FINANCIAL™ DATA EXCHANGE

API Security Model

Specification for Consumer Financial Account Aggregation Services

Version 4.2 November 2024

FDX Security and Authentication Working Group

Legal Notice

Financial Data Exchange, LLC (FDX) is a non-profit organization operating in the U.S. and Canada that is dedicated to unifying the financial industry around a common, interoperable, royalty-free standard for secure and convenient consumer and business access to their financial data. FDX empowers users through its commitment to the development, growth, and industry-wide adoption of the FDX API, according to the principles of control, access, transparency, traceability, and security. Many of the terms set forth herein are subject to additional interpretations under prevailing laws, industry norms, and/or governmental regulations. While referencing certain laws that may be applicable, readers, users, members, or any other parties should seek legal advice of counsel relating to their particular practices and applicable laws in the jurisdictions where they do business. See FDX's complete Legal Disclaimer located at http://www.financialdataexchange.org/ for other applicable disclaimers.

Membership is open to financial institutions, fintech companies, financial data aggregators, consumer advocacy groups, payment networks and other industry stakeholders. For more information and to join, visit http://www.financialdataexchange.org/.

Revision History

Document Version	Notes	Date
4.2	Minor formatting updates	**November 2024**
4.1	RFC 0237 FDX API Security Profile 1.0 has been included in Part 1 instead of referenced. RFC 0289 non-material document updates have been incorporated.	**September 2024**
4.0	RFC 0011 has been incorporated as Part 4 End to End Encryption. This was formerly published as section 4 of the FDX API Specification document though its version 6.0.	**December 2023**
3.5	Updates for recipient registration guidelines, automation, and delegation – Part 2.	**May 2022**
3.4	Updates for FAPI Security Profile 1.0 – Part 1 and Part 2.	**October 2021**
3.3	Includes FDX RFC 0117 FDX OAuth Extensions for identifying intermediary entities and FDX RFC 0115 Guidelines and Best Practices for Sharing Sensitive Data Securely.	**May 2021**
3.2	Initial Document Release	**December 2020**

Contents

This page intentionally left blank.

FDX API Security Model Profile Standard Specification

This FDX API Security Model is a companion document to FDX Control Considerations. This document contains specifications adopted by FDX for secure end-user permissioning for financial data sharing, identification of intermediaries, and maintaining security of data in transit.

Part 1 specifies the FDX API security profile.

Part 2 specifies FDX Recipient Registration profiles with three sub-sections. Part A details methods for identifying intermediaries in the data sharing chain between Data Recipient and Data Provider. Part B describes Recipient Registration Automation while Part C details Recipient Registration With Delegation to an Ecosystem Registry.

Part 3 specifies best practices for securing sensitive data in transit.

Part 4 specifies end to end encryption.

Part 1: FDX API Security Profile

FDX **SHALL** adopt the following FAPI specifications:

- Financial-grade API Security Profile 1.0 - Part 2: Advanced - for securing traffic to APIs

  Note: This references and builds upon the previously FDX-adopted Financial-grade API Security Profile 1.0 - Part 1: Baseline.

- OpenID Connect Client Initiated Backchannel Authentication Flow - Core 1.0 draft-03 - for authentication of end users

Further, FDX **SHALL** adopt the following certification profiles:

- FAPI Conformance Profile
- FAPI CIBA Profiles

FDX **SHALL NOT** adopt the OIDC Certification Profiles as it contains mandatory tests for items such as `client_secret_basic` authentication that are explicitly disallowed by FAPI Part 2: Advanced. Hence, a production data provider system **cannot** pass both FAPI and OIDC certifications.

These certification profiles **SHALL** be enforced in a time and manner as determined by the FDX Certification Working Group.

All three specifications, Financial-grade API Security Profile 1.0 Part 2: Advanced, Part 1: Baseline and OpenID Connect CIBA Flow - Core 1.0 may be viewed by clicking on those links, or can be found in the exhibits at the end of this document.

Introduction

As the FDX API standards for open finance data sharing have advanced to define over 660 financial data elements, FDX has recognized a need for API security profiles specific to the individual FDX API endpoints. In specifying these security profiles, FDX endeavors to make use of recognized industry security standards as practical and suitable for the needs of the FDX membership and its community of customers.

FDX API Security Profile 1.0 focuses on the security of authentication and authorization, principally but not exclusively through securing OAuth 2.0 and OIDC flows as replacements for password sharing and screen-scraping. Additional important security considerations are addressed in other FDX publications such as the FDX Security Model and FDX Control Considerations.

Notational Conventions

The keywords "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**MAY**", and "**CAN**" in this document **SHALL** only be interpreted as described in ISO Directive Part 2. These keywords **SHALL NOT** be interpreted with their natural language meanings.

Scope

This standard is strongly recommended for FDX APIs Specification published in FDX API v6.1 Spring Release 2024. Conformance with this specification is not required for conformance with the FDX API Specification. To be conformant with this specification, implementers **MUST** conform with all required and strongly recommended clauses.

Any subsequent version of the FDX APIs Specification may come either with the same or updated version of this FDX API Security Profile.

Any modification to the FDX API Security Profile strongly recommended for any published FDX APIs Specification will be issued as addenda or errata.

Implementers **SHOULD** comply with applicable laws and regulations in their jurisdictions.

Normative References

- FDX Taxonomy v1.5
- FDX API Specification 6.1
- FDX RFC 0011: Message Encryption
- FDX RFC 0188: Event Notifications
- FDX RFC 0156: Consent API: Request, Issue, and Retrieval Operations and Data Structures
- FDX RFC 0223 Control Considerations for Step Up Authentication
- FDX RFC 0257 Consent Grant details ↔ OAuth Scopes alignment
- BCP 195

- https://www.rfc-editor.org/info/rfc7525
- FAPI 1.0 Part One: Baseline Security Profile
- FAPI 1.0 Part Two: Advanced Security Profile
- https://datatracker.ietf.org/doc/html/rfc6749
- https://datatracker.ietf.org/doc/html/rfc7009
- https://datatracker.ietf.org/doc/html/rfc5246
- https://datatracker.ietf.org/doc/html/rfc8446
- https://datatracker.ietf.org/doc/html/rfc8725
- https://datatracker.ietf.org/doc/html/rfc9126
- https://www.rfc-editor.org/rfc/rfc8705
- OIDC Core Errata 2 for Pairwise Pseudonymous Identifiers
- NIST 800-63

Terms and definitions

FAPI: FAPI was previously known as the Financial-grade API. The name was updated to FAPI to reflect that the specification is intended for other APIs other than those belonging to financial services.

Symbols and abbreviated terms

FDX API Security Profile uses terms defined in FDX Taxonomy 1.5. In addition to that, the following terms are used:

- FDX Security Profile: This FDX API Security Profile
- FAPI 1.0 Advanced: FAPI 1.0 Part Two: Advanced Security Profile
- FAPI 1.0 Baseline: FAPI 1.0 Part One: Baseline Security Profile
- FAPI: The family of FAPI Security Profiles
- FDX Step Up Auth: Step up authentication as defined in FDX RFC 0223 (RFC 0222 for applicable APIs).
- FDX Message Encryption: Message encryption as defined in FDX RFC 0011 Message Encryption (encompassed in *Part 4* of this document *End to End Encryption*)
- FAL2: Federated Assurance Level 2
- FAL3: Federated Assurance Level 3
- JWT: JSON Web Token
- MTLS: Mutual Transport Layer Security
- PAR: Pushed Authorization Requests
- RAR: Rich Authorization Requests

Updates to other FDX RFCs and Conflicts

RFC 0122: FDX Adoption of FAPI is modified such that the requirement for conformance to FAPI 1.0 Advanced applies to all FDX API endpoints, apart from those explicitly specified as not requiring FAPI 1.0 Advanced. For avoidance of doubt, any API that is not covered in this FDX API Security Profile is strongly recommended to conform with FAPI 1.0 Advanced and any optional security standards explicitly defined for such APIs in FDX Security Profile.

In case of any conflict between this FDX API Security Profile and any endpoint RFCs, the FDX API Security Profile **SHALL** take precedence.

FDX API Security Profile

This section defines the general requirements of the FDX API Security Profile.

API Specific Profiles

FDX Security Profile defines security requirements for FDX APIs. It defines general requirements and prescribes distinct requirements for specific FDX APIs.

FDX Security Profile primarily leverages FAPI as the API security profile and assigns specific versions of FAPI to distinct FDX APIs. FDX Security Profile constrains certain FAPI provisions and may override them.

In addition to FAPI Security Profiles, FDX Security Profile adds requirements for additional security standards such as FDX RFC 0223 Control Considerations for Step Up Authentication.

Technical Scope

FDX Security Profile:

- Defines requirements for authorization servers and resource servers
- Defines requirements for application clients that connect to and consume FDX APIs.

Exceptions and Special Considerations

Most of the FDX APIs are used for financial data sharing, thus, with certain exceptions as noted below, all FDX APIs exposed by implementers are strongly recommended to comply with the primary security profile which is FAPI 1.0 Advanced. In addition, there are certain FDX APIs that require special consideration for which certain specifications of the FDX API Security Profile are modified.

Versioning

The FDX API Security Profile corresponds to and is strongly recommended for a specific version of the FDX specification. Any subsequent version of the FDX specification may require an update of the FDX API Security Profile. If required, any changes made to the FDX API Security Profile will be issued as addenda or errata.

Conformance Certification

Procedures and timelines for certification of conformance with the FDX API Security Profile are out of the scope for this standard and are to be addressed by the FDX certification program.

Requirements for Parties that Expose FDX APIs

This section contains security requirements for implementers that expose the following FDX APIs that require FAPI 1.0 Advanced:

- Consent
- Core
- Customer
- Fraud
- Money Movement
- Recipient Registration
- Registry
- Tax.

Authorization Server

1. Authorization server **SHALL** reject authentication or authorization requests from requestors not in Approved or Tentative status. (Reference: FDX RFC 0117 regarding extension for status of authorization requestor. RFC 0117 - FDX OAuth Extensions)
2. The authorization server **SHALL** comply with the requirements set in FAPI 1.0 Advanced, subject to the following provisions:
    a. Whenever FAPI mentions PAR as an option it **SHALL** be required and the only possible option. Furthermore:
        i. Authorization requests not involving PAR that are allowed per FAPI 1.0 Advanced **SHALL** be rejected by the authorization server as stated in the sec. 5.3.1.2 par. 2 of the FAPI 2.0
        ii. Authorization server **SHALL** comply with other provisions related to PAR defined in the section 5.3.1 of the FAPI 2.0
    b. Out of response type and response mode options that FAPI allows, the authorization server **SHALL** only allow and require use of the response_type code and response_mode JWT as mentioned in the section 5.2.2 par. 2.2 of the FAPI 1.0 Advanced
    c. The maximum lifetime from issuance to token retrieval of authorization codes **SHALL** be 300 seconds.
3. Authorization server **MUST** expose the OAuth2 Token Revocation Endpoint.
4. Access tokens **SHALL** expire after no more than 900 seconds.

Resource Server

1. Shall comply with FAPI 1.0 Advanced
2. Any requests to Data Provider's APIs that share data owned by customers **MUST** be authenticated with the access_token issued as a result of the authorization flow that is part of the customer consent flow.
3. With each request the resource server **SHALL** verify whether:
    a. The access_token has an OAuth scope required to call the API (FDX RFC 0257),

b. The consent record associated with the access_token entitles the requestor to perform the action on the target resource.

Requirements for Parties that Consume FDX APIs

This section contains security requirements for parties which consume FDX APIs, i.e. Data Recipients or Data Access Platforms.

Application Client

Client applications that consume FDX APIs for which FAPI is strongly recommended **SHALL** comply with client provisions included in this FDX API Security Profile.

Clients that consume notifications **SHALL** comply with requirements set in this FDX Security Profile and in the FDX notifications specification.

API Specific Requirements and Considerations

The table below assigns strongly recommended FAPI versions for specific FDX APIs and defines additional, optional security requirements.

API	Required for Conformance with this Specification	Optional for Conformance with this Specification
Consent	FAPI 1.0 Advanced RAR (required by consent request flow)	
Core	FAPI 1.0 Advanced	FDX RFC 0223 Control Considerations for Step Up Authentication FDX RFC 0011 Message Encryption (Encompassed in Part 4 End to End Encryption)
Customer	FAPI 1.0 Advanced	
Notifications Publishing	MTLS	
Notifications Get	FAPI 1.0 Baseline	FAPI 1.0 Advanced

API	Required for Conformance with this Specification	Optional for Conformance with this Specification
Notifications Subscription	FAPI 1.0 Baseline	FAPI 1.0 Advanced
Fraud	FAPI 1.0 Advanced	FDX Message Encryption
Meta	FAPI 1.0 Baseline	FAPI 1.0 Advanced
Money Movement	FAPI 1.0 Advanced	FDX RFC 0223 Control Considerations for Step Up Authentication FDX RFC 0011 Message Encryption (Refer to Part 4 End to End Encryption)
Recipient Registration	Security profile defined in RFC 0153	FAPI 1.0 Advanced
Registry	FAPI 1.0 Baseline and RFC 9126 PAR	
Tax	FAPI 1.0 Advanced	FDX RFC 0223 Control Considerations for Step Up Authentication FDX RFC 0011 Message Encryption (Refer to Part 4 End to End Encryption)

Security Requirements for Specific APIs

Consent APIs

Consent Request Flow Security Requirements

1. FDX consent request flow requires use of the pushed authorization request (PAR), which is optional per FAPI 1.0 Advanced. Authorization servers **SHALL** require PAR.
2. Authorization servers **SHALL** verify whether scopes included in the authorization request match aggregated total data clusters that have been sent in the consent

request object. If there are fewer scopes sent or there are any extra scopes included the authorization request the request **SHALL** fail.

Consent Grant Revocation

If a consent grant is revoked, then the refresh token and access token associated with the consent grant **SHALL** be revoked immediately.

Core API

1. Agreements between Data Provider, Data Access Platform and Data Recipient **SHALL** govern the optional, additional implementation of FDX RFC 0011 Message Encryption.
2. Agreements between Data Provider, Data Access Platform and Data Recipient **SHALL** govern the optional, additional implementation of FDX RFC 0223 Control Considerations for Step-up Authentication.

Money Movement API

1. Agreements between Data Provider, Data Access Platform and Data Recipient **SHALL** govern the optional, additional implementation of FDX RFC 0011 Message Encryption.
2. Agreements between Data Provider, Data Access Platform and Data Recipient **SHALL** govern the optional, additional implementation of FDX RFC 0223 Control Considerations for Step-up Authentication. The use of step up authentication is strongly recommended**.**

Notifications - Publishing

Notification publishing requires MTLS authentication and does not require OAuth2 or FAPI.

Notifications Get API

FAPI 1.0 Part One: Baseline is required. Implementers may optionally use FAPI 1.0 Part Two: Advanced instead.

Notifications Subscription

FAPI 1.0 Part One: Baseline is required. Implementers may optionally use FAPI 1.0 Part Two: Advanced instead.

Recipient Registration API

Recipient registration described in FDX RFC 0153 defines its own security profile. Implementers can use the current security profile defined in FDX RFC 0153 or implementers may optionally use FAPI 1.0 Part Two: Advanced.

Tax API

1. Implementations may optionally use FDX RFC 0011: Message Encryption.
2. Use of FDX RFC 0223 Control Considerations for Step-up Authentication is optional and strongly recommended.

Cryptography

When creating and processing JWTs, authorization servers, clients and resource servers **SHALL**:

1. Comply with RFC 8725
2. Use one of the following algorithms for JWT signing:
    a. RSASSA-PSS using SHA-256 and MGF1 with SHA-256 (PS256), as defined in RFC 7518
    b. ECDSA using curve P-256 and SHA-256 (ES256), as defined in RFC 7518
    c. EdDSA with Ed25519 curve, as defined in RFC 8032 https://datatracker.ietf.org/doc/html/rfc8032#section-5.1
3. Not use or accept the none algorithm
4. Use keys with minimum key length of 160 bits when using elliptic curve encryption algorithms.

Key Management

Unless otherwise noted, key management practices including but not limited to certification authority practices, key usage, key storage, crypto-period, auditing, key destruction, key rotation, key recovery and key compromise **SHALL** comply with the current approved versions of NIST Special Publication 800-57 Parts 1 and 2.

Security Considerations for Federated Identity

Moderate risk endpoints

Implementers of moderate risk endpoints utilizing a system of identity federation **SHOULD** adhere to FAL2 or FAL3 as specified in NIST 800-63.

High risk endpoints

Implementers of high risk endpoints utilizing a system of identity federation **SHOULD** adhere to FAL3 as specified in NIST 800-63.

Privacy Considerations

Pairwise pseudonymous identifiers

To protect consumers from identity correlation and linking among implementers, the use of pairwise pseudonymous identifiers **SHOULD** be used in accordance with OpenID Core Errata 2 section 8 and NIST 800-63.

Data minimization

To uphold the data minimization principle, implementers **SHOULD** only share and collect the minimum data needed to fulfill the end user's purpose.

Part 2: FDX Recipient Registration Profiles

Part A. Identifying Intermediaries Between Data Recipient and Data Provider Using OAuth

Introduction

Intermediaries such as Data Access Platforms (aka aggregators) play an important role in financial data sharing ecosystem. Software applications (aka Data Recipients) used by end-users to manage their finances or provision financial services, leverage Data Access Platforms to connect to thousands of financial institutions. The data sharing thus involves at least three parties - Data Recipient, Data Access Platform and Data Provider (e.g., Financial Institutions.) Sometime there may be more than one intermediary, for example, when one DAP leverages another DAP to gain access to financial institutions in another market for expanded coverage.

This specification supports Intermediary identification such that:

1. The Data Provider can register and issue client credentials to the intermediary chain to support the auditability of data processing (i.e., apply applicable local & national data privacy laws for Data Provider action, if/where needed).

2. The End User (Consumer) will have awareness of the parties who have access to their data and can provide consent based upon the identity of the intermediaries, facilitating future actions if required or desired.

Specification

1. FDX lends support to Aaron Parecki's IETF OAuth 2.0 Client Intermediary Metadata draft-03. This specification defines a mechanism for including information about additional parties involved in an OAuth transaction. It defines *"Intermediary" as "One or more entities that the user's data will pass through or be shared with by using the OAuth client. This information is voluntarily provided by the OAuth client, and is typically enforced by a business relationship between the organization providing the Client and the organization providing the Resource Server."*

2. Intermediary including "core-providers" **SHOULD** list themselves as part of the Intermediary chain.

3. If an Intermediary is added/removed the whole chain **MUST** be re-registered and a new client-id/secret issued.

4. The Client **SHOULD** register with the "Scope" as specified in RFC7591.

5. Proposed extension to the OAuth 2.0 Protocol RFC7591

    a. Required for FDX Certification

        i. Adds a “description” field to the Client Metadata fields defined by RFC7591

        ii. In some instance the Data Provider needs time to conduct due-diligence on the Clients being registered. An extension “status”: “Approved” | “Tentative” | “Pending” | “Rejected” is being proposed to enable bulk client registration and yet allow time to conduct due-diligence. The Data Provider will issue client_id/client_secret and set the “status” as appropriate.

        iii. “Approved” states that Data Recipient can use their client credentials to initiate authorization consent flow.

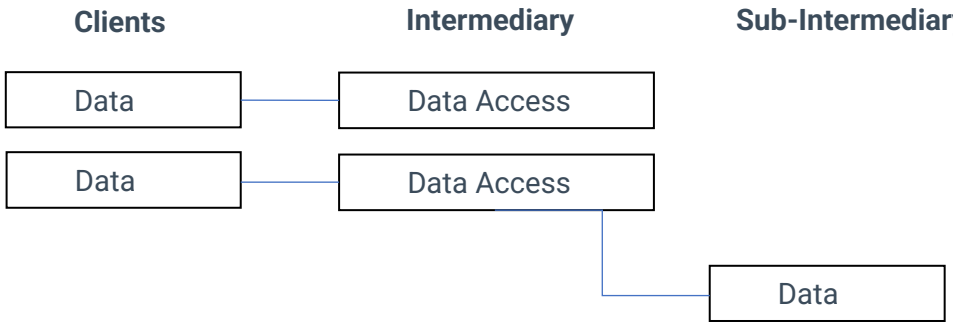
        iv. “Pending” states that Data Provider is conducting due diligence and upon completion will update the status.

        v. “Rejected” states that Data Provider has denied access to the Data Recipient.

        vi. “Tentative” states that Data Recipient can use their client credentials to initiate authorization consent flow until they are either “Approved” or “Rejected”.

    b. Not Required for FDX Certification

        i. Where an Intermediary has Sub-intermediaries, they **SHOULD** be listed as a child of an Intermediary. A Sub-intermediary is an entity to which the Intermediary sends end-user’s data to for various reasons. Sub-intermediary metadata follows the same data structure as intermediaries. Implement Sub-intermediary as an extension.

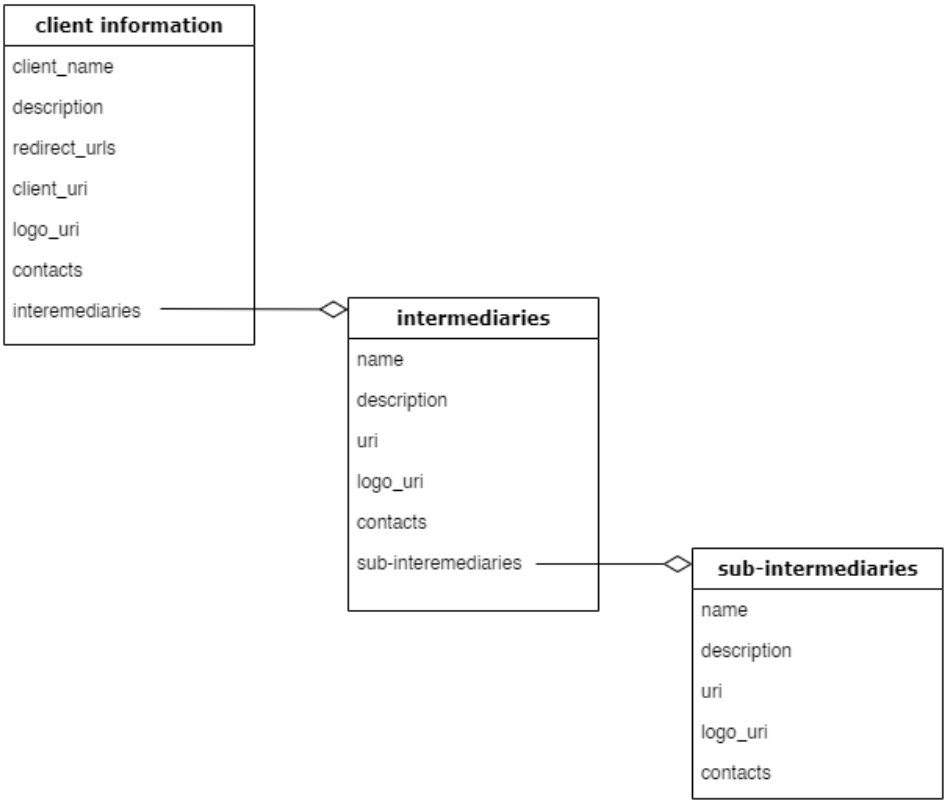


Example

The following is a non-normative example of inclusion of an intermediary chain in an OAuth request.

```
{

  "client_name": "Client App Name",

  "description": "User is interacting with this application in
their browser",

  "redirect_uris": ["https://client.example.org/callback"],

  "client_uri": ["https://example.net/"],

  "logo_uri": ["https://example.net/logo.png"],

  "contacts": ["support@example.net"],

  "intermediaries": [

    {

      "name": "Inter Partner App One",

      "description":"Intermediary that may also receive this
user's Account, Transaction, Statements data when the user
authorizes the Client",

      "uri": ["https://partner-one.example/"],

      "logo_uri": ["https://partner-one.example/logo.png"],

      "contacts": ["support@partner-one.example"],

      "sub_intermediaries": [

        {

          "name": "Sub Partner",

          "description": "Entity that receives user's data from
Intermediary",

          "uri": ["https://sub-partner-one.example/"],

          "logo_uri": ["https://sub-partner-
one.example/logo.png"],

          "contacts": ["support@sub-partner-one.example"]
```

```
        },
        {
          "name": "Inter Partner App Two",
          "description": "Sub-Intermediary that may receive this
user's data when the user authorizes the Client",
          "uri": ["https://partner-two.example/"],
          "logo_uri": ["https://partner-two.example/logo.png"],
          "contacts": ["support@partner-two.example"]
        }
      ]
    }
  ]
}
```



Reference:

- RFC 0117 - FDX OAuth Extensions (FDX members only. Content abstracted into section above)
- IETF OAuth 2.0 Client Intermediary Metadata draft-03

Part B. Recipient Registration Automation

Refer to the FDX Recipient Registration Guidelines document for details.

