

[1] ГОСТ Р ИСО/МЭК 27000–2012 «Информационные технологии. Методы обеспечения безопасности. Менеджмент информационной безопасности. Общий обзор и терминология».

[2] ГОСТ Р 58833–2020 «Защита информации. Идентификация и аутентификация. Общие положения».

[3] Р 50.1.053–2005 «Информационные технологии. Основные термины и определения в области технической защиты информации».

[4] ГОСТ Р 57580.1–2017 «Безопасность финансовых (банковских) операций. Защита информации финансовых организаций. Базовый состав организационных и технических мер».

[5] ГОСТ Р 34.12–2015 «Информационная технология. Криптографическая защита информации. Блочные шифры».

[6] ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи».

[7] Словарь криптографических терминов. Под ред. Б.А. Погорелова и В.Н. Сачкова. – М.: МЦНМО, 2006. – 94 с.

[8] Р 50.1.113–2016 «Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования».

[9] Р 50.1.041–2002 «Информационные технологии. Руководство по проектированию профилей среды открытой системы (СОС) организации-пользователя».

[10] Hardt D. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012. https://tools.ietf.org/html/rfc6749 (дата обращения: 11.06.2020).

[11] 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 (дата обращения: 11.06.2020).

[12] JSON Web Token (JWT). RFC 7519. https://tools.ietf.org/html/rfc7519 (дата обращения: 11.06.2020).

[13] Jones M., Hardt D. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, October 2012. https://tools.ietf.org/html/rfc6750 (дата обращения: 11.06.2020).

[14] The Base16, Base32, and Base64 Data Encodings. RFC 4648. https://tools.ietf.org/html/rfc4648 (дата обращения: 11.06.2020).

[15] 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-ID2.html (дата обращения: 11.06.2020).

[16] Financial-grade API – Part 2: Read & Write API Security Profile (Implementer's Draft). OpenID Foundation, Financial-grade API (FAPI) WG, 2019. https://openid.net/specs/openid-financial-api-part-2-ID2.html (дата обращения: 11.06.2020).

[17] 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 (дата обращения: 11.06.2020).

[18] Федеральный закон от 27.07.2006 № 152-ФЗ «О персональных данных».

[19] Постановление Правительства Российской Федерации от 01.11.2012 № 1119 «Об утверждении требований к защите персональных данных при их обработке в информационных системах персональных данных».

[20] Приказ ФСБ России от 10.07.2014 № 378 «Об утверждении Состава и содержания организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных с использованием средств криптографической защиты информации, необходимых для выполнения установленных Правительством Российской Федерации требований к защите персональных данных для каждого из уровней защищенности».

[21] Приказ ФСТЭК России от 18.02.2013 № 21 «Об утверждении Состава и содержания организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных».

[22] Положение Банка России от 09.06.2012 № 382-П «О требованиях к обеспечению защиты информации при осуществлении переводов денежных средств и о порядке осуществления Банком России контроля за соблюдением требований к обеспечению защиты информации при осуществлении переводов денежных средств».
[23] Положение Банка России от 17.04.2019 № 683-П «Об установлении обязательных для кредитных организаций требований к обеспечению защиты информации при осуществлении банковской деятельности в целях противодействия осуществлению переводов денежных средств без согласия клиента».
[24] Положение Банка России от 17.04.2019 № 684-П «Об установлении обязательных для некредитных финансовых организаций требований к обеспечению защиты информации при осуществлении деятельности в сфере финансовых рынков в целях противодействия осуществлению незаконных финансовых операций».
[25] Jones P., Salgueiro G., Jones M., J. Smarr. WebFinger. RFC 7033. September 2013. https://tools.ietf.org/html/rfc7033 (дата обращения: 11.06.2020).
[26] 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 (дата обращения: 11.06.2020).
[27] OAuth 2.0 Authorization Server Metadata. RFC 8414. https://tools.ietf.org/html/rfc8414 (дата обращения: 11.06.2020).
[28] 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 (дата обращения: 11.06.2020).
[29] OAuth 2.0 Dynamic Client Registration Protocol. RFC 7591. https://tools.ietf.org/html/RFC7591 (дата обращения: 11.06.2020).
[30] Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7521. May 2015. https://tools.ietf.org/html/rfc7521 (дата обращения: 11.06.2020).
[31]JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants. RFC 7523. May 2015. https://tools.ietf.org/html/rfc7523 (дата обращения: 11.06.2020).
[32]Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. May 2008. https://tools.ietf.org/html/rfc5280.
[33] HTTP Authentication: Basic and Digest Access Authentication. RFC 2617. June 1999. https://tools.ietf.org/html/rfc2617 (дата обращения: 11.06.2020).
[34] JSON Web Signature (JWS). RFC 7515. https://tools.ietf.org/html/rfc7515 (дата обращения: 11.06.2020).
[35] JSON Web Encryption (JWE). RFC 7516. https://tools.ietf.org/html/rfc7516 (дата обращения: 11.06.2020).
[36] JSON Web Key (JWK). RFC 7517. https://tools.ietf.org/html/rfc7517 (дата обращения: 11.06.2020).
[37] МР 26.2.001–2013 «Использование наборов алгоритмов шифрования на основе ГОСТ 28147–89 для протокола безопасности транспортного уровня (TLS). Методические рекомендации технического комитета ТК26».
[38] OAuth 2.0 Token Introspection. RFC 7662.
[39] OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens. RFC 8705. https://tools.ietf.org/html/rfc8705 (дата обращения: 11.06.2020).
[40] A UniveRSAlly Unique IDentifier (UUID) URN Namespace. RFC 4122. https://tools.ietf.org/html/rfc4122 (дата обращения: 11.06.2020).
[41] OAuth 2.0 Token Binding. https://tools.ietf.org/html/draft-ietf-OAuth-token-binding (дата обращения: 11.06.2020).
[42] OAuth 2.0 Multiple Response Type Encoding Practices. https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html (дата обращения: 11.06.2020).
[43] OpenID Connect Session Management 1.0: https://openid.net/specs/openid-connect-session-1_0.html (дата обращения: 11.06.2020).
[44] OAuth 2.0 Form Post Response Mode: https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html (дата обращения: 11.06.2020)

УДК 681.3.06

Ключевые слова: финансовая технология, открытые программные интерфейсы, токен доступа, авторизация