Automation and Use Cases

Refer to the Dynamic Client Registration Automation Use Cases section of the *FDX Recipient Registration Guidelines* document for additional details.

1. Providers **MUST** implement a Dynamic Client Registration API to allow for Automation in the Recipient Registration process.
2. Data Providers **MAY** implement an immediate approval and transmission, or **MAY** Deploy an offline approval process for Recipient application registrations.
3. Data Providers who require an offline approval process for some types of updates, **MUST** include the following capabilities in their DCR implementation:
    1. Provide a Status in the Dynamic Client Registration response
    2. Implement the Dynamic Client Registration read (GET) capability (RFC 7592)
4. Data Providers **MUST** implement DCR update capabilities (PUT).
5. Data Providers **SHOULD** allow simple Recipient and Intermediary metadata changes to be made immediately with no approvals.
6. Data Providers **MUST** implement a DELETE capability as part of their DCR implementation. Upon deletion of a Data Recipient a Provider **SHOULD** immediately revoke all consumer tokens associated with the deleted Client ID.

Interoperability and Consistency

Refer to the End-to-End Recipient Onboarding Process section of the *FDX Recipient Registration Guidelines* document for additional details.

- Provider Client Registration API end point implementations **MUST** adhere to the *OAuth 2.0 Dynamic Client Registration* IETF standards in RFC 7591 and RFC 7592.
- Provider DCR implementations **SHOULD** also follow *FDX OAuth Extensions* defined in **RFC 0117** and the guidelines defined in this document.

Compliance

Refer to the Dynamic Client Registration Standard and the Recipients With Multiple Applications or Use Cases sections of the *FDX Recipient Registration Guidelines* document for additional details.

The registering party for a Data Recipient **MAY** be a Direct Data Recipient who will register themselves or a Data Access Platform who will register Data Recipients who rely on their services for consumer permissioned data sharing.

- Registering party **MUST** collect the Recipient and intermediary metadata
- Registering party **MUST** ensure accuracy for all recipient and intermediary fields. If there is a question about how a recipient is displayed (Name, Logo), the Registering party **MUST** work with the Data Recipient to obtain the correct information.
- Data Recipients and Intermediaries **MUST** be easy to identify based on clear names, logos and URIs that can be looked up on the internet.
- Data Recipient metadata **MUST** be common for all Provider Registrations
- A Data Recipient **MAY** choose to service all Permissioned Data Sharing use cases through a single custom consent to a Provider, which subsequently requires a single Client ID Registration through DCR.
- If a Data Recipient has a single consent to cover all Data Recipient Application use cases, and the Data Provider requires scope during Registration, then the Client ID Registration (serviced by a Data Access Platform or Direct Data Recipient) **MUST** include Scope which covers all use cases.
- A Data Recipient **MAY** choose to require a separate consumer consent action for two or more separate applications, which subsequently requires a separate Client ID Registration for each Recipient Application.
- If a Data Recipient has multiple applications that each require a separate consent, and the Data Provider requires scope during Registration, then the Client ID Registration (serviced by a Data Access Platform or Direct Data Recipient) **MUST** register for the scope required for each separate Client registration.
- If a Data Recipient has multiple applications that each require a separate Client ID Registration, then the client_name and description **MUST** be different for each registration and **MUST** be human-readable and clear about the Data Recipient application use case to which the consumer is consenting to share their Provider data.
- If a Data Recipient utilizes multiple Data Access Platforms for consumer permissioned data sharing services, the Data Access Platforms **MAY** register the Data Recipient with the same client_name.
- Data Providers **MUST** be able to support multiple Data Recipient registrations with the same client_name. If the same client_name is registered by multiple Data Access

Platforms for the same Data Recipient, then the Data Provider **MUST** respond with a separate Client ID for each registration.

Recipient Registration Metadata

Refer to the Recipient Registration Metadata section of the *FDX Recipient Registration Guidelines* document for additional details.

FDX RFC 0153 Introduced the following new fields:

- duration_type[] - Optional
    - The duration of consent for the Data Recipient consumers. Options include:
        - persistent
        - time_based
        - one_time

This is an array for Providers that offer duration optionality during consumer authorization and consent.

- duration_period - Optional
    - The number of days for which consent is granted. Duration period is required when duration_type=time_based.
- lookback_period - Optional
    - The maximum number of days allowed for Data Recipient consumers to obtain in transaction history, effective from the current date.
- registry_references[] - Optional
    - An array of external registries containing registered_entity_name, registered_entity_id and registry fields for the registries where the data recipient or intermediary is registered. If one or more registry entries is included, then all three fields are required to create a valid array object.
    - registered_entity_name - Optional
        - The legal company name for the data recipient or intermediary in the identity chain. This may be the same as the client_name for a data recipient. In many cases this could be a parent company name or the entity name with Corporation or LLC
    - registered_entity_id - Optional
        - An ID representing the company that can be looked up from a legal identity registry source
    - registry - Optional

- The Registry source. Enum values include [PRIVATE, FDX, GLEIF, ICANN]

DCR Response Metadata Update

Refer to the Response Metadata section of the *FDX Recipient Registration Guidelines* document for additional details.

status - New valid value added

- Inactive - Registered but inactive Client ID

Security

Refer to the Securing Application Registration section of the *FDX Recipient Registration Guidelines* document for additional details.

1. Data Recipient registration **MUST** be secured with a method that ensures the Recipient Registration request is from a known Partner (Data Access Platform or Direct Recipient).
2. Data Providers **MUST** secure the Dynamic Client Registration end point with OAuth 2.0 authentication as defined in RFC 6749 (Access Token).
3. Data Providers **MUST** adhere to FAPI Authorization server requirements for Partner Client Authentication and sender-constrained access tokens as stated in section 5.2.2 #5, #6, and #14 (FAPI Part 2 Authorization Server).

Partners **MUST** use and Data Providers **MUST** enforce the practices in RFC 8705 to bind the Credential (Bearer Token) to the secure channel during DCR call execution.

Part C. Recipient Registration With Delegation to an Ecosystem Registry

Refer to the *FDX Recipient Registration with Delegation* document for full implementation detail.

Data Recipient Registration Metadata

The Data Recipient will register themselves with the Recipient Registry. Data Access Platforms or Direct Data Recipients will inform the Data Provider of the Recipient Registry and Data Recipient identifier at the Recipient Registry. Data Providers will request Data Recipient Metadata from the Recipient Registry.

Joint responsibilities:

- Data Access Platforms or Direct Data Recipients, and Data Providers **MUST** agree on a Recipient Registry and Recipient Registry identifier (i.e., “registry” field passed with OAuth authorization request information).

Data Provider responsibilities:

- Data Providers **MUST** have an established relationship with the Recipient Registry that allows them to request Data Recipient information from the Recipient Registry.

Data Access Platform responsibilities (if involved):

- Data Access Platforms **MUST** be able to identify Data Recipients at agreed upon Recipient Registries for Data Recipients who rely on their services.
- Data Access Platforms **MUST** collect intermediary information.

Direct Data Recipient responsibilities (if involved):

- Direct Data Recipients **MUST** be able to identify themselves at the agreed upon Recipient Registry.

Recipient Registry responsibilities:

- Recipient Registry **MUST** collect the Data Recipient metadata.
- Recipient Registry **MUST** ensure accuracy for all recipient fields. If there is a question about how a recipient is displayed (Name, Logo), the Recipient Registry **MUST** work with the Data Recipient to obtain the correct information.
- Data Recipients **SHOULD** be easy to identify based on clear names, logos and URIs that can be looked up on the internet.

Recipient Registration

This mechanism by which Data Recipients register themselves with Recipient Registries is out of the scope of this proposal. Recipient Registries and the Recipients they register **MUST** define and agree to this themselves.

Recipient Registry APIs

Recipient Registries **MUST** host two endpoints for bulk recipient information retrieval ("/recipients") and individual recipient information retrieval ("/recipient/<recipient_id>").

- GET "/recipients" will return the metadata for Data Recipients registered at the Recipient Registry (i.e., a list of "Recipient Metadata" objects under the "recipients" key). The endpoint is paginated as described in the REST best practics page of our member Confluence site.
- GET "/recipient/<recipient_id>" takes a recipient_id parameter that identifies a specific Data Recipient registered at the Recipient Registry and returns the Data Recipient metadata (i.e., a single "Recipient Metadata" object), or returns a HTTP 404 error if the Recipient Registry cannot identify the Data Recipient using the provided recipient_id. This recipient_id corresponds to the client_id field passed to Data Providers indirectly by Data Access Platforms or directly by Direct Data Recipients in the OAuth authorization request.

Recipient Registry Identification

The exact identifiers used by Data Access Platforms or Direct Data Recipients and Data Providers to identify Recipient Registries is out of scope of this document. Data Access Platforms or Direct Data Recipients and Data Providers **MUST** define and agree to the identifier themselves (solving for any identifier conflicts).

Recipient Registry Authentication

The exact mechanism that Recipient Registries use to onboard and authenticate clients (e.g. Data Providers) is out of scope for this proposal. Recipient Registries and their clients **MUST** define and agree to this themselves, and may include sharing client identifiers and secrets.

Recipient Metadata

FDX RFC 0206 introduced the following new fields to the Recipient Metadata:

- recipient_id - Required
    - The Recipient identifier at the Recipient Registry.
- registry_references[] - Optional
    - An array of external registries containing registered_entity_name, registered_entity_id and registry fields for the registries where the data recipient or intermediary is registered. If one or more registry entries is included, then all three fields are required to create a valid array object.
    - registered_entity_name - Optional
        - The legal company name for the data recipient or intermediary in the identity chain. This may be the same as the client_name for a data recipient. In many cases this could be a parent company name or the entity name with Corporation or LLC
    - registered_entity_id - Optional
        - An ID representing the company that can be looked up from a legal identity registry source
    - registry - Optional
        - The Registry source. Enum values include [PRIVATE, FDX, GLEIF, ICANN]

The RFC modified the following field from the Recipient Metadata:

- redirect_uris[] - Optional
    - An array of eligible Redirect URI targets to which the Authorization code is sent at the completion of the consent process.

Inclusion of redirect_uris is possible in this version only in instances in which DAP is the registry. Future versions will determine how non-DAP ecosystem registry can manage redirect_uris.

The RFC removed the following field from the Recipient Metadata:

- intermediaries[]
    - No longer stored with the Recipient Metadata at the Recipient Registry and instead is passed to the Data Provider as part of the OAuth authorization request.

The RFC introduced the following new fields to the Intermediary Metadata:

- registry_references[] - Optional
    - An array of external registries containing registered_entity_name, registered_entity_id and registry fields for the registries where the data recipient or intermediary is registered. If one or more registry entries is included, then all three fields are required to create a valid array object.
    - registered_entity_name - Optional
        - The legal company name for the data recipient or intermediary in the identity chain. This may be the same as the client_name for a data recipient. In many cases this could be a parent company name or the entity name with Corporation or LLC
    - registered_entity_id - Optional
        - An ID representing the company that can be looked up from a legal identity registry source
    - registry - Optional
        - The Registry source. Enum values include [PRIVATE, FDX, GLEIF, ICANN]

New Fields

The Data Provider **MUST** extend its OAuth authorization endpoint to support two new authorization request fields, “registry” and “intermediaries”. If a Data Access Platform is involved, the Data Access Platform **MUST** indirectly provide the “registry” and **MUST** provide the “intermediaries” fields in OAuth authorization requests, if intermediaries are present. If a Direct Data Recipient is involved, the Direct Data Recipient **MUST** provide the “registry” and **MUST NOT** provide the “intermediaries” fields in OAuth authorization requests. These fields are defined below.

1. “registry” specifies the Recipient Registry where the Data Recipient has been registered and can be identified with the client_id passed as the recipient in the “/recipient/<recipient_id>” endpoint.
2. “intermediaries” which is a base64 encoded list of “Intermediary Metadata” objects that the Data Access Platform has collected in order for the Data Recipient to use its service.

RFC 9126: OAuth 2.0 Pushed Authorization Requests

Data Access Platforms or Direct Data Recipients, and Data Providers **MUST** agree to use RFC 9126: OAuth 2.0 Pushed Authorization Requests (PAR).

Part 3: Best Practices for Sharing Sensitive Data

Overview

The FDX API contains data elements that are generally accepted as sensitive data. The purpose of these best practices is to enable organizations that participate in App2App data exchanges some options for additional data protection and privacy of specific individual or groups of data elements. These techniques are intended to be used in addition to your organization's data security controls and requirements such as TLS message encryption.

While certain practices outlined in this section may be applied against any set of data, they were produced within the context of use cases supported by the Financial Data Exchange standard. Including but not limited to account verification, money movement, credit lending and management, and personal financial management. Specifically with the use of account number, account owner, and account address data elements.

Each of the presented best practices were identified through a detailed analysis that were evaluated against a set of criteria that were grouped into the following:

- Security
    - Increasing security while data is in motion
    - Increasing security during data handling
    - Implementation sensitivity
- Adoption
    - Feasibility and impact to the ecosystem in regards to data use, business process, solution dependencies, and participation.
- Implementation impact
    - General impact to specific use cases (verification, credit lending and servicing, money movement) and existing business process.

The recommended security patterns are grouped into either General Purpose Patterns or Use case patterns.

- General Purpose - Meets all three criteria and is recommended for all FDX use cases
- Use Case Patterns - Recommended for consideration by FDX use case task forces where an RFC describing the usage exists. Recommend RFC development where such an RFC doesn't exist, and would be an improvement over a General Purpose pattern.

For data elements other than account number, account owner name and address that are determined to be sensitive based on FDX use case, it is recommended that a review of patterns outlined in this document be conducted and given consideration for use. If none of them are determined to be fit for purpose than you are encouraged to submit an RFC to expand the FDX sharing of sensitive data best practices.

Terminology

The keywords "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this part are to be interpreted as described in IETF RFC2119.

- **MUST** - This word, or the terms "**REQUIRED**" or "**SHALL**", mean that the definition is an absolute requirement of the specification.
- **MUST NOT** - This phrase, or the phrase "**SHALL NOT**", mean that the definition is an absolute prohibition of the specification.
- **SHOULD** - This word, or the adjective "**RECOMMENDED**", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- **SHOULD NOT** - This phrase, or the phrase "**NOT RECOMMENDED**" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications **SHOULD** be understood and the case carefully weighed before implementing any behavior described with this label.
- **MAY** - This word, or the adjective "**OPTIONAL**", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option **MUST** be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option **MUST** be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

Certification

Use or non-use of the patterns outlined below are not determining factors in obtaining FDX certification.

Pattern Summary

Identified patterns have been grouped into two categories. Within these categories each of the applicable data elements and use cases are identified.

- General Purpose - these are comprised of those patterns that have been recognized as beneficial to all use cases.
    - Asymmetric Encryption – **SHOULD** be used for identified sensitive data elements as outlined in FDX RFC 0011
    - Granular Consent – **SHOULD** be applied for all use cases where consumer permission is required for the sharing of data.
- Use Case Specific - techniques that add security benefits when applied to a specific use case.
    - Tokenization – **SHOULD** be used as additional data security of account number for Money Movement use cases.
    - Alternative Data – **SHOULD** be used as an alternative to passing the account number when performing account validation.
    - Verification Query – **SHOULD** be used as an additional data validation technique for all data sensitive data elements within the scope of Identity Verification and Money Movement Setup.
    - Masking/Truncation – **SHOULD** be used when the full representation of the data element is not required for account verification, account identification, or API call requirements.
    - Hashing – **MAY** be used when the integrity of a data element **SHOULD** be validated.

A review was performed against emerging patterns that were determined to have a broader scope and benefit. While these are not detailed in this section there is support for continued discussion of their use. For more information on these please refer to the section *Emerging Security Pattern Overview*.

- Trust Frameworks
- Homomorphic encryption
- Polymorphic encryption
- Secure Multi-Party Computation (SMPC)
- Trusted Execution Environments

Impact and Benefits

Providing best practices that have been agreed to by industry participants will lead to increased security, ease of adoption, consumer confidence, and innovation by achieving the following:

- Improved protections of relevant data by authorized parties as shared throughout the ecosystem.
- Supplementing message encryption that terminates at an API gateway by encrypting specific data elements throughout internal networks.
- Benefits existing data security models by layering techniques.
- Providing additional transparency to consumers in regards to how their data is being used.
- Challenges entitles to establish alternative data-driven solutions and use minimal data practices.
- Limiting the need for account number replacements and thus minimizing disruptions in money movement.
- Reduction in fraudulent transactions being processed.
- Minimizes the amount of sensitive data replicated throughout the ecosystem.
- Improvements in maintaining data integrity.

Pattern Details

The following patterns are recommended in the context of App2App integrations. Data at rest is not addressed with this technique. All patterns are to be considered as additive to existing security practices that your organization has in place.

General Purpose Patterns

Asymmetric (Hybrid) Encryption

Data in Scope	All data
Use Cases in Scope	All Use Cases
Considerations	• Use only for hops from data provider to data access platform, and from data access platform to data recipient. Reference App2App (FDX RFC 0011). • Only encrypt relevant data keeping data needed by intermediary systems in the clear.
What Problem Can/Does It Solve	Prevents PII and sensitive data from traversing the internal network unencrypted. TLS will typically terminate at the API gateway and the raw content will traverse the internal network unencrypted. Supports data minimization along with controlling what consumer information is being secured. Thus supporting bi-lateral agreements. Layered prevention against first party attacks, compromised transport-layer-security encryption. In alignment with FDX security control considerations. Should not be used to circumvent bi-lateral or multi-lateral agreements. Could be used with additional patterns to provide additional levels of security.

Granular Consent

Data in Scope	All data
Use Cases in Scope	All Use Cases

Considerations	• A granular consent **SHOULD** always be used where possible to separate use case consents that need access to sensitive data from those that don't. • Information on requested use case and the associated sensitive data **SHOULD** be made available to all parties - data providers and data access platforms - to enable them to trigger appropriate controls for the data and use case.
What Problem Can/Does It Solve	Limiting the delivery of data provided through exchanges between parties along with providing transparency to the end user. Thus enabling consumers to have a clear and concise understanding of the data use when it comes to making informed decisions. Provides consumers a means for increased control over the privacy of their data.

Use Case Specific Patterns

Tokenization

Data in Scope	Account Number
Use Cases in Scope	Money Movement
Considerations	• Implemented at the data provider for better security and control. • Ensure the tokens are as usable for the purpose as the original account number e.g., no change to ACH, SWIFT, or other money movement schemes **SHOULD** be needed. • Use of this pattern outside of the use of money movement could lead to limited participation. Thus making the pattern use case specific. Should not restrict use of the "network". • Implementation is based on FDX RFC 0005. Other uses **SHOULD** be reviewed and not consider this to be a general use pattern.

What Problem Can/Does It Solve	Protects the account number from being leaked by using a substitute account number that can only be used to execute transactions. Substitute account numbers can be reissued and replaced without impacting the end customer. The end customer can revoke consent, deactivating a substitute account number and taking away any account holder ability to move money. Replacing account numbers can be costly/risky for the account holder as it requires the customer to be involved. Reduces the risk to all parties in the chain as nobody holds the actual account number.
Alternative Data	
Data in Scope	Account Number
Use Cases in Scope	Account Verification
Considerations	• Potential for removing sensitive data from transaction. • Account validation can be done through data such as transaction history rather than account number • Becoming more common that credit furnishers are not providing the full account numbers • Can impact the robustness of a automated verification process • Without the full account number it can lead to fraud
What Problem Can/Does It Solve	• Replaces the use of sensitive data with non-sensitive data reducing the need for additional security measures or design patterns associated with sensitive data. • Accomplishes the same business objective using non-sensitive data, which is neutral from a business perspective, but superior from a compliance, risk and security perspective. The exact alternative data would be decided on a use-case-by-use-case basis. • Minimizes the amount of sensitive data throughout the ecosystem.
Verification Query	
Data in Scope	All data

Use Cases in Scope	• Account Owner Identity Verification • Money Movement Setup
Considerations	• Security risk is deemed low because the sensitive data is gathered and transmitted to the provider for verification. If that is a concern, hashing may be employed to prevent transmission in the clear. • Instead of requiring the data recipient to send the end-user account number to verify, the data recipient sends an end-user identifier e.g., account number or phone number, to the data provider. The data provider compares that with the account number or phone number on record responding with yes if the data matches and no if they do not. As part of the verification, data provider may verify or notify end-user via 2nd or 3rd factor e.g., by sending a text notification or request to the phone number on file. A similar service is currently being piloted by SSA.gov. • Bank information becomes the primary source versus derived.
What Problem Can/Does It Solve	• Reduces the risk from rogue or poorly implemented data recipient apps. Since this method relies on end-user providing the sensitive data to verify, it prevents the data recipient from obtaining this data from data provider without the end-user knowing about it, or worse, under false pretexts. • It also reduces the risk surface for Accunt Takeover (ATO) fraud by making it difficult for a fraudulent user that took over legitimate user's credentials from carrying out fraud, as the fraudulent user now also needs to know the sensitive data to complete the operation.
Masking / Truncation	
Data in Scope	All Data
Use Cases in Scope	Account Verification, Account Identification, API call requirements.

Considerations	• Reconciliation of information with partially masked data when recipient party already has unmasked data element. • Integrity of data structure with selective masking of data elements. • Data Recipient discretion based on use case, recipient needs the ability to identify in the request what data is needed (may need enhancement to API). • Masking used in conjunction with Alternative Data and Verification Query increases the value proposition.
What Problem Can/Does It Solve	• Protects the data element from being leaked or re-distributed as source data is masked • Pseudonymization for analytical modeling, regulatory compliance and privacy scenarios

Hashing with Salting	
Data in Scope	Account Number, Account Holder Name, Account Holder Address.
Use Cases in Scope	Account Verification
Considerations	• Hashing is a method used to verify the integrity of a value has been maintained by transforming the original value into a fixed-length value. Hashing algorithms are specifically used to ensure the original value has not been modified or tampered. • Hashing is not the encryption of data in that it is a one-way transformation of the data. Whereas encryption algorithms are two-way (encryption, decryption) functions. • This recommendation is limited to the use of SHA256 or SHA-3 (Secure Hashing Algorithms). • The technique is versatile in that it can be combined with various encryption techniques for added security. • Use is well known and there are no real blockers to adoption. • If used it **SHOULD** be a complement to other data encryption/protection techniques. • Both parties know the original data that is used for comparing.

What Problem Can/Does It Solve	Hashing provides a solution for ensuring the integrity of a given value while adding an additional level of security. Recipients are able to perform verification without needing the value in clear text. If encryption is not implemented/not being used than it is a feasible alternative to ensure the integrity of the data element and/or message payload.

Emerging Security Pattern Overview

Emerging patterns are viewed as having promise, but at the time the FDX API Security Model task force had insufficient information or expertise to analyze.

Pattern	Reason and/or Background Information
Trust Frameworks	• Described in W3C+DID - an ability for end-user to present their identity in a cryptographically verifiable manner without the reliance on account attributes is attractive, and could drastically alter KYC, account owner verification algorithms. The task force felt that the framework needs to be described and adopted at least within FDX context before it can be recommended. Hence, the recommendation is to include it on Security WG roadmap for definition, review, and approval. Once defined, and approved, it may be picked up again for consideration.
Homomorphic encryption	• This is best used for manipulating data without decrypting it. Within the current scope of account number, account owner name, and account owner address, mathematical operations generally do not apply. It may be worthy of picking up for other sensitive data for use cases requiring logical or mathematical operations.
Polymorphic encryption	• Computing resources and Data Governance concerns. Complexity of adoption that can be addressed with more broadly adopted security patterns that achieve similar protections.
Secure Multi-Party Computation (SMPC)	• Compute on non-shared data.
Trusted Execution Environments	• A protected environment that guarantees the confidentiality and integrity of the data shared within it.

Part 4 End to End Encryption

Overview

The purpose of this section is to define the standard for an optional feature of the FDX API in which Data Providers, Data Access Platforms and Data Recipients protect the transport of certain consumer information through application-level message encryption that uses hybrid cryptography.

Security Guidelines

Financial institutions operating in the United States are subject to US federal law and regulatory guidance regarding the protection of consumer information. The Gramm-Leach-Bliley Act requires financial institutions to protect consumers' "non-public personal information." The Federal Financial Institutions Examination Council specifies requirements for encryption and key management for consumer information.

Applicability

Agreements between Data Provider, Data Access Platform and Data Recipient **SHALL** govern whether shared consumer information **SHALL** be encrypted, which consumer information **SHALL** be encrypted at the application level and which party **SHALL** be the owner of the private key of an asymmetric key pair.

Transport protection of consumer information between Data Providers, Data Access Platforms and Data Recipients is in scope.

Transport protection of consumer information from the Data Provider, the Data Access Platform or the Data Recipient to the End User's device is not in scope.

Protection of consumer information at-rest is not in scope.

Assumptions

Authentication, authorization and transport layer security between the Data Provider and the Data Access Platform or between the Data Provider and Data Recipient **SHALL** comply with *FDX Control Considerations*.

Message Encryption

Payload Encryption

Payload encryption from Data Provider to Data Access Platform or Data Recipient **SHALL** comply with nested JWT implementation specified in the RFC 7519 JSON Web Encryption, with the JWS encapsulated in the JWE.

Field Encryption

Field encryption from Data Provider to Data Access Platform or Data Recipient **SHALL** comply with nested JWT implementation specified in the RFC 7519 JSON Web Encryption, with the JWS encapsulated in the JWE.

Key Management

A nested JWT implementation requires two pairs of asymmetric keys and one symmetric key. The symmetric key is used for content encryption. One asymmetric key pair is for the encryption and decryption of the symmetric content encryption key. The second asymmetric key pair is for signing the payload and validating the signature.

Encryption and decryption public key and private key pair and public key certificate

The encryption and decryption public key and private key pair are used in the JWE to encrypt and decrypt the symmetric content encryption key.

Certification Authority

The issuing Certification Authority (CA) **SHALL** be suitable to the Data Provider and the Data Access Platform, or the Data Provider and the Data Recipient for cases in which the Data Provider shares customer information directly with the Data Recipient. The CA **SHALL** comply with the current approved versions of NIST Special Publication 800-57 Parts 1 and 2.

Exchange

Message Encryption requires a means for the provider and recipient of encrypted consumer information to share encryption public keys. Methods for sharing public keys may include:

- JSON Web Key Set endpoint
- Directory of public keys
- Another method as agreed by the implementers.

Methods for protecting access to public keys **SHALL** be determined by the implementers. To protect the connection from the Data Provider to the JWKS endpoint of the Data Access Platform or of the Data Recipient, the connection may occur after mutual authentication via OAuth 2.0 (FAPI).

Key storage

The owner of the encryption private key **SHALL** protect its private key in a system that has been validated as meeting at least FIPS 140-2 level 3 or Common Criteria Protection Profile evaluation assurance level 4.

Key usage

The public key - private key pair **SHALL** be used for encrypting and decrypting the content encryption key used to encrypt and decrypt consumer information and have no other uses.

The owner of the encryption private key **SHALL NOT** share the private key with parties outside of its organization.

Crypto-period

The duration of the crypto-periods for encryption public key and private key pairs **SHALL** be no longer than two years.

Algorithm and key size

Algorithms and key sizes **SHALL** comply with the recommended JOSE implementation requirements of the current version of IANA JSON Object Signing and Encryption.

Other key management practices

Other key management practices of the owner of the encryption private key, including but not limited to auditing, key destruction, key recovery and key compromise, **SHALL** comply with the current versions of NIST Special Publication 800-57 Parts 1 and 2.

Signing public key and private key pair

The signing asymmetric key pair is for signing the payload in the JWS and validating the signature.

Certification Authority

The issuing Certification Authority (CA) **SHALL** be suitable to the Data Provider and the Data Access Platform, or the Data Provider and the Data Recipient for cases in which the Data Provider shares customer information directly with the Data Recipient. The CA **SHALL** comply with the current approved versions of NIST Special Publication 800-57 Parts 1 and 2.

Exchange

The signing public key is sent in the header of the JWS as specified in IETF RFC 7515 JSON Web Signature.

Key Storage

The owner of the signing private key **SHALL** protect its private key in a system that has been validated as meeting at least FIPS 140-2 level 3 or Common Criteria Protection Profile evaluation assurance level 4.

Key Usage

The signing public key - private key pair **SHALL** be used for signing the payload in the JWS and validating the signature and no other uses.

The owner of the signing private key **SHALL NOT** share the private key with parties outside of its organization.

Crypto-period

The duration of the crypto-periods for signing public key and private key pairs **SHALL** be no longer than two years.

Algorithm and key size

Algorithms and key sizes **SHALL** comply with the recommended JOSE implementation requirements of the current version of IANA JSON Object Signing and Encryption.

Other key management practices

Other key management practices of the owner of the signing private key, including but not limited to auditing, key destruction, key recovery and key compromise, **SHALL** comply with the current versions of NIST Special Publication 800-57 Parts 1 and 2.

Content encryption key

The symmetric key is used for content encryption.

Key management

Key management of the content encryption key **SHALL** comply with RFC 7516 JSON Web Encryption (JWE).

Algorithm and key size

Algorithms and key sizes **SHALL** comply with the recommended JOSE implementation requirements of the current version of IANA JSON Object Signing and Encryption.

Examples

The examples below are provided for clarity only. The individual JWE and JWS RFCs (RFC 7516 and RFC 7515) referenced supersede implementation details where the examples are inconsistent.

Example: Payload Encryption using a Nested JWT

HTTP Request

```
GET /customers/current HTTP/1.1

...
```

HTTP Response

```
HTTP/1.1 200 OK

Content-type: application/jwt

...
```

```
//Nested JWT (JWS encapsulated in a JWE)

//JWE outside, JWS inside

{

    "protected": BASE64({

        "alg":"RSA-OAEP",

        "enc":"A256GCM"

    }),

    "unprotected:": {

        "jku":"https://server.example.com/keys.jwks"

    },

    "ciphertext": BASE64(ENCRYPT<A256GCM using
'encrypted_key'>({ //JWS to be encrypted (shown before
encryption below)

        "protected": {

            "kid":"123",

            "jku":"https://server.example.com/keys.jwks",

            "alg":"RS256"

        },

        "payload": BASE64({//The encrypted content will be valid
schema of the `/customers/current` endpoint ie. `Customer`
entity

            "customerId" : "abc-12345-xyz"

            "name" : {

                "first" : "John",

                "last" : "Smith"

            }

            "email" : ["jsmith@email.com"]

        }),
```

```
        "signature": //Using Data Provider signing private key


"nupgm7iFqSnERq9GxszwBrsYrYfMuSfUGj8tGQlkY3Ksh3o_IDfq1GO5ngHQLZu
YPD-
8qPIovPBEVomGZCo_jYvsbjmYkalAStmF01TvSoXQgJd09ygZstH0liKsmINSti"

    ))},

    "iv": "AxY8DCtDaGlsbGljb3RoZQ"

    "encrypted_key": //A256GCM key encrypted using RSA-OAEP with
Data Access Platform's Public Key


"UGhIOguC7IuEvf_NPVaXsGMoLOmwvc1GyqlIKOK1nN94nHPoltGRhWhw7Zx0-
kFm1NJn8LE9XShH59_i8J0PH5ZZyNfGy2xGdULU7sHNF6Gp2vPLgNZ__deLKxGHZ
7PcHALUzoOegEI-8E66jX2E4zyJKx-YxzZIItRzC5hlRirb6Y5Cl_p-
ko3YvkkysZIFNPccxRU7qve1WYPxqbb2Yw8kZqa2rMWI5ng8OtvzlV7elprCbuPh
cCdZ6XDP0_F8rkXds2vE4X-ncOIM8hAYHHi29NX0mcKiRaD0-D-ljQTP-
cFPgwCp6X-nZZd9OHBv-B3oWh2TbqmScqXMR4gp_A",

    "tag": //Protection of payload

         "Mz-VPPyU4RlcuYv1IwIvzw"

}
```

7.2 Example: Field-level Encryption using a Nested JWT

```
HTTP Request

GET /customers/current HTTP/1.1

...

HTTP Response

HTTP/1.1 200 OK

Content-type: application/json

...



{

    "customerId" : "abc-12345-xyz"

    "encryptedName" : //Nested JWT (JWS encapsulated in a JWE).
JWE outside, JWS inside
```

```
    {
        "protected": BASE64{
            "alg":"RSA-OAEP",
            "enc":"A256GCM"
        }),
        "unprotected": {
            "jku":"https://server.example.com/keys.jwks"
        }
        "aad": BASE64({ //Additional Authenticated Data that
contains the JWS with non-PII
            "protected": {
                "kid":"123",
                "jku":"https://server.example.com/keys.jwks",
                "alg":"RS256"
            },
            "payload": BASE64({ //Non-PII fields that are signed
but not encrypted. For example only.
                "bank": "Chase"
            }),
            "signature": //Signature provides a means to verify
the authenticity
                "DtEhU3ljbEg8L38VWAfUAqOyKAM6-Xx-
F4GawxaepmXFCgfTjDxw5djxLa8ISlSApmWQxfKTUJqPP3-Kg6NU1Q"
                }),
        "ciphertext": BASE64(ENCRYPT<A256GCM using
'encrypted_key'>({ //JWS to be encrypted (shown before
encryption below)
            "protected": {
                "kid":"123",
```

```
                "jku":"https://server.example.com/keys.jwks",

                "alg":"RS256"

            },

            "payload": BASE64({ //Encrypted content matches the
schema of the "name" field of the `Customer` entity. The
`CustomerName` entity

                "first" : "John",

                "last" : "Smith"

            }),

            "signature":


"nupgm7iFqSnERq9GxszwBrsYrYfMuSfUGj8tGQlkY3Ksh3o_IDfq1GO5ngHQLZu
YPD-
8qPIovPBEVomGZCo_jYvsbjmYkalAStmF01TvSoXQgJd09ygZstH0liKsmINSti"

        }),

        "iv": "AxY8DCtDaGlsbGljb3RoZQ",

        "encrypted_key": //A256GCM key encrypted using RSA-OAEP
with Data Access Platform's Public Key


"UGhIOguC7IuEvf_NPVaXsGMoLOmwvc1GyqlIKOK1nN94nHPoltGRhWhw7Zx0-
kFm1NJn8LE9XShH59_i8J0PH5ZZyNfGy2xGdULU7sHNF6Gp2vPLgNZ__deLKxGHZ
7PcHALUzoOegEI-8E66jX2E4zyJKx-YxzZIItRzC5hlRirb6Y5Cl_p-
ko3YvkkysZIFNPccxRU7qve1WYPxqbb2Yw8kZqa2rMWI5ng8OtvzlV7elprCbuPh
cCdZ6XDP0_F8rkXds2vE4X-ncOIM8hAYHHi29NX0mcKiRaD0-D-ljQTP-
cFPgwCp6X-nZZd9OHBv-B3oWh2TbqmScqXMR4gp_A",

        "tag": //Protection of AAD and payload

            "Mz-VPPyU4RlcuYv1IwIvzw"

    }

}
```

References for End to End Encyption

1. Barker, E., NIST Special Publication 800-57 Part 1 Revision 5: *Recommendation for Key Management*, May 2020. <https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final>

2. Barker, E., Barker, W., NIST Special Publication 800-57 Part 2 Revision 1: *Recommendation for Key Management,* May 2019. <https://csrc.nist.gov/publications/detail/sp/800-57-part-2/rev-1/final>

3. FIPS PUB 140-2 *Security Requirements for Cryptographic Modules*, May 2001. <https://csrc.nist.gov/publications/detail/fips/140/2/final>

4. [JOSE] JSON Object Signing and Encryption (JOSE), <https://www.iana.org/assignments/jose/jose.xhtml>

5. [JWA] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, May 2015, <http://www.rfc-editor.org/info/rfc7518>.

6. [JWE] Jones, M., "JSON Web Encryption (JWE)", RFC 7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

7. [JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>

Nested JSON Web Token (JWT) using RSA

Overview

This section provides implementation details on using RSA encryption algorithms for JSON Web Signatures and JSON Web Encryption.

Pre-conditions

Both the data provider and data recipients **SHOULD** generate RSA key pairs and the public keys **SHOULD** be available as signed certificates on a JSON Web Key set (JWKS) endpoint.

RSA: Creating Key Pairs Using OpenSSL

Creating a private RSA key using the command line

```
openssl genrsa -out private_key.pem 2048
```

Creating a public RSA key from a private key

```
openssl rsa -in private_key.pem -outform PEM -pubout -out
public_key.pem
```

RSA: Sample Keys Used in Examples

Data Provider's Key Pair

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIEpAIBAAKCAQEA40b3nY6KgCBekJVfj3MbbXEMvcDFz1ew97LQ2B8AuQi3Khtl
RpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAnpAM8UJdmW2jtvUyE+7WjmEZkW9E+
xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/16wRZ1HonV6OjXOjp5fF2og4OhL3S
2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5AYbGEVoP2VKMe2KKQIBypqawxBIl
qVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+a7e66pWlk7XfykZ5Bg9/dm+1aZPc
653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi4QIDAQABAoIBAQCR1UOlcYUlWZ/U
HkTHAxEiVFvclglXVelTxwWC58HK+PCusA14Eb2x1eLSb9P89g55P6YWitWJ/7ym
EuJ1jYFiGEFnZP5kwRtrT+I0sQRb/S8AS29/Zrm/rVQw2yYrSiR5xilZnjNpUmXz
tyEmuXMObihF0mvULEfyofETfrCkojKDRe0AO5B1FCY4EHKcssuqdpux57+nB1rd
2CUv+6LuAb0IB0fB1KQP24d0NCuhbKUly9aiJMfLJoHmb6KEJfsySYpmzBxOSH71
V1GTjDzmrqRSbgAse7+XnlYRSAz091hpqQpj1EJge0l/1VDEJessxCbOLO1g5P55
fUgxZQbBAoGBAPy1ZGbafqAeQuHQxg1ehix3n2kLnwM4HBOhmNf+GvL0vHbIyAVY
VuuCiGSlyOPCJsjLH7d0/PMqMMtR1AgeS8LZS8PAAB6Yy8HRAZQhOnyx8aqKHEyc
i0+q2IqQXLld1R8QRdEy2KmvE08iQ+fQo112q4XCxIdkio3rbjbGFznFAoGBAOY8
x3Yh6Of92Rnn5muJ3WHn1gK9i4rS1KgJNFvzEaaBc26QQeEmP9kldx3KFOVUBk6S
ZlZzSLc9QggB05/om5w9wztS8f6kg1+jOa6V/SEKkosyHsueSUsRV8lNgbF9ODOf
45Vx0eWHa5YFrnYI0KgLYrgXDvrE8+An2boA1kJtAoGAIJBLQMm0+XMM0UZyzvQ4
O/CqNQIPWn3XeFwhcuvGkzogMvpKdA3fHXfzlWybh2XUU5mBG8XSdo8gPILt3KHy
x0fy8GWEXmz4DKCfIHRrsffIGV60qNafSQPN5YUWvbgup1MUfBGeQ7dQuKjEsVF8
S6XoElN3ua6mAAWvbRV3lrUCgYEAqtBA82XpE+UDYvLnwrT/6BlGb7YMhywv1ZMu
o2FMoQm9iDPCjLYB/KqNGy7IHfQe0cBP6KeTNU9fY+1nAmZivKId7C93loKbbSL+
MobYy+C6JEdFDbAblHQDezfjlrjeL37aLA/Lt8ymhyEj9DJKC8KWtRl2ZZoljRJD
uHnSfGkCgYAsYv6QDMPnJH1T2rGOm0X6roH3M4OTiXnQXBMOxbUzYacg4/1bBY1L
/uGXmq1mS19ntHHAaf/wTrqlP/hCh7cHBy3SCh3Pt/ktd+90ZXaU3au9mAVAX72j
xYzYot10UYj/autRMH/Vxt399DZ5reA/gaOBnK2pSHwwW0oAr1ERgA==
-----END RSA PRIVATE KEY-----
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA40b3nY6KgCBekJVfj3Mb
bXEMvcDFz1ew97LQ2B8AuQi3KhtlRpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAn
pAM8UJdmW2jtvUyE+7WjmEZkW9E+xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/1
6wRZ1HonV6OjXOjp5fF2og4OhL3S2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5
AYbGEVoP2VKMe2KKQIBypqawxBIlqVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+
a7e66pWlk7XfykZ5Bg9/dm+1aZPc653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi
4QIDAQAB
-----END PUBLIC KEY-----
```

Data Recipient's Key Pair

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAuRK7Tfo+7OBUcuI/Tk4Qnh0lamAnwMXK/Kcd5Sz1P9DJl4QE
4tY5Wv6IKMCbdYKX5muo8DjahR/ZtAv7UvPobprmmWU2201uLaLROmuhTJzWWFqE
JGhvtabwdr7/7JsXywQI1ZchzON4RUJaukqu0MsUnfVZEJAedYvUZBfY5Pa6GCl+
dgSV9vZ6nsFGcH/hmQU7qKrk4PrTNV4lc5ehO7Tkckn0BA9soDjBGfdxaYwxm9kU
jgkJMQYYIkm4gzPfLk7UlLhMRy9UWBoRvajGXz5vrif/0KIFKVQtjbt108rwC+/u
KVlmEHqybkanSG8T1tDL1BLKpKxwtZo0fsAURQIDAQABAoIBAQCyuNwYWWka5yem
KcZooAp8JjlTmLfK+Tck9V1xSxErJz0GDH+LbsTEkrh6YkW+HPcDlUP3d2/Ozws1
S7zQRqpW5U97Irru3L9hYrIacIW7rllvyTmCzzfRe/0LLzeGDd/UNXkyilghvCkQ
+RsUe7qF9xdZ0uzHieVgLkBUyJKzx8k40VfJE/2FdPZafdDMv1p+4CkDpxZUS7F6
MPFSuAumLJ3lSUey0B1+IpUZhZBaSjHVyBfUOWluOEtz99NjQKS/8I8lYetAJ+Mo
GkkXLW5sHRwkdhc050cuKp41BEOeByCtESH30vJLEvwbF4o2/vj6Z+v4NvpSZMmA
0/p02goBAoGBAPCn0kRjMKOxmYJVkZkQFHpVs6LH/J0SZY3K3bdJfiNmOYdYVyAn
NZQ9IvQ712AJTuVawkOhjBnTeMD5TUQvjV0WAVp9AIu8oV3vTcl/8oubZ9D4COt9
S+aJOTkwD8voH2MkQc7tNl7Qw31wcE+MGIKh3qigdgwifkhFzmaqpt1xAoGBAMTf
phXzMmkwKEW5v/ULLsOHS96x89izjebu8P8fWbuVgU0sK6n46Gfxb798hEd7gjU/
```

```
EzOSbNRIFyertRIiUUWcmeo6d9aqxbzfkc2GqEJ+Jym5QFSlJgxqxMtmTPnO5Bat
S58/Scl7TSqk5bE3j2yBR4rJEjur5V1SldByQYoVAoGATKpJ8/tdbWiQrNKxtX9H
5skSlxL6yNcpfwhXpaJGCuTwAswDxXx4Nyda0U+XB0Mv3SUSqhT22uthlqhVExnL
ARKXj8ouuFV5WsF3mG+oRw1U/19lCBA8c87Xaf6DqcPi6+SLCm7LWV1MSdPeE5lf
3Y3PrwyfTrJWZJPIczB+RCECgYBYoVbkCthnAoce3MDOUHp9DCvb1cExjaQUkv1r
3XFIQcY0N+5wVt5J7SehzSzAAZpc3kiGryTPbKT/9w1NXKW58QZZrHjG65qZrQy2
uiiFxsVaw0tyz+aRMH/oEeYVkE6e5uVki9lsG1ZiHFpLrfejoY/TqzHKK1jW6pcH
gGiBAQKBgAIeLlx91P4DLHuHBSmAWElqVVJf+nNQYBX7uAnftGlD+/PcyquPE/wL
eKgQzj6g0/kDM6Erex08hcw2bkc0ZULfhFFMt0NwGGhVGyt+eQOR1pkIJFLy72TX
WrsXAt3Nkhzw59ZyMT2hxpdEsxdeosoioJkDBBXaNcdeck4OLTNN
-----END RSA PRIVATE KEY-----
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuRK7Tfo+7OBUcuI/Tk4Q
nh0lamAnwMXK/Kcd5Sz1P9DJl4QE4tY5Wv6IKMCbdYKX5muo8DjahR/ZtAv7UvPo
bprmmWU2201uLaLROmuhTJzWWFqEJGhvtabwdr7/7JsXywQI1ZchzON4RUJaukqu
0MsUnfVZEJAedYvUZBfY5Pa6GCl+dgSV9vZ6nsFGcH/hmQU7qKrk4PrTNV4lc5eh
O7Tkckn0BA9soDjBGfdxaYwxm9kUjgkJMQYYIkm4gzPfLk7UlLhMRy9UWBoRvajG
Xz5vrif/0KIFKVQtjbt108rwC+/uKVlmEHqybkanSG8T1tDL1BLKpKxwtZo0fsAU
RQIDAQAB
-----END PUBLIC KEY-----
```

RSA: JSON Web Signature

This example shows how to create a JWS using RSASSA-PKCS1-v1_5 and SHA-256

Assemble the JWS Protected Header by adding all of the header elements that are to be included as part of the signature

```
{
  "alg": "RS256",
```

```
  "jku": "https://server.example.com/keys.jwks",
  "kid": "123"
}
```

JWS Payload

```
{
  "customerId": "abc-12345-xyz",
  "name": {
    "first": "Jane",
    "last": "Smith"
  },
  "email": [
    "jane_smith@email.com"
  ]
}
```

JWS Signature

To create the JWS signature, the protected header and payload **MUST** be base64URL-encoded and then joined together by a period .

Encoded header and payload

BASE64URL(UTF8(JWS Protected Header))

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyIsImtpZCI6IjEyMyJ9
```

BASE64URL(JWS Payload)

```
eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJK
YW5lIiwibGFzdCI6

IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWwuY29tIl19
```

Join the encoded header and payload using a period (.) with no spaces before or after to create the input for JWS signing

BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload)

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyIs
```

```
ImtpZCI6IjEyMyJ9.eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWU
iOnsiZmlyc3QiOiJ

KYW5lIiwibGFzdCI6IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWw
uY29tIl19
```

Signing the encoded header and payload with the SHA-256 algorithm

Hexadecimal JWS Signature

```
0ea7ffcc4d96d0af0996e3f4e4f17bb30a2934712c561ef5a0dbe90156ff1459
```

Then Base64URL encode the result into:

Encoded JWS Signature

```
Dqf_zE2W0K8JluP05PF7swopNHEsVh71oNvpAVb_FFk
```

Concatenating these parts in the order Header.Payload.Signature with periods between the parts yields:

Complete JWS representation (with line breaks for display purposes only)

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyIs

ImtpZCI6IjEyMyJ9

.

eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJ

KYW5lIiwibGFzdCI6IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWw
uY29tIl19

.

Dqf_zE2W0K8JluP05PF7swopNHEsVh71oNvpAVb_FFk
```

RSA: JSON Web Encryption

Encryption

> **Note**: This example uses an RSA signed JWS. You may see signed payloads encrypted using other algorithms such as Elliptic Curve.

1. JWE Protected Header

```
{

  "alg": "RSA-OAEP-256",

  "cty": "JWT",
```

```
  "enc": "A256GCM"

}
```

2. Generate a 256-bit AES Content Encryption Key (Symmetric Key) Hex Byte Array

```
[b1 29 bc ff 2b 76 01 d3 a9 06 d7 0c b6 ac 7b 5a fe c5 49 0a 86
f6 f1 1a a1 5c

8a fa 20 39 18 22]
```

3. Generate the Initialization Vector Hex Byte Array

```
[b2 37 21 88 5c a4 21 9f 5a 6f f3 96]
```

Byte array encoding as BASE64URL

```
sjchiFykIZ9ab_OW
```

4. Encrypt the compact serialized form of the JWS using A256GCM with the content encryption key and initialization vector byte arrays as inputs. The two outputs of the process will be the ciphertext and authentication tag value

Ciphertext

```
lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7

vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSrgc80ZbYajZkaZZqfTgY2QY3pEGmz
pELnC65vQF2No6fH

Xdd3CewdoZHagyHFhBtqqoDRAw5q-Naj5O36GJXT--
l2D1yxBTN8tZs7FNZKqRmFngPUj17wyM27vjqC

8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW-nTY-xm-
fH6xUL6ZQvFQnBa2jnJIsQ_WYIkEOddBd_0X

NJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGCZO7WFFh580Af2cyOe7vTD8irUm02
qnxFGIMgOjLGbS87

1zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySYHGIv70xJq1ZQQ1MohxwJ_0jUkWcT
VWa-n-xDSESMGEH_

1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIkwB0xDIJOERB4
EjRoWVWgYLGSduZb

SpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jrm95ot48_Chs2G7A8JRXUl2sx9pbf
ayxktr0hBNmcVx3k

Poww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn9XLj8dD0aimhz6vPMOX4zAtdaWc0

3fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJPMDbC92G1oEc
```

Authentication Tag

```
4aT8aykEG7TAVrBfS27qaw
```

5. Encrypt the Content Encryption Key with Data Recipient's public encryption key using RSA-OAEP-256

```
JdC7ydHPnn-lrrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLL

D4D0FtLm6f_S-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8

f0_D3-JNkkNpMLUYZAWSE8ZgRclwOHL1KD6zhBIF-
uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-CJcxYPo

JmjAeSdooahBrsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto

0kDQMDf_Tk2igBfslXYiTA
```

JSON Web Encryption Generation

A JWE consists of the following 5 elements from the previous encryption steps encoded in BASE64URL form .

1. BASE64URL(UTF8(JWE Protected Header))

```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NN
In0
```

2. BASE64URL(JWE Encrypted Content Encryption Key)

```
JdC7ydHPnn-lrrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLL

D4D0FtLm6f_S-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8

f0_D3-JNkkNpMLUYZAWSE8ZgRclwOHL1KD6zhBIF-
uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-CJcxYPo

JmjAeSdooahBrsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto

0kDQMDf_Tk2igBfslXYiTA
```

3. BASE64URL(JWE Initialization Vector)

```
sjchiFykIZ9ab_OW
```

4. BASE64URL(JWE Ciphertext)

```
lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7

vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSrgc80ZbYajZkaZZqfTgY2QY3pEGmz
pELnC65vQF2No6fH

Xdd3CewdoZHagyHFhBtqqoDRAw5q-Naj5O36GJXT--
l2D1yxBTN8tZs7FNZKqRmFngPUj17wyM27vjqC

8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW-nTY-xm-
fH6xUL6ZQvFQnBa2jnJIsQ_WYIkEOddBd_0X

NJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGCZO7WFFh580Af2cyOe7vTD8irUm02
qnxFGIMgOjLGbS87

1zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySYHGIv70xJq1ZQQ1MohxwJ_0jUkWcT
VWa-n-xDSESMGEH_

1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIkwB0xDIJOERB4
EjRoWVWgYLGSduZb

SpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jrm95ot48_Chs2G7A8JRXUl2sx9pbf
ayxktr0hBNmcVx3k

Poww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn9XLj8dD0aimhz6vPMOX4zAtdaWc0

3fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJPMDbC92G1oEc
```

5. BASE64URL(JWE Authentication Tag)

```
4aT8aykEG7TAVrBfS27qaw
```

JWE in JSON Serialized Form

```
{

  "protected":
"eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0N
NIn0",

  "encrypted_key":

"JdC7ydHPnn-lrrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLLD4D0FtLm6f_S-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8f0_D3-JNkkNpMLUYZAWSE8ZgRclwOHL1KD6zhBIF-
uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-
```

```
CJcxYPoJmjAeSdooahBrsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto0kDQMDf_Tk2igBfslXYiTA",

  "ciphertext":

"lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJY
H7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSrgc80ZbYajZk
aZZqfTgY2QY3pEGmzpELnC65vQF2No6fHXdd3CewdoZHagyHFhBtqqoDRAw5q-
Naj5O36GJXT--
l2D1yxBTN8tZs7FNZKqRmFngPUj17wyM27vjqC8liKc3DJM3wzFNC98ixsqjlAsi
MJJ7lT_tXW-nTY-xm-
fH6xUL6ZQvFQnBa2jnJIsQ_WYIkEOddBd_0XNJMFf62l6g6IObrHSc1pEGsviIAF
FvlOXzGCZO7WFFh580Af2cyOe7vTD8irUm02qnxFGIMgOjLGbS871zJ_40_T3xfe
_8e7nKgX4ti4PuwDmi5aBySYHGIv70xJq1ZQQ1MohxwJ_0jUkWcTVWa-n-
xDSESMGEH_1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIkwB
0xDIJOERB4EjRoWVWgYLGSduZbSpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jrm9
5ot48_Chs2G7A8JRXUl2sx9pbfayxktr0hBNmcVx3kPoww-
xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn9XLj8dD0aimhz6vPMOX4zAtdaWc03fe_neuU3B19A2UvTz
ymO4EnY4fA_dngBYzJPMDbC92G1oEc",

  "iv": "sjchiFykIZ9ab_OW",

  "tag": "4aT8aykEG7TAVrBfS27qaw"

}
```

JWE in Compact Serialized Form

1. BASE64URL(UTF8(JWE Protected Header)) || '.' || BASE64URL(JWE Encrypted Content Encryption Key) || '.' || BASE64URL(JWE Initialization Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE Authentication Tag)

```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NN
In0.JdC7ydHPnn-l

rrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLLD4D0FtLm6f_S

-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8f0_D3-JNkkNp

MLUYZAWSE8ZgRclwOHL1KD6zhBIF-uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-
CJcxYPoJmjAeSdooahB

rsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto0kDQMDf_Tk2i
```

```
gBfslXYiTA.sjchiFykIZ9ab_OW.lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1I
gtHSs_mhBpOv8cYM

sRi6_IWlbJYH7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSr
gc80ZbYajZkaZZqf

TgY2QY3pEGmzpELnC65vQF2No6fHXdd3CewdoZHagyHFhBtqqoDRAw5q-
Naj5O36GJXT--l2D1yxBTN8

tZs7FNZKqRmFngPUj17wyM27vjqC8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW
-nTY-xm-fH6xUL6Z

QvFQnBa2jnJIsQ_WYIkEOddBd_0XNJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGC
ZO7WFFh580Af2cyO

e7vTD8irUm02qnxFGIMgOjLGbS871zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySY
HGIv70xJq1ZQQ1Mo

hxwJ_0jUkWcTVWa-n-
xDSESMGEH_1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIk

wB0xDIJOERB4EjRoWVWgYLGSduZbSpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jr
m95ot48_Chs2G7A8

JRXUl2sx9pbfayxktr0hBNmcVx3kPoww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn

9XLj8dD0aimhz6vPMOX4zAtdaWc03fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJ
PMDbC92G1oEc.4aT

8aykEG7TAVrBfS27qaw
```

Sending data to the Data Recipient

See: Sending Nested JWT HTTP Response

RSA Consumption

Decrypting JWE Payload

1. Decode the Compact Serialized JWE

JWE Protected Header

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ
```

JWE Encrypted Key

```
KOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe

psEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb

v04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
```

```
qgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
UklfCpIMfIjf7iGdXKHzg
```

JWE Initialization Vector

```
8V1_ALb6US04U3b
```

JWE Ciphertext

```
eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs9tX_EFShS8iB7j6ji
diwkIr3ajwQzaBtQD_A
```

JWE Authentication Tag

```
FBoMYUZodetZdvTiFvSkQ
```

2. Look up private key (key encryption key) associated with the kid from the protected header ```

```
 -----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAuRK7Tfo+7OBUcuI/Tk4Qnh0lamAnwMXK/Kcd5Sz1P9DJl4QE
4tY5Wv6IKMCbdYKX5muo8DjahR/ZtAv7UvPobprmmWU2201uLaLROmuhTJzWWFqE
JGhvtabwdr7/7JsXywQI1ZchzON4RUJaukqu0MsUnfVZEJAedYvUZBfY5Pa6GCl+
dgSV9vZ6nsFGcH/hmQU7qKrk4PrTNV4lc5ehO7Tkckn0BA9soDjBGfdxaYwxm9kU
jgkJMQYYIkm4gzPfLk7UlLhMRy9UWBoRvajGXz5vrif/0KIFKVQtjbt108rwC+/u
KVlmEHqybkanSG8T1tDL1BLKpKxwtZo0fsAURQIDAQABAoIBAQCyuNwYWWka5yem
KcZooAp8JjlTmLfK+Tck9V1xSxErJz0GDH+LbsTEkrh6YkW+HPcDlUP3d2/Ozws1
S7zQRqpW5U97Irru3L9hYrIacIW7rllvyTmCzzfRe/0LLzeGDd/UNXkyilghvCkQ
+RsUe7qF9xdZ0uzHieVgLkBUyJKzx8k40VfJE/2FdPZafdDMv1p+4CkDpxZUS7F6
MPFSuAumLJ3lSUey0B1+IpUZhZBaSjHVyBfUOWluOEtz99NjQKS/8I8lYetAJ+Mo
GkkXLW5sHRwkdhc050cuKp41BEOeByCtESH30vJLEvwbF4o2/vj6Z+v4NvpSZMmA
0/p02goBAoGBAPCn0kRjMKOxmYJVkZkQFHpVs6LH/J0SZY3K3bdJfiNmOYdYVyAn
NZQ9IvQ712AJTuVawkOhjBnTeMD5TUQvjV0WAVp9AIu8oV3vTcl/8oubZ9D4COt9
S+aJOTkwD8voH2MkQc7tNl7Qw31wcE+MGIKh3qigdgwifkhFzmaqpt1xAoGBAMTf
```

```
phXzMmkwKEW5v/ULLsOHS96x89izjebu8P8fWbuVgU0sK6n46Gfxb798hEd7gjU/

EzOSbNRIFyertRIiUUWcmeo6d9aqxbzfkc2GqEJ+Jym5QFSlJgxqxMtmTPnO5Bat

S58/Scl7TSqk5bE3j2yBR4rJEjur5V1SldByQYoVAoGATKpJ8/tdbWiQrNKxtX9H

5skSlxL6yNcpfwhXpaJGCuTwAswDxXx4Nyda0U+XB0Mv3SUSqhT22uthlqhVExnL

ARKXj8ouuFV5WsF3mG+oRw1U/19lCBA8c87Xaf6DqcPi6+SLCm7LWV1MSdPeE5lf

3Y3PrwyfTrJWZJPIczB+RCECgYBYoVbkCthnAoce3MDOUHp9DCvb1cExjaQUkv1r

3XFIQcY0N+5wVt5J7SehzSzAAZpc3kiGryTPbKT/9w1NXKW58QZZrHjG65qZrQy2

uiiFxsVaw0tyz+aRMH/oEeYVkE6e5uVki9lsG1ZiHFpLrfejoY/TqzHKK1jW6pcH

gGiBAQKBgAIeLlx91P4DLHuHBSmAWElqVVJf+nNQYBX7uAnftGlD+/PcyquPE/wL

eKgQzj6g0/kDM6Erex08hcw2bkc0ZULfhFFMt0NwGGhVGyt+eQOR1pkIJFLy72TX

WrsXAt3Nkhzw59ZyMT2hxpdEsxdeosoioJkDBBXaNcdeck4OLTNN

-----END RSA PRIVATE KEY-----
```

3. Use the private key (key encryption key) to decrypt the Encrypted JWE Content Encryption Key (Base 64 encoded)

```
sSm8_yt2AdOpBtcMtqx7Wv7FSQqG9vEaoVyK-iA5GCI
```

4. Use JWE Content Encryption Key, Initialization Vector and Authentication Tag to decrypt the ciphertext. You **SHOULD** be left with plaintext that is a compact serialized JWS.

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyJ9

.eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJ
KYW5lIiwibGFzdCI

6IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWwuY29tIl19.n7zJKh
erBaBL4UACye3j41

AK48GZwGKHS383G2ZSBTqPUeh4_wXckngpRWqPP5y5O6DGaP34f_24LKAZ1KKpGh
V_HgWKSghTIDR6av

F_SHF0yMlHeU7PZZfyzy4OyLh0UCa92GQn3e_l7VK6m4rCjSBLS58vTx6g_QMoqY
Fuwbt9mGviO1aPbq

AZT-e-
NvQuZDwX1TJsqks7QG8zSEOB4KHgjHvLvh1wIcQAgXW5Y_qa2Txh3MqkUxAT3oGU
7ip9EFL6ZP
```

```
mwbVruf738cQVybmsCP3YH_AMF3twFPkJ56UzZBKj4HeFlFp-N3-
x6j8yHDdmaFypc5rQWWVTK9sxMug
```

Verifying JWS Payload

1. Decode the compacted payload extracting out the JWS Protected Header, Payload and Signature

JWS Protected Header

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyJ9
```

JWS Payload

```
eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJK
YW5lIiwibGFzdCI6

IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWwuY29tIl19
```

JWS Signature

```
n7zJKherBaBL4UACye3j41AK48GZwGKHS383G2ZSBTqPUeh4_wXckngpRWqPP5y5
O6DGaP34f_24LKAZ

1KKpGhV_HgWKSghTIDR6avF_SHF0yMlHeU7PZZfyzy4OyLh0UCa92GQn3e_l7VK6
m4rCjSBLS58vTx6g

_QMoqYFuwbt9mGviO1aPbqAZT-e-
NvQuZDwX1TJsqks7QG8zSEOB4KHgjHvLvh1wIcQAgXW5Y_qa2Txh

3MqkUxAT3oGU7ip9EFL6ZPmwbVruf738cQVybmsCP3YH_AMF3twFPkJ56UzZBKj4
HeFlFp-N3-x6j8yH

DdmaFypc5rQWWVTK9sxMug
```

2. Retrieve the public key associated with the kid from the protected header. The public key can either be in a local store or can be retrieved from signed certificate via a JWKS endpoint registered for the data provider.

```
-----BEGIN CERTIFICATE-----

MIIDhzCCAm+gAwIBAgIBADANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJCRTEN

MAsGA1UECgwEVGVzdDENMAsGA1UECwwEVGVzdDENMAsGA1UEAwwEVGVzdDAeFw0y

MTAxMjQyMjI0NTlaFw0yMjAxMjQyMjI0NTlaMDoxCzAJBgNVBAYTAkJFMQ0wCwYD

VQQKDARUZXN0MQ0wCwYDVQQLDARUZXN0MQ0wCwYDVQQDDARUZXN0MIIBIjANBgkq

hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA40b3nY6KgCBekJVfj3MbbXEMvcDFz1ew

97LQ2B8AuQi3KhtlRpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAnpAM8UJdmW2jt
```

```
vUyE+7WjmEZkW9E+xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/16wRZ1HonV6Oj
XOjp5fF2og4OhL3S2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5AYbGEVoP2VKM
e2KKQIBypqawxBIlqVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+a7e66pWlk7Xf
ykZ5Bg9/dm+1aZPc653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi4QIDAQABo4GX
MIGUMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFJ45lQ9T0P52egkfVdB6h/Wj
SIpUMGIGA1UdIwRbMFmAFJ45lQ9T0P52egkfVdB6h/WjSIpUoT6kPDA6MQswCQYD
VQQGEwJCRTENMAsGA1UECgwEVGVzdDENMAsGA1UECwwEVGVzdDENMAsGA1UEAwwE
VGVzdIIBADANBgkqhkiG9w0BAQUFAAOCAQEAvk3PD1Sbo/h2AR+twDv+LrQOyyN7
GkElF28goFhOiVBkGUmVvfy7bQEZa3q7RTUoe2pWqRdWQt1UkFcOGLWWtTdfDrak
9PJyMCIRQenD5DqdqwMmuPscy9y9GLJ2VV7KLzhTYQk5Og6LCn0PfWSLp9Yb3WCz
+AhBCs13N497DMBrLqwaLDITjnWNnXmv4AreiOqrvPJRY8acHeAVYmNph5Ine/Fe
nXTsGCHj7ERgZ8dD3+kvu7Ia8h4m+JbA1SeZajnKP34lKW7DucaHzgbUjVTj14oG
dZv+ZZAZ6QgS+xAHmVkKqX7SNNGEfmuzS4j6p9AMoy/77jkRnPSxy0Nnzw==
-----END CERTIFICATE-----
```

3. Confirm that the public key is valid.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA40b3nY6KgCBekJVfj3Mb
bXEMvcDFz1ew97LQ2B8AuQi3KhtlRpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAn
pAM8UJdmW2jtvUyE+7WjmEZkW9E+xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/1
6wRZ1HonV6OjXOjp5fF2og4OhL3S2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5
AYbGEVoP2VKMe2KKQIBypqawxBIlqVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+
a7e66pWlk7XfykZ5Bg9/dm+1aZPc653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi
4QIDAQAB
-----END PUBLIC KEY-----
```

a. Confirm that it is signed by a trusted certificate chain

b. Confirm that it has not expired

c. Confirm that it has not been revoked

4. Verify the signature using the public key

    a. Verify issued at date

    b. Verify additional claims if applicable (Examples)

Nested JWT using Elliptic Curve

Overview

This section provides implementation details for using EC algorithms for JSON Web Signatures and JSON Web Encryption.

RECOMMENDED Curves

Per Section 6.2.1.1 from RFC 7518, acceptable curves for ECDH-ES key agreement are P-256, P-384, P-521.

- P-256 uses prime256v1 elliptic curve
- P-384 uses secp384r1 elliptic curve
- P-521 uses secp521r1 elliptic curve

REQUIRED Elements/Minimum Standards

HTTP Headers

The following fields in the JOSE header for the JWS are allowed.

	Parameter	Required	Description
1	alg	**MUST**	Identifies the *cryptographic algorithm* used to encrypt or determine the value of the CEK.
2	enc	**MUST**	Identifies the *content encryption algorithm* used to perform authenticated encryption on the plaintext to produce the ciphertext and the AuthenticationTag.
3	zip	OPT	Identifies the *compression algorithm* applied to the plaintext before encryption, if any.
4	jku	OPT	Contains the *JWK Set URL*, a URI that refers to a resource for a set of JSON-encoded public keys, one of which corresponds to the public key with which the JWE was encrypted. An HTTP GET request to retrieve the JWK Set **MUST** use TLS (1.2 at a minimum, 1.3 is now recommended at the time of FDX API v6.0). The identity of the server **MUST** be validated by JWS recipient. jku **SHOULD** be included if multiple public keys are exposed in the JWK Set URL.
5	jwk	OPT	Contains the *JSON Web Key*, the public key with which the JWE was encrypted

	Parameter	Required	Description
6	kid	OPT	Contains the *Key ID*, a hint indicating the public key with which the JWE was encrypted. This parameter allows originators to explicitly signal a change of key to JWE recipients.
7	x5u	OPT	Contains the *X.509 public key certificate* or *certificate chain* RFC5280, the public key with which the JWE was encrypted.
8	x5c	OPT	Contains the *X.509 public key certificate* or *certificate chain* RFC5280 corresponding to the public key with which the JWE was encrypted. The certificate or certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded (not base64url-encoded) DER PKIX certificate value. The certificate containing the public key corresponding to the key used to encrypt the JWE **MUST** be the first certificate. The recipient **MUST** validate the certificate chain and consider the certificate or certificate chain to be invalid if any validation failure occurs.
9	x5t	OPT	Contains the *X.509 certificate SHA-1 thumbprint*, a base64url-encoded SHA-1 message digest of the DER encoding of the X.509 certificate corresponding to the public key with which the JWE was encrypted. Certificate thumbprints are also known as certificate fingerprints.
10	x5t#S256	OPT	Contains the *X.509 certificate SHA-256 thumbprint*, a base64url-encoded SHA-256 message digest of the DER encoding of the X.509 certificate corresponding to the public key with which the JWE was encrypted.
11	typ	OPT	Contains the *type*, used by JWE applications to declare the media type of this complete JWE. The "typ" value "JOSE+JSON" can be used to indicate that this object is a JWS or JWE using the JWS or the JWE JSON Serialization. Other type values can also be used by applications.
12	cty	OPT	Contains the *content type*, used by JWE application to declare the media type of the secured content (the plaintext).
13	crit	OPT	Contains the *critical* Header Parameter to indicate that extensions to this specification and/or JWA are being used that **MUST** be understood and processed. Its value is an array listing the Header Parameter names present in the JOSE Header that use those extensions. If any of the listed extension Header Parameters are not understood and supported by the recipient, then the JWE is invalid.
14	epk, apu, apv	COND	When alg is one of ECDH-ES, ECDH-ES+A128KW or ECDH-ES+A256KW algorithms, then "epk", "apu", "apv" headers **MUST** be added.

JWS Claims

As JWS is used only for signature validation, no additional claims are added. This example shows how to create an EC certificate request used in ECDH key agreement.

How to create an EC Certificate Request

1. OpenSSL config for EC certificate request

```
# openssl.conf


[req]

req_extensions = v3_req

distinguished_name  = req_distinguished_name


[v3_req]

basicConstraints = CA:FALSE

keyUsage=keyAgreement


[req_distinguished_name]

countryName                     = Country Name (2 letter code)

countryName_default             = US


stateOrProvinceName             = State or Province Name (full
name)

stateOrProvinceName_default     = IL


localityName                    = Chicago


0.organizationName              = Organization Name (eg,
company)

0.organizationName_default      = Internet Widgits Pty Ltd


organizationalUnitName          = Organizational Unit Name (eg,
section)
```

```
commonName                      = Common Name (e.g. server FQDN
or YOUR name)

commonName_max                  = 64


emailAddress                    = Email Address

emailAddress_max                = 64
```

2. Generate EC private key

```
`openssl ecparam -out server.key -name prime256v1 -genkey -
config openssl.conf`
```

3. Generate CSR

```
`openssl req -new -key server.key -out server.csr -sha256 -
config openssl.conf`
```

4. Verify signature on CSR

```
openssl req -text -noout -verify -in server.csr


verify OK

Certificate Request:

    Data:

        Version: 1 (0x0)

        Subject: C = US, ST = UT, O = Internet Widgits Pty Ltd

        Subject Public Key Info:

            Public Key Algorithm: id-ecPublicKey

                Public-Key: (256 bit)

                pub:


04:97:08:cd:34:34:8c:2b:7e:3e:db:c2:fc:4e:b3:


78:d5:7e:ee:4a:31:f9:4c:c6:d0:ac:b0:a4:2d:76:
```

```
3d:9c:62:03:16:69:26:ae:c8:4a:43:f9:0c:59:c5:

3d:79:a5:7f:9a:73:48:09:db:27:96:47:8a:97:c0:
                    b3:e0:13:d0:45
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Key Agreement
    Signature Algorithm: ecdsa-with-SHA256
         30:46:02:21:00:a1:07:38:2f:7d:fa:31:5c:07:51:8c:23:52:
         84:0e:c3:cb:ef:5c:5b:88:1d:5e:9a:59:50:77:84:55:b1:31:
         49:02:21:00:b6:0b:39:f3:70:ec:28:f3:71:5f:d3:39:82:5f:
         5c:26:9e:31:99:39:a6:2b:67:52:0e:f7:93:d7:55:1a:c4:4a
```

5. Follow the prompts and enter the information related to your organization

EC: JSON Web Signature

Per FDX RFC 0011 Message Encryption, an RSA signing algorithm or an EC signing algorithm **MAY** also be used for the JWS that is then encrypted to form the JWE using an EC encryption algorithm. This example shows how to create a JWS with ECDSA using P-256 and SHA-256 (ES256). The protected header and payload **MUST** be base64URL-encoded in the same way as the RSA example.

JWK

```
{
  "kty":"EC",
  "use":"enc",
```

```
  "crv":"P-256",
  "kid":"a152e75b-8e11-4f87-908c-366552610467",
  "x":"YoUQaMiqgeSmNMXSqcYiFRO4nZEXV47APaUnm4PyaPg",
  "y":"iUuwma37B6YFPKLA7tv_-cczuT4WJf2WmVfsgELHr1c"
}
```

EC Content Encryption Key & Key Encryption Key Generation

1. The Sender and Receiver agree on an elliptic curve.

```
Curve: P-256
```

2. Data Recipient generates static EC keypair using an agreed-upon elliptic curve with Data Provider. Receiver provides their public EC key as trusted CA signed certificate via their JWK URL.

Receiver Private Key

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFoxJaJ8UtevP3F+dNWkax6neQTjeipwAewO0K/N+8SqoAoGCCqGSM49
AwEHoUQDQgAEEWXLOwDqRhiiSHkd3GCzpVZ4d1ucLH9mzOWZNWunILdrS3vkVxf4
/odFGtCYEW+y05MhkVBGVg8OCWyQHrvoJA==
-----END EC PRIVATE KEY-----
```

Receiver Public Key

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEEWXLOwDqRhiiSHkd3GCzpVZ4d1uc
LH9mzOWZNWunILdrS3vkVxf4/odFGtCYEW+y05MhkVBGVg8OCWyQHrvoJA==
-----END PUBLIC KEY-----
```

3. Sender generates ephemeral (temporary) EC keypair using the same agreed-upon elliptic curve.

Sender Private Key

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEILdwODe0rFBedZiW01KQEOZYnAwhR6k8jx9iRg8PuLN0oAoGCCqGSM49
```

```
AwEHoUQDQgAESW0RyC+JYmzdVolujnxEZDz+c8hgtUw8rfq5ELAT71IitgQ99l2h

hZvJ2GqM8nIq/TTwFJEfz39Pq1LB8ErWWA==

-----END EC PRIVATE KEY-----
```

Sender Public Key

```
-----BEGIN PUBLIC KEY-----

MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAESW0RyC+JYmzdVolujnxEZDz+c8hg

tUw8rfq5ELAT71IitgQ99l2hhZvJ2GqM8nIq/TTwFJEfz39Pq1LB8ErWWA==

-----END PUBLIC KEY-----
```

4. Sender generates ephemeral 256 bit key used to AES-encrypt the Content Encryption Key (CEK).

```
sSm8_yt2AdOpBtcMtqx7Wv7FSQqG9vEaoVyK-iA5GCI
```

5. Sender computes a shared secret using their own ephemeral private EC key and Receiver's static public EC key.

```
EaYbmvnPGxXREBtu0o5X6OzQNZelwx2d-v_ty-2Ivrs
```

6. Sender uses the shared secret from step 5 and derives a 256 bit AES Key using Concat KDF. This key is used as the Key Encryption Key (KEK).

```
Ie1XYS7XcaqYztrtzot0SXmtXqLS40k1izIuq2RuVRY
```

EC Encryption

Note: This example uses an Elliptic Curve signed JWS. You may see signed payloads encrypted using other algorithms such as RSA.

1. Sender uses the 256 bit AES key (KEK) derived in EC CEK & KEK Generation step 5 to encrypt the CEK (Content Encryption Key) from EC CEK & KEK Generation Step 3.

```
6ZKl9aHBfd7hFFlUBmr16PGRHt2po8teuDBZzVTi0AYGpPHw87yBVg
```

2. Sender encrypts sensitive data in the API response using the 256 bit AES key from EC CEK & KEK Generation Step 3.

Plaintext

```
eyJhbGciOiJFUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyJ9
```

```
.eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJ
Kb2huIiwibGFzdCI

6IlNtaXRoIn0sImVtYWlsIjpbImpzbWl0aEBlbWFpbC5jb20iXX0.1tQHVkd0aU5
EpE-oyY_R6tJjnYq

n2OQtprp8vKVZYtwYgzR6BdHPsi4xhgxoEq8iTAGJqjLdi1UnMLyPVABUYw
```

Encrypted

```
lA9EWgdnWrJ0pfHgR8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7

vIfpud6ZLW6Ma8sNVna00ptIAV5KvWeQIAO6-
bY8Ev47l7hPILaMawULTomzPg_HsWz2NvRqclrRn5nE

RNJOef0yjJWCgBfe2CRuj4r7XWpr1ei8xsCmA4nA__JmDiimFhhWtuYrL61SgSig
mCj9-Vfd5tL5rgGg

9XC4UX3JMDw6P-qHthR-hCZrjxUbDJcW_NWJnHvjkRjgT1qzJLCmRf0A-g-
kthE8sQn4Bfdkf7FbVOUs

SJQccKeAyFuUIKrhRM5xRFhnv64aQLxKaQCleujML15K8l8YweqCf4XSUc-
qNVsrgVRuF8UAOXLAYEM
```

3. Sender creates JWE containing encrypted sensitive data, encrypted CEK , ephemeral public EC key of Sender and some other additional material.

Protected Header

```
{
  "alg": "ECDH-ES+A256KW",
  "cty": "JWT",
  "enc": "A256GCM",
  "epk": {
    "crv": "P-256",
    "kty": "EC",
    "x": "SW0RyC-JYmzdVolujnxEZDz-c8hgtUw8rfq5ELAT71I",
    "y": "IrYEPfZdoYWbydhqjPJyKv008BSRH89_T6tSwfBK1lg"
  }
}
```

JWE JSON Serialization


```
{

  "ciphertext":

"lA9EWgdnWrJ0pfHgR8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJY
H7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAV5KvWeQIAO6-
bY8Ev47l7hPILaMawULTomzPg_HsWz2NvRqclrRn5nERNJOef0yjJWCgBfe2CRuj
4r7XWpr1ei8xsCmA4nA__JmDiimFhhWtuYrL61SgSigmCj9-
Vfd5tL5rgGg9XC4UX3JMDw6P-qHthR-
hCZrjxUbDJcW_NWJnHvjkRjgT1qzJLCmRf0A-g-
kthE8sQn4Bfdkf7FbVOUsSJQccKeAyFuUIKrhRM5xRFhnv64aQLxKaQCleujML15
K8l8YweqCf4XSUc-qNVsrgVRuF8UAOXLAYEM",

  "iv": "sjchiFykIZ9ab_OW",

  "encrypted_key":
"6ZKl9aHBfd7hFFlUBmr16PGRHt2po8teuDBZzVTi0AYGpPHw87yBVg",

  "tag": "JUhDoBNzNkDVZcCn2hwf8Q",

  "protected":

"eyJhbGciOiJFQ0RILUVTK0EyNTZLVyIsImN0eSI6IkpXVCIsImVuYyI6IkEyNTZ
HQ00iLCJlcGsiOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJTVzBSeUM
tSlltemRWb2x1am54RVpEei1jOGhndFV3OHJmcTVFTEFUNzFJIiwieSI6IklyWUV
QZlpkb1lXYnlkaHFqUEp5S3YwMDhCU1JIODlfVDZ0U3dmQksxbGcifX0"

}
```


4. Sender sends JWE to Receiver

HTTP Header

```
Content-Type: application/jwt
```

HTTP Body


```
eyJhbGciOiJFQ0RILUVTK0EyNTZLVyIsImN0eSI6IkpXVCIsImVuYyI6IkEyNTZH
Q00iLCJlcGsiOnsi

Y3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJTVzBSeUMtSlltemRWb2x1am54
RVpEei1jOGhndFV3

OHJmcTVFTEFUNzFJIiwieSI6IklyWUVQZlpkb1lXYnlkaHFqUEp5S3YwMDhCU1JI
ODlfVDZ0U3dmQksx

bGcifX0.6ZKl9aHBfd7hFFlUBmr16PGRHt2po8teuDBZzVTi0AYGpPHw87yBVg.s
jchiFykIZ9ab_OW.
```

```
lA9EWgdnWrJ0pfHgR8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7

vIfpud6ZLW6Ma8sNVna00ptIAV5KvWeQIAO6-
bY8Ev47l7hPILaMawULTomzPg_HsWz2NvRqclrRn5nE

RNJOef0yjJWCgBfe2CRuj4r7XWpr1ei8xvvDXpDA__JmDiimFhhWtuYrL61SgSig
mCj9-Vfd5tL5rgGg

9XC4UX3JMDwoP9C94BYJ9zptnzsqNIMS_tbs22fi-
xG6UVXoKMq0GtVgnCvzuAwckTzLY_NMcM4sD8YT

Jok5bNzL-TG0YYHgXeZNZ09WmL4SAYkOejGkde3WZWwT334e-
Pmhbr2FfvvLb3ZzvVRRMN4gGCz0aC90

4Qthpw.cYfrBMaS_NXftFKiGliS5Q
```

EC Consumption

JWE Decryption

1. Receiver retrieves its static private EC key from Step 1 and ephemeral public EC key from Sender to compute shared secret which will be same as shared secret in EC CEK & KEK Generation Step 4.

Data Recipient's Private Key

```
-----BEGIN EC PRIVATE KEY-----

MHcCAQEEIFoxJaJ8UtevP3F+dNWkax6neQTjeipwAewO0K/N+8SqoAoGCCqGSM49

AwEHoUQDQgAEEWXLOwDqRhiiSHkd3GCzpVZ4d1ucLH9mzOWZNWunILdrS3vkVxf4

/odFGtCYEW+y05MhkVBGVg8OCWyQHrvoJA==

----END EC PRIVATE KEY-----
```

2. Receiver derives a 256 bit AES Key (KEK) from shared secret.
3. Receiver uses the 256 bit AES key(KEK) to decrypt the CEK.
4. Receiver uses the CEK to decrypt sensitive data payload (e.g. PII data of customers).

Sending Nested JWT HTTP Response

The JWE **SHOULD** be sent to the Data Recipient using the compact serialized form and with a content type of `application/jwt`

```
Content-Type: application/jwt
```

```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NN
In0.JdC7ydHPnn-l

rrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLLD4D0FtLm6f_S

-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8f0_D3-JNkkNp

MLUYZAWSE8ZgRclwOHL1KD6zhBIF-uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-
CJcxYPoJmjAeSdooahB

rsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto0kDQMDf_Tk2i

gBfslXYiTA.sjchiFykIZ9ab_OW.lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1I
gtHSs_mhBpOv8cYM

sRi6_IWlbJYH7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSr
gc80ZbYajZkaZZqf

TgY2QY3pEGmzpELnC65vQF2No6fHXdd3CewdoZHagyHFhBtqqoDRAw5q-
Naj5O36GJXT--l2D1yxBTN8

tZs7FNZKqRmFngPUj17wyM27vjqC8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW
-nTY-xm-fH6xUL6Z

QvFQnBa2jnJIsQ_WYIkEOddBd_0XNJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGC
ZO7WFFh580Af2cyO

e7vTD8irUm02qnxFGIMgOjLGbS871zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySY
HGIv70xJq1ZQQ1Mo

hxwJ_0jUkWcTVWa-n-
xDSESMGEH_1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIk

wB0xDIJOERB4EjRoWVWgYLGSduZbSpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jr
m95ot48_Chs2G7A8

JRXUl2sx9pbfayxktr0hBNmcVx3kPoww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn

9XLj8dD0aimhz6vPMOX4zAtdaWc03fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJ
PMDbC92G1oEc.4aT

8aykEG7TAVrBfS27qaw
```

Error Handling

1. See table below.
2. Errors **SHOULD** be returned unencrypted. (Assuming they do not include any PII or other sensitive information.)

Nested JSON Web Token Java code using Nimbus

Introduction

A Nested JSON Web Token (JWT) is a token that involves a signed and encrypted payload. The inner token is a signed payload of the JSON Web Signature (JWS) type, and the outer token encapsulates the signed token by encrypting it into a JSON Web Encryption (JWE) payload.

An example Nested JWT is provided using the Nimbus library: Nested JWT - JWS within a JWE.

Sample Solution

This sample routine generates an elliptic curve key pair for signing and encryption before the function is called. It then builds the Nested JWT, and then breaks it down by decrypting and then validating the signature.

The code sample below uses Elliptical Curve cryptography (ECC) with P-256 curve ECC keys for signing and encryption.

Item	Algorithm
Signing	ES256 - Sign the payload with the ECC signing key and hash the signature output with SHA256
Asymmetric Encryption (Key encryption)	ECDH-ES or ECDH_ES_A256K using Concat KDF and CEK wrapped with "A256KW"
Symmetric Encryption (Payload)	AES-256 GCM

Nested JWT (JWS encapsulated in JWE) using Nimbus

Java version 11.

Imports

```
package com.jpmc.crypto;


import java.io.UnsupportedEncodingException;

import java.security.*;

import java.security.interfaces.RSAPublicKey;
```

```
import java.util.*;

import com.nimbusds.jose.*;

import com.nimbusds.jose.crypto.RSADecrypter;

import com.nimbusds.jose.crypto.RSAEncrypter;

import com.nimbusds.jwt.EncryptedJWT;

import com.nimbusds.jwt.JWTClaimsSet;
```

Dependency

In the pom.xml file add this dependency.

```
<dependency>

    <groupId>com.nimbusds</groupId>

    <artifactId>nimbus-jose-jwt</artifactId>

    <version>6.0.2</version>

</dependency
```

Sample code

```
protected static void RunNimbusNestedJWTSignFirstEC( ECKey
ECEncryptionKey,

  ECKey ECSigningKey) {

try {

  if (ECEncryptionKey != null && ECSigningKey != null) {

    // Nimbus implementation

    //Create a Signing key, see functions below.

    ECKey nimbusECSenderSigningKey = ECSigningKey;



    // Create an encryption key pair, see functions below for
generation.

    ECKey nimbusECEncryptionKey = ECEncryptionKey;
```

```
    //holds the public key of the asymmetric encryption key

    ECKey nimbusECEncryptionPublicKey =
nimbusECEncryptionKey.toPublicJWK();



    // Create an encryption key pair, see functions below.

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC
Encryption JWK: "

      + nimbusECEncryptionPublicKey.toString());



    // Create Signed JWT with EC P-256 DSA with SHA-256

    SignedJWT signedJWT = new SignedJWT(

            new JWSHeader

                    .Builder(JWSAlgorithm.ES256)

                    .keyID(nimbusECSenderSigningKey.getKeyID())

                    .build(),

            new JWTClaimsSet.Builder()

                    .subject("sally")

                    .issueTime(new Date())

                    .issuer("secure.chase.com")

                    .build());



    // Sign the JWT

    signedJWT.sign(new ECDSASigner(nimbusECSenderSigningKey));



    System.out.println("NimbusNestedJWT-SignThenEncrypt EC: JWS:
"

      + signedJWT.serialize());
```

```
    // Create JWE object with signed JWT as payload

    JWEObject jweObject = new JWEObject(

            new JWEHeader

                    .Builder(JWEAlgorithm.ECDH_ES_A256KW,
EncryptionMethod.A256GCM)

                    .contentType("JWT") // required to indicate
nested JWT

                    .build(),

            new Payload(signedJWT));



    // Encrypt with the public key

    jweObject.encrypt(new
ECDHEncrypter(nimbusECEncryptionPublicKey));



    // Serialise to JWE compact form

    String jweString = jweObject.serialize();



    System.out.println("NimbusNestedJWT-SignThenEncrypt EC: JWE
Encryption method: "

      + " JWE algorithm " +
jweObject.getHeader().getEncryptionMethod() + " "

      + jweObject.getHeader().getAlgorithm() + " " + " JWE
string " + jweString);



    System.out.println("NimbusNestedJWT-SignThenEncrypt EC: JWE
Encryption method: "

      + "JWE header " + jweObject.getHeader());



    //Now consume the Nested JWT
```

```
    // Parse the JWE string

    JWEObject jweObjectEncrypted = JWEObject.parse(jweString);



    // Decrypt with private key

    jweObjectEncrypted.decrypt(new
ECDHDecrypter(nimbusECEncryptionKey));



    // Extract payload

    SignedJWT signedJWTAfter =
jweObjectEncrypted.getPayload().toSignedJWT();

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC:
After decryption payload"

      + " (will contain JWS): " + signedJWTAfter.serialize());



    assertNotNull("Payload not a signed JWT EC ", signedJWT);



    // Check the signature

    assertTrue(signedJWTAfter.verify(new
ECDSAVerifier(nimbusECSenderSigningKey)));

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC:
After decryption check"

      + " signature of JWS: "

      + signedJWTAfter.verify(new
ECDSAVerifier(nimbusECSenderSigningKey)));



    System.out.println("NimbusNestedJWT-SignThenEncrypt EC:
After decryption subject"

      + " JWS: " +
signedJWTAfter.getJWTClaimsSet().getSubject());
```

```
  }

    }

    catch (Exception e) {

        System.out.println("NimbusNestedJWT-SignThenEncryptEC:
exception: " + e);

    }

}



protected static ECKey CreateNimbusECEncryptionKey() {

    ECKey jwkECEncryption = null;



    try {

        // Generate EC key pair in JWK format

        jwkECEncryption = new ECKeyGenerator(Curve.P_256)

                .keyUse(KeyUse.ENCRYPTION) // indicate the
intended use of the key

                .keyID(UUID.randomUUID().toString()) // give the
key a unique ID

                .generate();



    } catch (JOSEException e) {

        e.printStackTrace();

    }

    return jwkECEncryption;

}



protected static ECKey CreateNimbusECSigningKey() {

    ECKey jwkECSigning = null;
```

```
    try {
        // Generate EC key pair in JWK format
        jwkECSigning = new ECKeyGenerator(Curve.P_256)
                .keyUse(KeyUse.SIGNATURE) // indicate the
intended use of the key
                .keyID(UUID.randomUUID().toString()) // give the
key a unique ID
                .generate();

    } catch (JOSEException e) {
        e.printStackTrace();
    }
    return jwkECSigning;
}
```

Nested JWT output

Below is the output of the above routine. Note the epk, ephemeral public key.

```
NimbusNestedJWT-SignThenEncrypt EC Encryption JWK:
{"kty":"EC","use":"enc","crv":"P-256","kid":"a152e75b-8e11-4f87-
908c-366552610467",
"x":"YoUQaMiqgeSmNMXSqcYiFRO4nZEXV47APaUnm4PyaPg",
"y":"iUuwma37B6YFPKLA7tv_-cczuT4WJf2WmVfsgELHr1c"}
NimbusNestedJWT-SignThenEncrypt EC: JWS:
eyJraWQiOiI1ZDhhNTdiNi00ZjRmLTQ5NTItYTY5YS
1hYjcwNTU2NzkzYjAiLCJhbGciOiJFUzUxMiJ9.eyJzdWIiOiJzYWxseSIsImlzc
yI6InNlY3VyZS5jaGFz
ZS5jb20iLCJpYXQiOjE2MDIxNzQ1ODJ9.AYBIyjeNTPEZi_Vgfg2J9FpqL59coK5
xBU_YJaxwQg-5UCfE9p
```

```
MYwoBHR7gJZqYMMPtFpPkcxzZhIFFTElBoPz7mARRS-
r8N1iNUKbMP547qay4MIF-BtOxpyX8uTsint5cjx

hxtP8gsV-MEh-YXs89SNeZJo-cr_41KXr3_9XLUHrPo

NimbusNestedJWT-SignThenEncrypt EC: JWE Encryption method:  JWE
algorithm A256GCM

ECDH-ES+A256KW  JWE string
eyJlcGsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiIzbWt

nWjFDNXFxY19mQnhwc3NaeTRFSlMxVmFScGd2dS14aHFzenl1azYwIiwieSI6Im1
wQUVPRk1IamdCRk1GM

DN2bG84YnoweXFrR1NQajNna1RZUmtsV3BXX0kifSwiY3R5IjoiSldUIiwiZW5jI
joiQTI1NkdDTSIsImF

sZyI6IkVDREgtRVMrQTI1NktXIn0.YdWec3hyjvLSSDjjq4uZcBOG18BTZPO7L79
6v_3ytUmG9CyjWfjhZ

w.E8t4MircnwButjA1.a4w2_IRmxykljL7qfgXHEoz-my_g6qNFgJNdGTX_fi-
xAF8mJSG1gNSJzlLPq_X

_3npxPUmKbROmJmc3DOKxHukpElMVlWhhYo1XYHX9CwirZw77nO8McdTR9LNdPvF
bsjEwQ3QiamF56EPfl

zM7ABS7FBAqYaIwznBXJA_Vqvixtc-
KJOd1kIGcj9WqZ7xReFbk2KeRJEvya1l93Ajzia1eACcSyXEKnXR

wU23X_BPRQfv4eGHPoeU3tZwIiSAVV51oYHhcK8KSM4mmrHZ5doC0puJHqJnCgGS
OBiBcwNrBfSKYK7Y1i

djTwoRkHmWcvOOkJLfhFLTeixrPUVP0rFRdePItvb8w-
jrUBrA3yMhM5VbXCdrPkH7WwhIiHffdKDZ38eu

hoSwo3AyreUqBUA6OLu46J8vS0iGJHm4xCOTOXih9eZntrJhDv5klYg.i5IY3tmt
TDbqPdGGV4_H5g

NimbusNestedJWT-SignThenEncrypt EC: JWE Encryption method:

JWE header

{"epk":

{"kty":"EC","crv":"P-256",

"x":"3mkgZ1C5qqc_fBxpssZy4EJS1VaRpgvu-xhqszyuk60",

"y":"mpAEOFMHjgBFMF03vlo8bz0yqkGSPj3gkTYRklWpW_I"}

,"cty":"JWT",
```

```
"enc":"A256GCM",

"alg":"ECDH-ES+A256KW"}

NimbusNestedJWT-SignThenEncrypt EC: After decryption payload
(will contain JWS):

eyJraWQiOiI1ZDhhNTdiNi00ZjRmLTQ5NTItYTY5YS1hYjcwNTU2NzkzYjAiLCJh
bGciOiJFUzUxMiJ9.

eyJzdWIiOiJzYWxseSIsImlzcyI6InNlY3VyZS5jaGFzZS5jb20iLCJpYXQiOjE2
MDIxNzQ1ODJ9.AYBI

yjeNTPEZi_Vgfg2J9FpqL59coK5xBU_YJaxwQg-
5UCfE9pMYwoBHR7gJZqYMMPtFpPkcxzZhIFFTElBoP

z7mARRS-r8N1iNUKbMP547qay4MIF-BtOxpyX8uTsint5cjxhxtP8gsV-MEh-
YXs89SNeZJo-cr_41KXr

3_9XLUHrPo

NimbusNestedJWT-SignThenEncrypt EC: After decryption check
signature of JWS: true

NimbusNestedJWT-SignThenEncrypt EC: After decryption subject
JWS: sally

RunNimbusNestedJWTSignFirst EC: Total ns: 32099895- In ms
32.099895
```

Exhibits

Exhibit A : Financial-grade API Security Profile 1.0 - Part 2: Advanced

Note: Content in this section may also be found at FAPI 1.0 PART2: Advanced Security Profile (final), and is provided here in this exhibit for convenience.

Foreword

The OpenID Foundation (OIDF) promotes, protects and nurtures the OpenID community and technologies. As a non-profit international standardizing body, it is comprised by over 160 participating entities (workgroup participants). The work of preparing implementer drafts and final international standards is carried out through OIDF workgroups in accordance with the OpenID Process. Participants interested in a subject for which a workgroup has been established have the right to be represented in that workgroup. International organizations, governmental and non-governmental, in liaison with OIDF, also take part in the work. OIDF collaborates closely with other standardizing bodies in the related fields.

Final drafts adopted by the Workgroup through consensus are circulated publicly for the public review for 60 days and for the OIDF members for voting. Publication as an OIDF Standard requires approval by at least 50 % of the members casting a vote. There is a possibility that some of the elements of this document may be the subject to patent rights. OIDF **SHALL NOT** be held responsible for identifying any or all such patent rights.

Financial-grade API Security Profile 1.0 consists of the following parts:

- **Financial-grade API Security Profile 1.0 - Part 1: Baseline**
- **Financial-grade API Security Profile 1.0 - Part 2: Advanced**

These parts are intended to be used with **RFC6749**, **RFC6750**, **RFC7636**, and **OIDC**.

Introduction

The Financial-grade API is a highly secured OAuth profile that aims to provide specific implementation guidelines for security and interoperability. The Financial-grade API security profile can be applied to APIs in any market area that requires a higher level of security than provided by standard **OAuth** or **OpenID Connect**. Among other security enhancements, this specification provides a secure alternative to screen scraping. Screen scraping accesses user's data and functions by impresonating a user through password sharing. This brittle, inefficient, and insecure practice creates security vulnerabilities which require financial institutions to allow what appears to be an automated attack against their applications.

This document is Part 2 of FAPI Security Profile 1.0 that specifies an advanced security profile of OAuth that is suitable to be used for protecting APIs with high inherent risk.

Examples include APIs that give access to highly sensitive data or that can be used to trigger financial transactions (e.g., payment initiation). This document specifies the controls against attacks such as: authorization request tampering, authorization response tampering including code injection, state injection, and token request phishing. Additional details are available in the security considerations section.

Although it is possible to code an OpenID Provider and Relying Party from first principles using this specification, the main audience for this specification is parties who already have a certified implementation of OpenID Connect and want to achieve a higher level of security. Implementers are encouraged to understand the security considerations contained in Section 8.7 before embarking on a 'from scratch' implementation.

Notational Conventions

The keywords "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**MAY**", and "**CAN**" in this document are to be interpreted as described in **ISO Directive Part 2**. These keywords are not used as dictionary terms such that any occurrence of them **SHALL** be interpreted as keywords and are not to be interpreted with their natural language meanings.

Table of Contents

---

1. Scope

This part of the document specifies the method of

- applications to obtain the OAuth tokens in an appropriately secure manner for higher risk access to data;
- applications to use OpenID Connect to identify the customer; and
- using tokens to interact with the REST endpoints that provides protected data;

This document is applicable to higher risk use cases which includes commercial and investment banking and other similar industries.

---

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applied. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISODIR2 - ISO/IEC Directives Part 2

RFC6749 - The OAuth 2.0 Authorization Framework

RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage

RFC7636 - Proof Key for Code Exchange by OAuth Public Clients

RFC6819 - OAuth 2.0 Threat Model and Security Considerations

RFC7519 - JSON Web Token (JWT)

RFC7591 - OAuth 2.0 Dynamic Client Registration Protocol

RFC7592 - OAuth 2.0 Dynamic Client Registration Management Protocol

BCP195 - Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)

OIDC - OpenID Connect Core 1.0 incorporating errata set 1

OIDD - OpenID Connect Discovery 1.0 incorporating errata set 1

MTLS - OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

JARM - Financial-grade API: JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)

PAR - OAuth 2.0 Pushed Authorization Requests

JAR - OAuth 2.0 JWT Secured Authorization Request

---

3. Terms and definitions

For the purpose of this document, the terms defined in **RFC6749**, **RFC6750**, **RFC7636**, **OpenID Connect Core** and **ISO29100** apply.

---

4. Symbols and Abbreviated terms

API – Application Programming Interface

CSRF - Cross Site Request Forgery

FAPI - Financial-grade API

HTTP – Hyper Text Transfer Protocol

OIDF - OpenID Foundation

REST – Representational State Transfer

TLS – Transport Layer Security

---

5. Advanced security profile

5.1. Introduction

The OIDF Financial-grade API (FAPI) security profile specifies security requirements for high risk API resources protected by the OAuth 2.0 Authorization Framework that consists of **RFC6749**, **RFC6750**, **RFC7636**, and other specifications.

There are different levels of risks associated with access to these APIs. For example, read and write access to a bank API has a higher financial risk than read-only access. As such, the security profiles of the authorization framework protecting these APIs are also different.

This profile describes security provisions for the server and client that are appropriate for Financial-grade APIs by defining the measures to mitigate:

- attacks that leverage the weak binding of endpoints in **RFC6749**, and
- attacks that modify authorization requests and responses unprotected in **RFC6749**.

This profile does not support public clients.

The following ways are specified to protect against modifications of authorization responses: Implementations can leverage OpenID Connect's Hybrid Flow that returns an ID Token in the authorization response or they can utilize the JWT Secured Authorization Response Mode for OAuth 2.0 (**JARM**) that returns and protects all authorization response parameters in a JWT.

5.1.1. ID Token as Detached Signature

While the name ID Token (as used in the OpenID Connect Hybrid Flow) suggests that it is something that provides the identity of the resource owner (subject), it is not necessarily so. While it does identify the authorization server by including the issuer identifier, it is perfectly fine to have an ephemeral subject identifier. In this case, the ID Token acts as a detached signature of the issuer to the authorization response and it was an explicit design decision of OpenID Connect Core to make the ID Token act as a detached signature.

This document leverages this fact and protects the authorization response by including the hash of all of the unprotected response parameters, e.g. `code` and `state`, in the ID Token.

While the hash of the code is defined in **OIDC**, the hash of the state is not defined. Thus this document defines it as follows.

s_hash

State hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the state value, where the hash algorithm used is the hash algorithm used in the alg header parameter of the ID Token's JOSE header. For instance, if the alg is HS512, hash the state value with SHA-512, then take the left-most 256 bits and base64url encode them. The s_hash value is a case sensitive string.

5.1.2. JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)

An authorization server may protect authorization responses to clients using the "JWT Secured Authorization Response Mode" **JARM**.

JARM allows a client to request that an authorization server encodes the authorization response (of any response type) in a JWT. It is an alternative to utilizing ID Tokens as detached signatures for providing financial-grade security on authorization responses and can be used with plain OAuth.

This specification facilitates use of **JARM** in conjunction with the response type code.

NOTE: **JARM** can be used to protect OpenID Connect authentication responses. In this case, the OpenID RP would use response type code, response mode jwt and scope openid. This means **JARM** protects the authentication response (instead of the ID Token) and the ID Token containing End-User Claims is obtained from the token endpoint. This facilitates privacy since no End-User Claims are sent through the front channel. It also provides decoupling of message protection and identity providing since a client (or RP) can basically use **JARM** to protect all authorization responses and turn on OpenID if needed (e.g. to log the user in).

5.2. Advanced security provisions

5.2.1. Introduction

API resources may contain sensitive data and/or have increased security requirements. In order to fulfill different security needs, FAPI Security Profile 1.0 defines an advanced profile that is beyond the baseline security requirements defined in the **Part 1: Baseline** document.

As a profile of the OAuth 2.0 Authorization Framework, this document mandates the following for the advanced profile of the FAPI Security Profile 1.0.

5.2.2. Authorization server

The authorization server **SHALL** support the provisions specified in clause 5.2.2 of **Financial-grade API Security Profile 1.0 - Part 1: Baseline**, with the exception that Section 5.2.2-7 (enforcement of **RFC7636**) is not required.

In addition, the authorization server

1. **SHALL** require a JWS signed JWT request object passed by value with the request parameter or by reference with the request_uri parameter;
2. **SHALL** require
    1. the response_type value code id_token, or
    2. the response_type value code in conjunction with the response_mode value jwt;
3. (moved to 5.2.2.1);
4. (moved to 5.2.2.1);
5. **SHALL** only issue sender-constrained access tokens;
6. **SHALL** support **MTLS** as mechanism for constraining the legitimate senders of access tokens;
7. (withdrawn);
8. (moved to 5.2.2.1);
9. (moved to 5.2.2.1);
10. **SHALL** only use the parameters included in the signed request object passed via the request or request_uri parameter;
11. may support the pushed authorization request endpoint as described in **PAR**;

12. (withdrawn);
13. **SHALL** require the request object to contain an exp claim that has a lifetime of no longer than 60 minutes after the nbf claim;
14. **SHALL** authenticate the confidential client using one of the following methods (this overrides **FAPI Security Profile 1.0 - Part 1: Baseline** clause 5.2.2-4):
    1. tls_client_auth or self_signed_tls_client_auth as specified in section 2 of **MTLS**, or
    2. private_key_jwt as specified in section 9 of **OIDC**;
15. **SHALL** require the aud claim in the request object to be, or to be an array containing, the OP's Issuer Identifier URL;
16. **SHALL NOT** support public clients;
17. **SHALL** require the request object to contain an nbf claim that is no longer than 60 minutes in the past; and
18. **SHALL** require **PAR** requests, if supported, to use PKCE (**RFC7636**) with S256 as the code challenge method.

NOTE: MTLS is currently the only mechanism for sender-constrained access tokens that has been widely deployed. Future versions of this specification are likely to allow other mechanisms for sender-constrained access tokens.

NOTE: **PAR** does not present any additional security concerns that necessitated the requirement to use PKCE - the reason PKCE is not required in other cases is merely to be backwards compatible with earlier drafts of this standard.

5.2.2.1. ID Token as detached signature

In addition, if the response_type value code id_token is used, the authorization server

1. **SHALL** support **OIDC**;
2. **SHALL** support signed ID Tokens;
3. **SHOULD** support signed and encrypted ID Tokens;
4. **SHALL** return ID Token as a detached signature to the authorization response;
5. **SHALL** include state hash, s_hash, in the ID Token to protect the state value if the client supplied a value for state. s_hash may be omitted from the ID Token returned from the Token Endpoint when s_hash is present in the ID Token returned from the Authorization Endpoint; and
6. **SHOULD** not return sensitive PII in the ID Token in the authorization response, but if it needs to, then it **SHOULD** encrypt the ID Token.

NOTE: The authorization server may return more claims in the ID Token from the token endpoint than in the one from the authorization response

---

5.2.2.2. JARM

In addition, if the response_type value code is used in conjunction with the response_mode value jwt, the authorization server

1. **SHALL** create JWT-secured authorization responses as specified in **JARM**, Section 4.3.

5.2.3. Confidential client

A confidential client **SHALL** support the provisions specified in clause 5.2.3 and 5.2.4 of **Financial-grade API Security Profile 1.0 - Part 1: Baseline**, except for **RFC7636** support. In addition, the confidential client

1. **SHALL** support **MTLS** as mechanism for sender-constrained access tokens;
2. **SHALL** include the request or request_uri parameter as defined in Section 6 of **OIDC** in the authentication request;
3. **SHALL** ensure the Authorization Server has authenticated the user to an appropriate Level of Assurance for the client's intended purpose;
4. (moved to 5.2.3.1);
5. (withdrawn);
6. (withdrawn);
7. (moved 5.2.3.1);
8. **SHALL** send all parameters inside the authorization request's signed request object;
9. **SHALL** additionally send duplicates of the response_type, client_id, and scope parameters/values using the OAuth 2.0 request syntax as required by Section 6.1 of the OpenID Connect specification if not using **PAR**;
10. **SHALL** send the aud claim in the request object as the OP's Issuer Identifier URL;
11. **SHALL** send an exp claim in the request object that has a lifetime of no longer than 60 minutes;
12. (moved to 5.2.3.1);
13. (moved to 5.2.3.1);
14. **SHALL** send a nbf claim in the request object;
15. **SHALL** use **RFC7636** with S256 as the code challenge method if using **PAR**; and

16. **SHALL** additionally send a duplicate of the client_id parameter/value using the OAuth 2.0 request syntax to the authorization endpoint, as required by Section 5 of **JAR**, if using **PAR**.

5.2.3.1. ID Token as detached signature

In addition, if the response_type value code id_token is used, the client

1. **SHALL** include the value openid into the scope parameter in order to activate **OIDC** support;
2. **SHALL** require JWS signed ID Token be returned from endpoints;
3. **SHALL** verify that the authorization response was not tampered using ID Token as the detached signature;
4. **SHALL** verify that s_hash value is equal to the value calculated from the state value in the authorization response in addition to all the requirements in 3.3.2.12 of **OIDC**; and **NOTE:** This enables the client to verify that the authorization response was not tampered with, using the ID Token as a detached signature.
5. **SHALL** support both signed and signed & encrypted ID Tokens.

5.2.3.2. JARM

In addition, if the response_type value code is used in conjunction with the response_mode value jwt, the client

1. **SHALL** verify the authorization responses as specified in **JARM**, Section 4.4.

5.2.4. (withdrawn)

5.2.5. (withdrawn)

---

6. Accessing protected resources (using tokens)

6.1. Introduction

The FAPI endpoints are OAuth 2.0 protected resource endpoints that return protected information for the resource owner associated with the submitted access token.

6.2. Advanced access provisions

6.2.1. Protected resources provisions

The protected resources supporting this document

1. **SHALL** support the provisions specified in clause 6.2.1 **Financial-grade API Security Profile 1.0 - Part 1: Baseline**; and
2. **SHALL** adhere to the requirements in **MTLS**.

6.2.2. Client provisions

The client supporting this document **SHALL** support the provisions specified in clause 6.2.2 of **Financial-grade API Security Profile 1.0 - Part 1: Baseline**.

7. (Withdrawn)

8. Security considerations

8.1. Introduction

As a profile of the OAuth 2.0 Authorization Framework, this specification references the security considerations defined in Section 10 of **RFC6749**, as well as **RFC6819** - OAuth 2.0 Threat Model and Security Considerations, which details various threats and mitigations. The security of OAuth 2.0 has been proven formally - under certain assumptions – in **OAUTHSEC**. A detailed security analysis of FAPI Security Profile 1.0 can be found in **FAPISEC**.

8.2. Uncertainty of resource server handling of access tokens

There is no way that the client can find out whether the resource access was granted for a bearer or sender-constrained access token. The two differ in the risk profile and the client may want to differentiate them. The protected resources that conform to this document differentiate them. The protected resources that conform to this document **SHALL NOT** accept a bearer access token. They **SHALL** only support sender-constrained access tokens via **MTLS**.

8.3. Attacks using weak binding of authorization server endpoints

---

8.3.1. Introduction

In **RFC6749** and **RFC6750**, the endpoints that the authorization server offers are not tightly bound together. There is no notion of authorization server identifier (issuer identifier) and it is not indicated in the authorization response unless the client uses different redirection URI per authorization server. While it is assumed in the OAuth model, it is not explicitly spelled out and thus many clients use the same redirection URI for different authorization servers exposing an attack surface. Several attacks have been identified and the threats are explained in detail in **RFC6819**.

8.3.2. Client credential and authorization code phishing at token endpoint

In this attack, the client developer is socially engineered into believing that the token endpoint has changed to the URL that is controlled by the attacker. As a result, the client sends the code and the client secret to the attacker, which will be replayed subsequently.

When the FAPI Security Profile 1.0 client uses **MTLS**, the client's secret (the private key corresponding to its TLS certificate) is not exposed to the attacker, which therefore cannot authenticate towards the token endpoint of the authorization server.

8.3.3. Identity provider (IdP) mix-up attack

In this attack, the client has registered multiple IdPs and one of them is a rogue IdP that returns the same client_id that belongs to one of the honest IdPs. When a user clicks on a malicious link or visits a compromised site, an authorization request is sent to the rogue IdP. The rogue IdP then redirects the client to the honest IdP that has the same client_id. If the user is already logged on at the honest IdP, then the authentication may be skipped and a code is generated and returned to the client. Since the client was interacting with the rogue IdP, the code is sent to the rogue IdP's token endpoint. At the point, the attacker has a valid code that can be exchanged for an access token at the honest IdP. See **OAUTHSEC** for a detailed description of the attack.

This attack is mitigated by the use of OpenID Connect Hybrid Flow in which the honest IdP's issuer identifier is included as the value of iss or **JARM** where the iss included in the response JWT. On receiving the authorization response, the client compares the iss value from the response with the issuer URL of the IdP it sent the authorization request to (the rogue IdP). The client detects the conflicting issuer values and aborts the transaction.

8.3.4. (removed)

8.3.5. Access token phishing

Various mechanisms in this specification aim at preventing access token phishing, e.g., the requirement of exactly matching redirect URIs and the restriction on response types that do not return access tokens in the front channel. As a second layer of defense, FAPI Security Profile 1.0 Advanced clients use **MTLS** meaning the access token is bound to the client's TLS certificate. Even if an access token is phished, it cannot be used by the attacker. An attacker could try to trick a client under his control to make use of the access token as described in [FAPISEC] ("Cuckoo's Token Attack" and "Access Token Injection with ID Token Replay"), but these attacks additionally require a rogue AS or misconfigured token endpoint.

8.4. Attacks that modify authorization requests and responses

---

8.4.1. Introduction

In **RFC6749** the authorization request and responses are not integrity protected. Thus, an attacker can modify them.

8.4.2. Authorization request parameter injection attack

In **RFC6749**, the authorization request is sent as a query parameter.

Although **RFC6749** mandates the use of TLS, the TLS is terminated in the browser and thus not protected within the browser; as a result an attacker can tamper the authorization request and insert any parameter values.

The use of a request object or request_uri in the authorization request will prevent tampering with the request parameters.

The IdP confusion attack reported in **SoK: Single Sign-On Security – An Evaluation of OpenID Connect** is an example of this kind of attack.

8.4.3. Authorization response parameter injection attack

This attack occurs when the victim and attacker use the same relying party client. The attacker is somehow able to capture the authorization `code` and `state` from the victim's authorization response and uses them in his own authorization response.

This can be mitigated by using OpenID Connect Hybrid Flow where the c_hash, at_hash, and s_hash can be used to verify the validity of the authorization code, access token, and state parameters. It can also be mitigated using **JARM** by verifying the integrity of the authorization response JWT.

The server can verify that the state is the same as what was stored in the browser session at the time of the authorization request.

8.5. TLS considerations

As confidential information is being exchanged, all interactions **SHALL** be encrypted with TLS (HTTPS).

Section 7.1 of **Financial-grade API Security Profile 1.0 - Part 1: Baseline** **SHALL** apply, with the following additional requirements:

1. For TLS versions below 1.3, only the following 4 cipher suites **SHALL** be permitted:
    - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
2. For the authorization_endpoint, the authorization server **MAY** allow additional cipher suites that are permitted by the latest version of **BCP195**, if necessary to allow sufficient interoperability with users' web browsers or are required by local regulations.

   NOTE: Permitted cipher suites are those that **BCP195** does not explicity say **MUST NOT** use.
3. When using the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 or TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 cipher suites, key lengths of at least 2048 bits are required.

8.6. Algorithm considerations

For JWS, both clients and authorization servers

1. **SHALL** use PS256 or ES256 algorithms;

2. **SHOULD** not use algorithms that use RSASSA-PKCS1-v1_5 (e.g. RS256); and
3. **SHALL NOT** use none.

8.6.1. Encryption algorithm considerations

For JWE, both clients and authorization servers

1. **SHALL NOT** use the RSA1_5 algorithm.

8.7. Incomplete or incorrect implementations of the specifications

To achieve the full security benefits, it is important the implementation of this specification, and the underlying OpenID Connect and OAuth specifications, are both complete and correct.

The OpenID Foundation provides tools that can be used to confirm that an implementation is correct:

https://openid.net/certification/

The OpenID Foundation maintains a list of certified implementations:

https://openid.net/developers/certified/

Deployments that use this specification **SHOULD** use a certified implementation.

8.8. Session Fixation

An attacker could prepare an authorization request URL and trick a victim into authorizing access to the requested resources, e.g. by sending the URL via e-Mail or utilizing it on a fake site.

OAuth 2.0 prevents this kind of attack since the process for obtaining the access token (code exchange, CSRF protection etc.) is designed in a way that the attacker will be unable to obtain and use the token as long as it does not control the victim's browser.

However, if the API allows execution of any privileged action in the course of the authorization process before the access token is issued, these controls are rendered ineffective. Implementers of this specification therefore **SHALL** ensure any action is executed using the access token issued by the authorization process.

For example, payments **SHALL NOT** be executed in the authorization process but after the Client has exchanged the authorization code for a token and sent an "execute payment" request with the access token to a protected endpoint.

8.9. JWKS URIs

This profile requires both Clients and Authorization Servers to verify payloads with keys from the other party. The AS verifies request objects and private_key_jwt assertions. The Client verifies ID Tokens and authorization response JWTs. For AS's this profile strongly recommends the use of JWKS URI endpoints to distribute public keys. For Clients this profile recommends either the use of JWKS URI endpoints or the use of the jwks parameter in combination with **RFC7591** and **RFC7592**.

The definition of the AS jwks_uri can be found in **RFC8414**, while the definition of the Client jwks_uri can be found in **RFC7591**.

In addition, this profile

1. requires that jwks_uri endpoints **SHALL** be served over TLS;
2. recommends that JOSE headers for x5u and jku **SHOULD** not be used; and
3. recommends that the JWK set does not contain multiple keys with the same kid.

8.10. Multiple clients sharing the same key

The use of **MTLS** for client authentication and sender constraining access tokens brings significant security benefits over the use of shared secrets. However in some deployments the certificates used for **MTLS** are issued by a Certificate Authority at an organization level rather than a client level. In such situations it may be common for an organization with multiple clients to use the same certificates (or certificates with the same DN) across clients. Implementers **SHOULD** be aware that such sharing means that a compromise of any one client, would result in a compromise of all clients sharing the same key.

8.11. Duplicate Key Identifiers

JWK sets **SHOULD** not contain multiple keys with the same kid. However, to increase interoperability when there are multiple keys with the same kid, the verifier **SHALL** consider other JWK attributes, such as kty, use, alg, etc., when selecting the verification key for the particular JWS message. For example, the following algorithm could be used in selecting which key to use to verify a message signature:

1. find keys with a kid that matches the kid in the JOSE header;
2. if a single key is found, use that key;
3. if multiple keys are found, then the verifier **SHOULD** iterate through the keys until a key is found that has a matching alg, use, kty, or crv that corresponds to the message being verified.

---

9. Privacy considerations

9.1. Introduction

There are many factors to be considered in terms of privacy when implementing this document. However, since this document is a profile of OAuth and OpenID Connect, all of them are generic and applies to OAuth or OpenID Connect and not specific to this document. Implementers are advised to perform a thorough privacy impact assessment and manage identified risks appropriately.

NOTE: Implementers can consult documents like **ISO29100** and [ISO29134] for this purpose.

Privacy threats to OAuth and OpenID Connect implementations include the following:

- (Inappropriate privacy notice) A privacy notice provided at a policy_url or by other means can be inappropriate.
- (Inadequate choice) Providing a consent screen without adequate choices does not form consent.
- (Misuse of data) An AS, RS or Client can potentially use the data not according to the purpose that was agreed.
- (Collection minimization violation) Clients asking for more data than it absolutely needs to fulfil the purpose is violating the collection minimization principle.
- (Unsolicited personal data from the Resource) Some bad resource server implementations may return more data than was requested. If the data is personal data, then this would be a violation of privacy principles.
- (Data minimization violation) Any process that is processing more data than it needs is violating the data minimization principle.
- (RP tracking by AS/OP) AS/OP identifying what data is being provided to which Client/RP.
- (User tracking by RPs) Two or more RPs correlating access tokens or ID Tokens to track users.
- (RP misidentification by User at AS) User misunderstands who the RP is due to a confusing representation of the RP at the AS's authorization page.
- (Mismatch between User's understanding or what RP is displaying to a user and the actual authorization request). To enhance the trust of the ecosystem, best practice is for the AS to make clear what is included in the authorisation request (for example, what data will be released to the RP).
- (Attacker observing personal data in authorization request) Authorization request might contain personal data. This can be observed by an attacker.

- (Attacker observing personal data in authorization endpoint response) In some frameworks, even state is deemed personal data. This can be observed by an attacker through various means.
- (Data leak from AS) AS stores personal data. If AS is compromised, these data can leak or be modified.
- (Data leak from Resource) Some resource servers (RS) store personal data. If a RS is compromised, these data can leak or be modified.
- (Data leak from Clients) Some clients store personal data. If the client is compromised, these data can leak or be modified.

These can be mitigated by choosing appropriate options in OAuth or OpenID, or by introducing some operational rules. For example, "Attacker observing personal data in authorization request" can be mitigated by either using authorization request by reference using request_uri or by encrypting the request object. Similarly, "Attacker observing personal data in authorization endpoint response" can be mitigated by encrypting the ID Token or JARM response.

---

10. Acknowledgement

The following people contributed to this document:

- Nat Sakimura (NAT Consulting) -- Chair, Editor
- Anoop Saxena (Intuit) -- Co-chair, FS-ISAC Liaison
- Anthony Nadalin (Microsoft) -- Co-chair, SC 27 Liaison
- Edmund Jay (Illumila) -- Co-editor
- Dave Tonge (Moneyhub) -- Co-chair, UK Implementation Entity Liaison
- Paul A. Grassi (NIST) -- X9 Liaison
- Joseph Heenan (Authlete)
- Sascha H. Preibisch (CA)
- Henrik Biering (Peercraft)
- Anton Taborszky (Deutsche Telecom)
- John Bradley (Yubico)
- Tom Jones (Independent)
- Axel Nennker (Deutsche Telekom)
- Daniel Fett (yes.com)
- Torsten Lodderstedt (yes.com)

- Ralph Bragg (Raidiam)
- Brian Campbell (Ping Identity)
- Dima Postnikov (Independent)
- Stuart Low (Biza.io)
- Takahiko Kawasaki (Authlete)
- Vladimir Dzhuvinov (Connect2Id)
- Chris Michael (Open Banking)
- Freddi Gyara (Open Banking)
- Rob Otto (Ping Identity)
- Francis Pouatcha (adorsys)
- Kosuke Koiwai (KDDI)
- Bjorn Hjelm (Verizon)
- Lukasz Jaromin (Cloudentity)
- James Manger

---

11. Bibliography

- **Part1** Financial-grade API Security Profile 1.0 - Part 1: Baseline
- **ISODIR2** ISO/IEC Directives Part 2
- **ISO29100** ISO/IEC 29100 Information technology — Security techniques — Privacy framework
- [ISO29134] ISO/IEC 29134 Information technology — Security techniques — Guidelines for privacy impact assessment
- [ISO29184] ISO/IEC 29184 Information technology — Online privacy notices and consent
- **RFC6749** The OAuth 2.0 Authorization Framework
- **RFC6750** The OAuth 2.0 Authorization Framework: Bearer Token Usage
- **RFC7636** Proof Key for Code Exchange by OAuth Public Clients
- **RFC6819** OAuth 2.0 Threat Model and Security Considerations
- **RFC7519** JSON Web Token (JWT)
- **RFC7591** OAuth 2.0 Dynamic Client Registration Protocol
- **RFC7592** OAuth 2.0 Dynamic Client Registration Management Protocol

- **RFC8414** OAuth 2.0 Authorization Server Metadata
- **OIDC** OpenID Connect Core 1.0 incorporating errata set 1
- **OIDD** OpenID Connect Discovery 1.0 incorporating errata set 1
- **BCP195** Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
- **MTLS** OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens
- **JARM** Financial-grade API: JWT Secured Authorization Response Mode for OAuth 2.0
- **PAR** OAuth 2.0 Pushed Authorization Requests
- **JAR** OAuth 2.0 JWT Secured Authorization Request
- **SoK** Mainka, C., Mladenov, V., Schwenk, J., and T. Wich: SoK: Single Sign-On Security – An Evaluation of OpenID Connect
- **FAPISEC** Fett, D., Hosseyni, P., Kuesters, R.: An Extensive Formal Security Analysis of the OpenID Financial-grade API
- **OAUTHSEC** Fett, D., Kuesters, R., Schmitz, G.: A Comprehensive Formal Security Analysis of OAuth 2.0

---

12. IANA Considerations

12.1. Additions to JWT Claims Registry

This specification adds the following values to the "JSON Web Token Claims" registry established by **RFC7519**.

12.1.1. Registry Contents

- Claim name: s_hash
- Claim Description: State hash value
- Change Controller: OpenID Foundation Financial-Grade API Working Group - openid-specs-fapi@lists.openid.net
- Reference: Section 5 of [[this specification]]

Appendix A. Examples

The following are non-normative examples of various objects compliant with this specification, with line wraps within values for display purposes only.

The examples signed by the client may be verified with the following JWK:


```
{

  "kty": "RSA",

  "e": "AQAB",

  "use": "sig",

  "kid": "client-2020-08-28",

  "alg": "PS256",

  "n": "i0Ybm4TJyErnD5FIs-
6sgAdtP6fG631FXbe5gcOGYgn9aC2BS2h9Ah5cRGQpr3aLLVKCRWU6

HRfnGseUBOejo57vI-kgab2YsQJSwedAxvtKrIrJlgKn1gTXMNsz-
NQd1LyLSV50qJVEy5l9RtsdDzOV

8_kLCbzroEL3rc00iqVZBcQiYm8Bx4z0G8LYZ4oMJAG462Mf_znJkKXsuSIH735x
nSmx74CC8TOe6G-V

0Wi_wVSJ9bHPphSki_kWUtjVGcnyjYuQVE0LRj3qrGPAX9bsVKSqs8T9AM41TB9o
V5Sjz5YhggwICvvC

CGwil9qhUoQRkeXtWuGCfvCSeTdawQ"

}
```


The examples signed by the server may be verified with the following JWK:


```
{

  "kty": "RSA",

  "e": "AQAB",

  "use": "sig",

  "kid": "server-2020-08-28",

  "alg": "PS256",

  "n": "pz6g0h7Cu63SHE8_Ib4l3hft8XuptZ-
Or7v_j1EkCboyAEn_ZCuBrQOmpUIoPKrA0JNWK_fF
```

```
eZ2q1_26Gvn3E4dQlcOWpiWkKmxAhYCWnNDv3urVgldDp_kw0Dx2H8yn9tmFW28E
_WvrZRwHEF5Czigb

xlmFIrkniMHRzjyYQTHRU0gW3DRV9MrQQrmP71McvfLPeMBPPgsHgLo7KmUBDoUj
sgnwgycEOWPm8MWJ

13dpTsVnoWNIFQqVNz1L5pRU3Uoknl0MGoE6v0M9lfgQgzxIX9gSB1VGp5zZRcsn
ZGU3MFpwBhOWwiCU

wqztoX0H5P0g7OWocspHrDn6YOgxHw"

}
```

A.1. Example request object

```
eyJraWQiOiJjbGllbnQtMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJhdWQi
OiJodHRwczpcL1wv

ZmFwaS1hcy5leGFtcGxlLmNvbVwvIiwibmJmIjoxNTk0MTQwMDMwLCJzY29wZSI6
Im9wZW5pZCBwYXlt

ZW50cyIsImlzcyI6IjUyNDgwNzU0MDUzIiwicmVzcG9uc2VfdHlwZSI6ImNvZGUg
aWRfdG9rZW4iLCJy

ZWRpcmVjdF91cmkiOiJodHRwczpcL1wvZmFwaS1jbGllbnQuZXhhbXBsZS5vcmdc
L2ZhcGktYXMtY2Fs

bGJhY2siLCJzdGF0ZSI6IlZnU1VJRW5mbG5EeFRlMXZBdHI1NG8iLCJleHAiOjE1
OTQxNDAzOTAsIm5v

bmNlIjoiN3hEQ0h2aXVQTVNYSklpZ2tIT2NEaSIsImNsaWVudF9pZCI6IjUyNDgw
NzU0MDUzIn0.VSo5

VWN3lOiCry2KItU5RI62i9KG2KQlBdpsDT0DI0vSMK-
q85aJZvsMiHBNBv1PQ9qAWmU3oJS-yi-Ks_lD

lP6lIMFrOL_Ym3VxJ_SM6lrc8JSZH_nNx6sqxPpeMQTF4SFPx30vHrlBVJaCGfnC
MVC6Nbzwef0vOEpN

ixZT-9cwa3dZ-
pddAyt58dKGxS76NR_wxdBaSKN0AfPoui0HSSaAkIdRds21NKIOf4r9BjV5lr1Oi
-4I

JUQp-xdeLCPD3fD6Y-TJbHFToJ4FsQzglN83BfNYaeXV_yTtK7yeSw2R-
ee0b3uMV0iD1ee77b7bbcjR

3msLISFjM40d9Pv8qQ
```

which when decoded has the following body:

```
{

  "aud": "https://fapi-as.example.com/",

  "nbf": 1594140030,

  "scope": "openid payments",

  "iss": "52480754053",

  "response_type": "code id_token",

  "redirect_uri": "https://fapi-client.example.org/fapi-as-
callback",

  "state": "VgSUIEnflnDxTe1vAtr54o",

  "exp": 1594140390,

  "nonce": "7xDCHviuPMSXJIigkHOcDi",

  "client_id": "52480754053"

}
```

A.2. Example signed id_token for authorization endpoint response


```
eyJraWQiOiJzZXJ2ZXItMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJzdWIi
OiIxMDAxIiwiYXVk

IjoiNTI0ODA3NTQwNTMiLCJjX2hhc2giOiJRUjJ6dWNmWVpraUxyYktCS0RWcGdR
Iiwic19oYXNoIjoi

OXM2Q0JiT3hpS0U2NWQ5LVFyMFFJUSIsImF1dGhfdGltZSI6MTU5NDE0MDA5MCwi
aXNzIjoiaHR0cHM6

XC9cL2ZhcGktYXMuZXhhbXBsZS5jb21cLyIsImV4cCI6MTU5NDE0MDM5MCwiaWF0
IjoxNTk0MTQwMDkw

LCJub25jZSI6Ijd4RENIdml1UE1TWEpJaWdrSE9jRGkifQ.Z-
LpQRuYoiTqEBfVfctn-e6bLwSMqi8wA

3TuARGW6GyD05gPF6TVlUwHgJnSUlhETrzhEUAKKiyGDxGspuBU0OAnB4qepgrEB
izk980NjCEVXNkog

v0ANv9VX_01Lcl0d_6_c-
AUjwDSuKY8rDfvggKSJFzRilbQuB8b1drAIAZpc6kMObY3PcQZ_vKTMsQ8l
```

```
HCuXXRuAo__0xRE6l_iiRCos_940GrJr0Sih9uTQpnCWBoEab1dC0l-
vUp4lP0TQRKNpDoPoMOj10KJA

8T8pKhjZ8TKM-wo9A4qH2LBgUIYJyjd8bWfKTZxCNmLRzRr-
_JBG7fF_fpOUhGT_DhzMw
```

which when decoded has the following body:

```
{

  "sub": "1001",

  "aud": "52480754053",

  "c_hash": "QR2zucfYZkiLrbKBKDVpgQ",

  "s_hash": "9s6CBbOxiKE65d9-Qr0QIQ",

  "auth_time": 1594140090,

  "iss": "https://fapi-as.example.com/",

  "exp": 1594140390,

  "iat": 1594140090,

  "nonce": "7xDCHviuPMSXJIigkHOcDi"

}
```

A.3. Example signed and encrypted id_token for authorization endpoint response

```
eyJraWQiOiJjbGllbnQtZW5jLTIwMjAtMDgtMjgiLCJjdHkiOiJKV1QiLCJlbmMi
OiJBMjU2R0NNIiwi

YWxnIjoiUlNBLU9BRVAtMjU2In0.LFvxFCzJ-
1NRl48pXTUs8f2axm5MRe9Cv0dgV6sXTRKwkT3nC2SJ

QlutOol36VARLd3uaIoj4Z7LVV_MrdIYYvDci2WLlKSlI_NRgR3qJ25N3S6fCqNE
YRgDDbNzSr15MDRc

WQR5Jdl3VP8g748cowD_2gaopaCzZWTa3r_J2VOEETfcBAIMX0NbtVA3hHW-
rQ0aCC7UIbP0_oEB2YF0

u6qAXCXuC02nO6coMSpSHTDZwkqkmFiFEKERM_Gayz3lVddlgfcPR2k76bCUjWy9
34-rOrOBGcLyS1Ww

aTIqMUS3WEIsAwCDr1Jt4pAioryRLZfLmWNff4QZSBxWejRqpw.uRANzseIWYB9Y
eAW.sJGqF2ERkMEE
```

```
jm8h62tUA4UeZIBqvVRpkQqjTuae7-4ac-
4sSth0A3zeERvlyC5GcP0W2tj7uxMi0I4gpN33OfAOR-tA

9E_47oCHXrOH-
7cpLgVIxxWZFx43dhxUh5QHuBfli4nHErMVUsFq6CzQj8Z5SHvBD2Qx3suPEeCNo
_M2

woohCprwjOKhE-Q_VkWUJb-
Elrq9HxJcBtadw0spolqgYYTIWvV4fcKmbtGANYLac29oKWd5-jyDAsSF

FZrSCNxv-BtJUiUVWUn5eVufjJYCx62Ju-MZ8vsPNTE-
_I5em9RTBja6ylcivjzhW9Ncl6yKVfnB0XJN

cSSHQSFhc6Gvy7oYMBXx1C5G31OsiklkKQX2gsAZlxFQ_X25AXpMoV8-
5xsUwdMdTaPxIIsccbrK2dfA

aP0rUruSV8zrlrbsN3ftjTJSka2XGG3kra76EPAlzSwxy6XdFVtEV31hirV3f9g0
4Gj_e-Q7J7HR62eY

3_09WyARShQL3DVXWOcK_8YrLr58JjNAbm0s5dAUq-
zt9cMv8rl05t_dE59Gi6Hnl2YAiRdYG6B71FxJ

CE2Uqciy2jLe6mCDFDfqkog4G5R9FzNz5VzhVpmZVm3OJkug-
UzayN7nwZ7jsmxQ2ucCM03xq-0MLdsk

H-cleahkFw5S-
W40cn5hLrRXSqUoYfKmVSd9RltOZ6T0VrYpw2LaF2uUYEO9w9bMmg2zzfxft4WHs
EbD

OlJVb5SE8mUjzBBZAcgaHYSv0Wii70lEJvLSdnVI1r9kuu9ae_j1Tu08RVyFGfgi
xYjI9z2L_sc8uOoO

HJ-
Tq1iuncL3lCQJBuwBFoxyINlFgz4YV2AgreNsX8bDfE9XbRB9TnfvSd6rmes9lO0
-3VQFlsC0C5dx

VXgp5o05E8nisPwuLmlGO5BTtBzCQ3tIH2SuTLTG-
gohTEUVn4fACwIiyuXdPXcF4GxJNRNgNOH7xwxx

55qEM0xl2GuSseV59FiZR-WKMMs.kScy0JLB4XECklDAwTIVNA
```

which when decrypted using the following key:

```
{

    "kty": "RSA",

    "d": "OjDe8EkZXgvB-
Gy5A4EdU8fBuAjdHLMyHKAtMaS_W_joEJHDvZRhIYbh1jAyHYoR3kFMXu

tCIYpRjDrsUEhjYuVKLm90CVtysoRjjkiXyupcEW3o--X_HBJhKm1Y-0I7LQ-
cA7CotJpTVMR2fRTqP1
```

```
T4FsORAjg9l-
fbdpVmeDiZBRbL2zCWmKWhtDpHyy7vbSCRghntihz_M5Hrchk7r8ito_K3dFrV9I
ZSF9

RoEY7kyK5bL36Kpgai44PYCzqOzqP2fteO_rZ9fn-
uK59pI3ySo_PgSbJ55n14Nd9Z8m70zE9Z4aIeND

EFspZUhavngRwc7MuJ7f_hVGQ9RFbbkQ",

    "e": "AQAB",

    "use": "enc",

    "kid": "client-enc-2020-08-28",

    "n":
"jVc92j0ntTV0V1nwZ3mpGaV2bME4d6AMS2SRrJBM0fLehaTEqDNzGu0warz2SC9
bhcBOB5

_q3mYBFjmTwWzSbsk6RYETnAgViXg67PgH7Vkx2NCtwgQW3cNdnUZWRNYHsoevkx
_Ta1X6Vi9ulebU_B

CKjrF-
6CjVcGgEsO_S5DKcukGHdf81WlQOq3zGQg4h7MLArrbPSTHHORDsu_87qY9m2Ehi
YSOBSF5rHs

fDo7zWI5FWNG-_HO-CBM005bykIIS1aXCXx1jOW1OrKcp5xv3e-
BR6MJTxncZJ4o1GtynJI8kLXRgltL

ArSOkbzNEr9GjU9lnSSxKLMtRLKkG2Ow"

}
```

has the following body:

```
{

  "sub": "1001",

  "aud": "2334382354153498",

  "acr": "urn:cds.au:cdr:2",

  "c_hash": "BLfy9hvQUZTDq6_KmF4kDQ",

  "s_hash": "9s6CBbOxiKE65d9-Qr0QIQ",

  "auth_time": 1595827190,

  "iss": "https://fapi-as.example.com/",

  "exp": 1595827490,
```

```
  "iat": 1595827190,

  "nonce": "7xDCHviuPMSXJIigkHOcDi"

}
```

A.4. Example JARM response

```
eyJraWQiOiJzZXJ2ZXItMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJhdWQi
OiI0NjkxODA2NDgw

MzkwNTEiLCJjb2RlIjoiendrR2FjOWp1TFg4RjhmcmFwRElTaTNLMkZ3bG40cXh3
eWZOSUkzQ2p6MCIs

ImlzcyI6Imh0dHBzOlwvXC9mYXBpLWFzLmV4YW1wbGUuY29tXC8iLCJzdGF0ZSI6
IlZnU1VJRW5mbG5E

eFRlMXZBdHI1NG8iLCJleHAiOjE1OTQxNDEwOTB9.k_3df0dIDX6watKxQkzAHOL
gf4FBi_xIPN-n8aT

5hMX3gaBbeDqdUA5NR764L4ugdDgXyQm8dNcZrZldKIPfSfRcjBTtSx9PEdiffn_
xUkwnS18YNAfEoq0

HjvkOQ59F21ImKn113kon00uC2dqBGByRrZcaUYOnvW2DdHCVA0VTW2je5nzbI02
z9csLa8uGGGwjWRP

Ec9j9bvR1Adc2m2Z-o0QCRIBl81sZz6_AnE-wPTw-KZFQBs3FgS-
r0FDYOzE7FHIMgDBSKAg1J5tWY3J

wRuIv_oAbYdSlxdYzrbFQ9grX4MA0p7pk5lS-kwnN845GZ2k1_yaOLtYYyvRFrw
```

which when decoded has the following body:

```
{

  "aud": "469180648039051",

  "code": "zwkGac9juLX8F8frapDISi3K2Fwln4qxwyfNII3Cjz0",

  "iss": "https://fapi-as.example.com/",

  "state": "VgSUIEnflnDxTe1vAtr54o",

  "exp": 1594141090

}
```

A.5. Example private_key_jwt client assertion

```
eyJraWQiOiJjbGllbnQtMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJzdWIi
OiI1MjQ4MDc1NDA1

MyIsImF1ZCI6Imh0dHBzOlwvXC9mYXBpLWFzLmV4YW1wbGUuY29tXC9hcGlcL3Rv
a2VuIiwiaXNzIjoi

NTI0ODA3NTQwNTMiLCJleHAiOjE1OTQxNDAxNTEsImlhdCI6MTU5NDE0MDA5MSwi
anRpIjoiNHZCY3RN

U2tLNHdmdU91aTlDeWMifQ.h3i0k2DWc7V6WEiinHAsse-
pOFiWxe5kD4KetdGX65Q03orj0Fh6EWfdE

AntCrOodUsypKjM1ia3evbQmsSkhIb4YK5s53hYYtEbJC_eG9jFnVc4ki7Qc5O-
1K-D80w7WT1UI--Ih

Ku-i22Ai_nMed-71UWLHcPi7W20SCroPHXfaLiFj_TOsr7I8h7VNsoa7P3-
coHlXT5q4cMjIA7t8cRag

sGtKlIgwdFYySlimtSESDM0U-
_NUPperTgnF8FVn7SqtizBJneZNAWwSLJD9AVsnMOH6kOeNLtpopsru

Dcs54S_aIlroP-BdiHw9R1qRTIVSoX3k_EStvoWSf8NcQ
```

which when decoded has the following body:

```
{

  "sub": "52480754053",

  "aud": "https://fapi-as.example.com/api/token",

  "iss": "52480754053",

  "exp": 1594140151,

  "iat": 1594140091,

  "jti": "4vBctMSkK4wfuOui9Cyc"

}
```

Appendix B. Copyright notice & license

Authors' Addresses

Nat Sakimura

Nat Consulting

Email: nat@nat.consulting

URI: http://nat.sakimura.org/

John Bradley

Yubico

Email: ve7jtb@ve7jtb.com

URI: http://www.thread-safe.com/

Illumila

Illumila

Email: ejay@mgi1.com

URI: http://illumi.la/

Exhibit B : Financial-grade API Security Profile 1.0 - Part 1: Baseline

Note: Content in this section may also be found at Financial-grade API Security Profile 1.0 - Part 1: Baseline, and is provided here in this exhibit for convenience.

Foreword

The OpenID Foundation (OIDF) promotes, protects and nurtures the OpenID community and technologies. As a non-profit international standardizing body, it is comprised by over 160 participating entities (workgroup participants). The work of preparing implementer drafts and final international standards is carried out through OIDF workgroups in accordance with the OpenID Process. Participants interested in a subject for which a workgroup has been established have the right to be represented in that workgroup. International organizations, governmental and non-governmental, in liaison with OIDF, also take part in the work. OIDF collaborates closely with other standardizing bodies in the related fields.

Final drafts adopted by the Workgroup through consensus are circulated publicly for the public review for 60 days and for the OIDF members for voting. Publication as an OIDF Standard requires approval by at least 50 % of the members casting a vote. There is a possibility that some of the elements of this document may be the subject to patent rights. OIDF **SHALL NOT** be held responsible for identifying any or all such patent rights.

Financial-grade API Security Profile 1.0 consists of the following parts:

- Financial-grade API Security Profile 1.0 - Part 1: Baseline
- Financial-grade API Security Profile 1.0 - Part 2: Advanced

These parts are intended to be used with RFC6749, RFC6750, RFC7636, and OIDC.

Introduction

The Financial-grade API is a highly secured OAuth profile that aims to provide specific implementation guidelines for security and interoperability. The Financial-grade API security profile can be applied to APIs in any market area that requires a higher level of security than provided by standard OAuth or OpenID Connect. Among other security enhancements, this specification provides a secure alternative to screen scraping. Screen scraping accesses user's data and functions by impresonating a user through password sharing. This brittle,

inefficient, and insecure practice creates security vulnerabilities which require financial institutions to allow what appears to be an automated attack against their applications.

This document is Part 1 of FAPI Security Profile 1.0. It specifies a baseline security profile of OAuth that is suitable for protecting APIs with a moderate inherent risk. Importantly, this profile does not provide non-repudiation (signing of authorization requests and responses) and sender-constrained access tokens. If such features or a higher level of security is desired, the use of **Financial-grade API Security Profile 1.0 - Part 2: Advanced** is recommended.

Although it is possible to code an OpenID Provider and Relying Party from first principles using this specification, the main audience for this specification is parties who already have a certified implementation of OpenID Connect and want to achieve a higher level of security. Implementers are encouraged to understand the security considerations contained in Section 7.6 before embarking on a 'from scratch' implementation.

Notational Conventions

The keywords "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**MAY**", and "**CAN**" in this document are to be interpreted as described in **ISO Directive Part 2**. These keywords are not used as dictionary terms such that any occurrence of them **SHALL** be interpreted as keywords and are not to be interpreted with their natural language meanings.

Table of Contents

---

1. Scope

This document specifies the method for an application to:

- obtain OAuth tokens in a moderately secure manner for access to protected data;
- use OpenID Connect (OIDC) to identify the customer (user); and
- use tokens to access REST APIs in a moderately secure manner.

---

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applied. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISODIR2 - ISO/IEC Directives Part 2

RFC4122 - A Universally Unique IDentifier (UUID) URN Namespace

RFC6749 - The OAuth 2.0 Authorization Framework

RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage

RFC7636 - Proof Key for Code Exchange by OAuth Public Clients

RFC6125 - Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)

BCP212 - OAuth 2.0 for Native Apps

RFC6819 - OAuth 2.0 Threat Model and Security Considerations

BCP195 - Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)

OIDC - OpenID Connect Core 1.0 incorporating errata set 1

X.1254 - Entity authentication assurance framework

MTLS - OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

RFC8414 - OAuth 2.0 Authorization Server Metadata

OIDD - OpenID Connect Discovery 1.0 incorporating errata set 1

RFC7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content

---

3. Terms and definitions

For the purpose of this document, the terms defined in **RFC6749**, **RFC6750**, **RFC7636**, **OpenID Connect Core** apply.

---

4. Symbols and abbreviated terms

API – Application Programming Interface

CSRF - Cross Site Request Forgery

FAPI - Financial-grade API

HTTP – Hyper Text Transfer Protocol

REST – Representational State Transfer

TLS – Transport Layer Security

---

5. Baseline security profile

5.1. Introduction

The OIDF Financial-grade API (FAPI) security profile specifies security requirements for API resources protected by the OAuth 2.0 Authorization Framework that consists of **RFC6749**, **RFC6750**, **RFC7636**, and other specifications.

FAPI Security Profile 1.0 - Part 1: Baseline and **Part 2: Advanced** specify different levels of security. The characteristics required of the tokens are different and the methods to obtain tokens are explained separately. This document specifies the baseline security provisions.

5.2. Baseline security provisions

5.2.1. Introduction

Some APIs, such as ones that provide potentially sensitive information, require a greater level of protection than basic **RFC6749** requires. FAPI provides such greater protection.

As a profile of the OAuth 2.0 Authorization Framework, this document mandates the following to the baseline profile of the FAPI Security Profile 1.0.

5.2.2. Authorization server

The authorization server

1. **SHALL** support confidential clients;
2. **SHOULD** support public clients;
3. **SHALL** provide a client secret that adheres to the requirements in Section 16.19 of **OIDC** if a symmetric key is used;
4. **SHALL** authenticate the confidential client using one of the following methods:
    1. Mutual TLS for OAuth Client Authentication as specified in Section 2 of **MTLS**, or
    2. `client_secret_jwt` or `private_key_jwt` as specified in Section 9 of **OIDC**;
5. **SHALL** require and use a key of size 2048 bits or larger for RSA algorithms;
6. **SHALL** require and use a key of size 160 bits or larger for elliptic curve algorithms;
7. **SHALL** require **RFC7636** with S256 as the code challenge method;
8. **SHALL** require redirect URIs to be pre-registered;
9. **SHALL** require the redirect_uri in the authorization request;
10. **SHALL** require the value of redirect_uri to exactly match one of the pre-registered redirect URIs;
11. **SHALL** require user authentication to an appropriate Level of Assurance for the operations the client will be authorized to perform on behalf of the user;
12. **SHALL** require explicit approval by the user to authorize the requested scope if it has not been previously authorized;
13. **SHALL** reject an authorization code (Section 1.3.1 of **RFC6749**) if it has been previously used;
14. **SHALL** return token responses that conform to Section 4.1.4 of **RFC6749**;

15. **SHALL** return the list of granted scopes with the issued access token if the request was passed in the front channel and was not integrity protected;
16. **SHALL** provide non-guessable access tokens, authorization codes, and refresh token (where applicable), with sufficient entropy such that the probability of an attacker guessing the generated token is computationally infeasible as per **RFC6749** Section 10.10;
17. **SHOULD** clearly identify the details of the grant to the user during authorization as in 16.18 of **OIDC**;
18. **SHOULD** provide a mechanism for the end-user to revoke access tokens and refresh tokens granted to a client as in 16.18 of **OIDC**;
19. **SHALL** return an invalid_client error as defined in 5.2 of **RFC6749** when mis-matched client identifiers were provided through the client authentication methods that permits sending the client identifier in more than one way;
20. **SHALL** require redirect URIs to use the https scheme;
21. **SHOULD** issue access tokens with a lifetime of under 10 minutes unless the tokens are sender-constrained; and
22. **SHALL** support **OIDD**, may support **RFC8414** and **SHALL NOT** distribute discovery metadata (such as the authorization endpoint) by any other means.
    NOTE: The use of refresh tokens instead of long-lived access tokens for both public and confidential clients is recommended.
    NOTE: The Financial-grade API Security Profile 1.0 server may limit the scopes for the purpose of not implementing certain APIs.
    NOTE: Clients are expected to treat access tokens as opaque strings and replay them as is. Authorization servers can issue unstructured or structured access tokens (for example, a signed JWT).
    NOTE: The requirement to return the list of granted scopes allows clients to detect when the authorization request was modified to include different scopes. Servers **MUST** still return the granted scopes if they are different from those requested.

5.2.2.1. Returning authenticated user's identifier

Further, if it is desired to provide the authenticated user's identifier to the client in the token response, the authorization server:

1. **SHALL** support the authentication request as in Section 3.1.2.1 of **OIDC**;
2. **SHALL** perform the authentication request verification as in Section 3.1.2.2 of **OIDC**;
3. **SHALL** authenticate the user as in Section 3.1.2.2 and 3.1.2.3 of **OIDC**;
4. **SHALL** provide the authentication response as in Section 3.1.2.4 and 3.1.2.5 of **OIDC** depending on the outcome of the authentication;

5. **SHALL** perform the token request verification as in Section 3.1.3.2 of **OIDC**; and
6. **SHALL** issue an ID Token in the token response when openid was included in the requested scope as in Section 3.1.3.3 of **OIDC** with its sub value corresponding to the authenticated user and optional acr value in ID Token.

5.2.2.2. Client requesting openid scope

If the client requests the openid scope, the authorization server

1. **SHALL** require the nonce parameter defined in Section 3.1.2.1 of **OIDC** in the authentication request.

5.2.2.3. Clients not requesting openid scope

If the client does not requests the openid scope, the authorization server

1. **SHALL** require the state parameter defined in Section 4.1.1 of **RFC6749**.

5.2.3. Public client

A public client

1. **SHALL** support **RFC7636**;
2. **SHALL** use S256 as the code challenge method for the **RFC7636**;
3. **SHALL** use separate and distinct redirect URI for each authorization server that it talks to;
4. **SHALL** store the redirect URI value in the resource owner's user-agents (such as browser) session and compare it with the redirect URI that the authorization response was received at, where, if the URIs do not match, the client **SHALL** terminate the process with error;
5. (withdrawn); and
6. **SHALL** implement an effective CSRF protection.
   Further, if it is desired to obtain a persistent identifier of the authenticated user, then the public client
7. **SHALL** include openid in the scope value; and
8. **SHALL** include the nonce parameter defined in Section 3.1.2.1 of **OIDC** in the authentication request.
   If openid is not in the scope value, then the public client
9. **SHALL** include the state parameter defined in Section 4.1.1 of **RFC6749**;

10. **SHALL** verify that the scope received in the token response is either an exact match, or contains a subset of the scope sent in the authorization request; and
11. **SHALL** only use Authorization Server metadata obtained from the metadata document published by the Authorization Server at its well known endpoint as defined in **OIDD** or **RFC8414**.
    NOTE: Adherence to **RFC7636** means that the token request includes code_verifier parameter in the request.

5.2.4. Confidential client

In addition to the provisions for a public client, a confidential client

1. **SHALL** support the following methods to authenticate against the token endpoint:
    1. Mutual TLS for OAuth Client Authentication as specified in Section 2 of **MTLS**, and
    2. client_secret_jwt or private_key_jwt as specified in Section 9 of **OIDC**;
2. **SHALL** use RSA keys with a minimum 2048 bits if using RSA cryptography;
3. **SHALL** use elliptic curve keys with a minimum of 160 bits if using Elliptic Curve cryptography; and
4. **SHALL** verify that its client secret has a minimum of 128 bits if using symmetric key cryptography.

---

6. Accessing Protected Resources

6.1. Introduction

The FAPI endpoints are OAuth 2.0 protected resource endpoints that return protected information for the resource owner associated with the submitted access token.

6.2. Baseline access provisions

6.2.1. Protected resources provisions

The resource server with the FAPI endpoints

1. **SHALL** support the use of the HTTP GET method as in Section 4.3.1 of **RFC7231**;
2. **SHALL** accept access tokens in the HTTP header as in Section 2.1 of OAuth 2.0 Bearer Token Usage **RFC6750**;

3. **SHALL NOT** accept access tokens in the query parameters stated in Section 2.3 of OAuth 2.0 Bearer Token Usage **RFC6750**;
4. **SHALL** verify that the access token is neither expired nor revoked;
5. **SHALL** verify that the scope associated with the access token authorizes access to the resource it is representing;
6. **SHALL** identify the associated entity to the access token;
7. **SHALL** only return the resource identified by the combination of the entity implicit in the access and the granted scope and otherwise return errors as in Section 3.1 of **RFC6750**;
8. **SHALL** encode the response in UTF-8 if applicable;
9. **SHALL** send the Content-type HTTP header Content-Type: application/json if applicable;
10. **SHALL** send the server date in HTTP Date header as in Section 7.1.1.2 of **RFC7231**;
11. **SHALL** set the response header x-fapi-interaction-id to the value received from the corresponding FAPI client request header or to a **RFC4122** UUID value if the request header was not provided to track the interaction, e.g., x-fapi-interaction-id: c770aef3-6784-41f7-8e0e-ff5f97bddb3a;
12. **SHALL** log the value of x-fapi-interaction-id in the log entry; and
13. **SHALL NOT** reject requests with a x-fapi-customer-ip-address header containing a valid IPv4 or IPv6 address.
    NOTE: While this document does not specify the exact method to obtain the entity associated with the access token and the granted scope, the protected resource can use OAuth Token Introspection **RFC7662**.
    Further, the resource server
14. **SHOULD** support the use of Cross Origin Resource Sharing (CORS) [CORS] and or other methods as appropriate to enable JavaScript clients to access the endpoint if it decides to provide access to JavaScript clients.
    NOTE: Providing access to JavaScript clients has other security implications. Before supporting those clients **RFC6819** **SHOULD** be consulted.

6.2.2. Client provisions

The client supporting this document

1. **SHALL** send access tokens in the HTTP header as in Section 2.1 of OAuth 2.0 Bearer Token Usage **RFC6750**; and
2. (withdrawn);
   Further, the client

3. may send the last time the customer logged into the client in the x-fapi-auth-date header where the value is supplied as a HTTP-date as in Section 7.1.1.1 of **RFC7231**, e.g., x-fapi-auth-date: Tue, 11 Sep 2012 19:43:31 GMT;
4. may send the customer's IP address if this data is available in the x-fapi-customer-ip-address header, e.g., x-fapi-customer-ip-address: 2001:DB8::1893:25c8:1946 or x-fapi-customer-ip-address: 198.51.100.119; and
5. may send the x-fapi-interaction-id request header, in which case the value **SHALL** be a RFC4122 UUID to the server to help correlate log entries between client and server, e.g., x-fapi-interaction-id: c770aef3-6784-41f7-8e0e-ff5f97bddb3a.

---

7. Security considerations

7.1. TLS and DNSSEC considerations

As confidential information is being exchanged, all interactions **SHALL** be encrypted with TLS (HTTPS).

The recommendations for Secure Use of Transport Layer Security in **BCP195** **SHALL** be followed, with the following additional requirements:

1. TLS version 1.2 or later **SHALL** be used for all communications.
2. A TLS server certificate check **SHALL** be performed, as per **RFC6125**.

Endpoints for the use by web browsers **SHOULD** use mechanisms to ensure that connections cannot be downgraded using TLS Stripping attacks. A preloaded HTTP Strict Transport Security policy (see **PRELOAD** and **RFC6797**) can be used for this purpose. Some top-level domains, like .bank and .insurance, have set such a policy and therefore protect all second-level domains below them.

For a comprehensive protection against network attackers, all endpoints **SHOULD** additionally use DNSSEC to protect against DNS spoofing attacks that can lead to the issuance of rogue domain-validated TLS certificates.

NOTE: Even if an endpoint uses only organization validated (OV) or extended validation (EV) TLS certificates, rogue domain-validated certificates can be used to impersonate the endpoints and conduct man-in-the-middle attacks. CAA records **RFC8659** can help to mitigate this risk.

7.2. Message source authentication failure

Authorization request and response are not authenticated. For higher risk scenarios, they **SHOULD** be authenticated. See **Financial-grade API Security Profile 1.0 - Part 2: Advanced**, which uses request objects to achieve the message source authentication.

7.3. Message integrity protection failure

The authorization request does not have message integrity protection and hence request tampering and parameter injection are possible. Where such protection is desired, **Financial-grade API Security Profile 1.0 - Part 2: Advanced** **SHOULD** be used.

The response is integrity protected when the ID Token is returned from the authorization endpoint.

7.4. Message containment failure

7.4.1. Authorization request and response

In this document, the authorization request is not encrypted. Thus, it is possible to leak the information contained if the web browser is compromised.

Authorization response can be encrypted as ID Token can be encrypted.

It is possible to leak the information through the logs if the parameters were recorded in the logs and the access to the logs are compromised. Strict access control to the logs in such cases **SHOULD** be enforced.

7.4.2. Token request and response

It is possible to leak information through the logs if the parameters were recorded in the logs and the access to the logs are compromised. Strict access control to the logs in such cases **SHOULD** be enforced.

7.4.3. Resource request and response

Care **SHOULD** be taken so that the sensitive data will not be leaked through the referrer.

If the access token is a bearer token, it is possible to exercise the stolen token. Since the access token can be used against multiple URIs, the risk of leaking is much larger than the refresh token, which is used only against the token endpoint. Thus, the lifetime of the access token **SHOULD** be much shorter than that of the refresh token. Refer to Section 16.18 of **OIDC** for more discussion on the lifetimes of access tokens and refresh tokens.

7.5. Native Apps

When native apps are used as either public clients, dynamically registered confidential clients or user-agents receiving the authorization response for a server based confidential

client, the recommendations for OAuth 2.0 for Native Apps in **BCP212** **SHALL** be followed, with the following additional requirements:

When registering redirect URIs, authorization servers

1. **SHALL NOT** support "Private-Use URI Scheme Redirection"; and
2. **SHALL NOT** support "Loopback Interface Redirection".

These requirements mean that FAPI Security Profile 1.0 compliant implementations can only support native apps through the use of "Claimed https Scheme URI Redirection".

NOTE: Nothing in this document seeks to disallow fixed urls in the form https://localhost:port-number/callback, as these are particularly useful in non-production systems or in clients used in development, to facilitate faster and easier development.

7.6. Incomplete or incorrect implementations of the specifications

To achieve the full security benefits, it is important the implementation of this specification, and the underlying OpenID Connect and OAuth specifications, are both complete and correct.

The OpenID Foundation provides tools that can be used to confirm that an implementation is correct:

https://openid.net/certification/

The OpenID Foundation maintains a list of certified implementations:

https://openid.net/developers/certified/

Deployments that use this specification **SHOULD** use a certified implementation.

7.7. Discovery & Multiple Brands

Organizations who need to support multiple "brands" with individual authorization endpoints from a single Authorization Server deployment **SHALL** use a separate issuer per brand. This can be achieved either at the domain level (e.g. https://brand-a.auth.example.com and https://brand-b.auth.example.com) or with different paths (e.g. https://auth.example.com/brand-a and https://auth.example.com/brand-b)

As stated in 5.2.2-22 Clients **SHALL** only use metadata values obtained via metadata documents as defined in **OIDD**. Communicating metadata through other means (e.g. via email) opens up a social engineering attack vector.

Note that the requirement to use **OIDD** is not a requirement to support Dynamic Client Registration.

---

8. Privacy considerations

8.1. Introduction

There are many factors to be considered in terms of privacy when implementing this document. However, since this document is a profile of OAuth and OpenID Connect, all of them are generic and apply to OAuth or OpenID Connect and are not specific to this document. Implementers are advised to perform a thorough privacy impact assessment and manage identified risks appropriately.

NOTE: Implementers can consult documents like **ISO29100** and [ISO29134] for this purpose.

Privacy threats to OAuth and OpenID Connect implementations include the following:

- (Inappropriate privacy notice) A privacy notice provided at a policy_url or by other means can be inappropriate.
- (Inadequate choice) Providing a consent screen without adequate choices does not form consent.
- (Misuse of data) An AS, RS or Client can potentially use the data not according to the purpose that was agreed.
- (Collection minimization violation) A client asking for more data than it absolutely needs to fulfil the purpose is violating the collection minimization principle.
- (Unsolicited personal data from the Resource) Some bad resource server implementations may return more data than was requested. If the data is personal data, then this would be a violation of privacy principles.
- (Data minimization violation) Any process that is processing more data than it needs is violating the data minimization principle.
- (RP tracking by AS/OP) AS/OP identifying what data is being provided to which Client/RP.
- (User tracking by RPs) Two or more RPs correlating access tokens or ID Tokens to track users.
- (RP misidentification by User at AS) User misunderstands who the RP is due to a confusing representation of the RP at the AS's authorization page.
- (Mismatch between User’s understanding or what RP is displaying to a user and the actual authorization request) To enhance the trust of the ecosystem, best practice is for the AS to make clear what is included in the authorization request (for example, what data will be released to the RP).
- (Attacker observing personal data in authorization request) Authorization request might contain personal data. This can be observed by an attacker.

- (Attacker observing personal data in authorization endpoint response) In some frameworks, even state is deemed personal data. This can be observed by an attacker through various means.
- (Data leak from AS) AS stores personal data. If AS is compromised, these data can leak or be modified.
- (Data leak from Resource) Some resource servers store personal data. If a resource server is compromised, these data can leak or be modified.
- (Data leak from Clients) Some clients store personal data. If the client is compromised, these data can leak or be modified.

These threats can be mitigated by choosing appropriate options in OAuth or OpenID, or by introducing some operational rules. For example, "Attacker observing personal data in authorization request" can be mitigated by either using authorization request by reference using request_uri or by encrypting the request object. Similarly, "Attacker observing personal data in authorization endpoint response" can be mitigated by encrypting the ID Token or JARM response.

---

9. Acknowledgement

The following people contributed to this document:

- Nat Sakimura (NAT Consulting) -- Chair, Editor
- Anoop Saxena (Intuit) -- Co-chair, FS-ISAC Liaison
- Anthony Nadalin (Microsoft) -- Co-chair, SC 27 Liaison
- Edmund Jay (Illumila) -- Co-editor
- Dave Tonge (Moneyhub) -- Co-chair, UK Implementation Entity Liaison
- Paul A. Grassi (NIST) -- X9 Liaison
- Joseph Heenan (Authlete)
- Sascha H. Preibisch (CA)
- Henrik Biering (Peercraft)
- Anton Taborszky (Deutsche Telecom)
- John Bradley (Yubico)
- Tom Jones (Independent)
- Axel Nennker (Deutsche Telekom)
- Daniel Fett (yes.com)
- Torsten Lodderstedt (yes.com)

- Ralph Bragg (Raidiam)
- Brian Campbell (Ping Identity)
- Dima Postnikov (Independent)
- Stuart Low (Biza.io)
- Takahiko Kawasaki (Authlete)
- Vladimir Dzhuvinov (Connect2Id)
- Chris Michael (Open Banking)
- Freddi Gyara (Open Banking)
- Rob Otto (Ping Identity)
- Francis Pouatcha (adorsys)
- Kosuke Koiwai (KDDI)
- Bjorn Hjelm (Verizon)
- Lukasz Jaromin (Cloudentity)
- James Manger

---

10. Bibliography

- **Part2** Financial-grade API Security Profile 1.0 - Part 2: Advanced
- **ISODIR2** ISO/IEC Directives Part 2
- **ISO29100** ISO/IEC 29100 Information technology — Security techniques — Privacy framework
- [ISO29134] ISO/IEC 29134 Information technology — Security techniques — Guidelines for privacy impact assessment
- **RFC4122** A Universally Unique IDentifier (UUID) URN Namespace
- **RFC6749** The OAuth 2.0 Authorization Framework
- **RFC6750** The OAuth 2.0 Authorization Framework: Bearer Token Usage
- **RFC6797** HTTP Strict Transport Security (HSTS)
- **RFC7636** Proof Key for Code Exchange by OAuth Public Clients
- **RFC7662** OAuth 2.0 Token Introspection

- **RFC6125** Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)
- **BCP212** OAuth 2.0 for Native Apps
- **RFC6819** OAuth 2.0 Threat Model and Security Considerations
- **RFC8414** OAuth 2.0 Authorization Server Metadata
- **RFC8659** DNS Certification Authority Authorization (CAA) Resource Record
- **OIDD** OpenID Connect Discovery 1.0 incorporating errata set 1
- **BCP195** Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
- **OIDC** OpenID Connect Core 1.0 incorporating errata set 1
- **X.1254** Entity authentication assurance framework
- **MTLS** OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens
- **PRELOAD** HSTS Preload List Submission

Appendix A. Copyright notice & license

other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

Authors' Addresses

Nat Sakimura

Nat Consulting

Email: nat@nat.consulting

URI: http://nat.sakimura.org/

John Bradley

Yubico

Email: ve7jtb@ve7jtb.com

URI: http://www.thread-safe.com/

Illumila

Illumila

Email: ejay@mgi1.com

URI: http://illumi.la/

Exhibit C: OpenID Connect Client Initiated Backchannel Authentication Flow – Core 1.0

Note: Content in this section may also be found at OpenID Connect Client Initiated Backchannel Authentication Flow – Core 1.0 draft-03, and is provided here in this exhibit for convenience.

Abstract

OpenID Connect Client Initiated Backchannel Authentication Flow is an authentication flow like OpenID Connect. However, unlike OpenID Connect, there is direct Relying Party to OpenID Provider communication without redirects through the user's browser. This specification has the concept of a Consumption Device (on which the user interacts with the Relying Party) and an Authentication Device (on which the user authenticates with the OpenID Provider and grants consent). This specification allows a Relying Party that has an identifier for a user to obtain tokens from the OpenID Provider. The user starts the flow with the Relying Party at the Consumption Device, but authenticates and grants consent on the Authentication Device.

Table of Contents

1. Introduction

OpenID Connect allows Relying Parties (RP) to authenticate their users for clients of all types, including browser-based JavaScript and native mobile apps, to launch sign-in flows and receive verifiable assertions about the identity of signed-in users.

In all of these flows initiated by the RP, the end-user interaction from the consumption device is required and, they are based on HTTP redirection mechanisms. However, some use cases not covered by these flows have been raised, where the RP needs to be the initiator of the user authentication flow and end-user interaction from the consumption device is not needed.

Client Initiated Backchannel Authentication (CIBA) is a new authentication flow in which RPs, that can obtain a valid identifier for the user they want to authenticate, will be able to initiate an interaction flow to authenticate their users without having end-user interaction from the consumption device. The flow involves direct communication from the Client to the OpenID Provider without redirect through the user's browser (consumption device).

This specification does not change the semantics of the OpenID Connect Authentication flow. It introduces a new endpoint to which the authentication request is posted. It introduces a new asynchronous method for authentication result notification or delivery. It does not introduce new scope values nor does it change the semantics of standard OpenID Connect parameters.

As the user does not provide authentication credentials directly to the consumption device, supporting this flow requires the to OP have some mechanism of initiating user authentication out-of-band from the interaction with the consumption device.

1.1. Requirements Notation and Conventions

The keywords "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in IETF RFC2119.

Throughout this document, values are quoted to indicate that they are to be taken literally. When using these values in protocol messages, the quotes **MUST NOT** be used as part of the value.

2. Terminology

This specification uses the terms "OpenID Provider (OP)" and "Relying Party (RP)" as defined by OpenID Connect Core. Furthermore, it uses the term "Client" as defined by OAuth 2.0. OAuth 2.0 Authorization Servers implementing OpenID Connect and CIBA are also referred to as OpenID Providers (OPs). OAuth 2.0 Clients using OpenID Connect and CIBA are also referred to as Relying Parties (RPs). This specification also uses the following terms:

Consumption Device (CD)

The Consumption Device is the device that helps the user consume the service. In the CIBA use case, the user is not necessarily in control of the CD. For example, the CD may be in the

control of an RP agent (e.g. at a bank teller) or may be an RP controlled device (e.g. a petrol pump).

Authentication Device (AD)

The device on which the user will authenticate and authorize the request, often a smartphone.

3. Overview

Client Initiated Backchannel Authentication (CIBA) enables a Client to initiate the authentication of an end-user by means of out-of-band mechanisms.

1. The Client **SHALL** make an "HTTP POST" request to the Backchannel Authentication Endpoint to ask for end-user authentication.
2. The OP will respond immediately with a unique identifier that identifies that authentication while it tries to authenticate the user in the background.
3. The Client will receive the ID Token, Access Token and optionally Refresh Token by means of either the Poll, Ping or Push modes, this choice **MUST** be established by the Client at registration time.

   Poll Mode

   When configured in Poll mode, the Client will poll the token endpoint to get a response with the tokens.

   Ping Mode

   When configured in Ping mode, the OP will send a request to a callback URI previously registered by the Client with the unique identifier returned from the Backchannel Authentication Endpoint. Upon receipt of the notification, the Client makes a request to the token endpoint to obtain the tokens.

   Push Mode

   When configured in Push mode, the OP will send a request with the tokens to a callback URI previously registered by the Client.

4. Registration and Discovery Metadata

Grant Type

This specification introduces the CIBA grant type (an extension grant type as defined by Section 4.5 of OAuth 2.0) with the value: `urn:openid:params:grant-type:ciba`

OpenID Provider Metadata

The following authorization server metadata parameters are introduced by this specification for OPs publishing their support of the CIBA flow and details thereof.

- backchannel_token_delivery_modes_supported: **REQUIRED**. JSON array containing one or more of the following values: poll, ping and push.
- backchannel_authentication_endpoint: **REQUIRED**. URL of the OP's Backchannel Authentication Endpoint as defined in Section 7.
- backchannel_authentication_request_signing_alg_values_supported: **OPTIONAL**. JSON array containing a list of the JWS signing algorithms (alg values) supported by the OP for signed authentication requests, which are described in Section 7.1.1. If omitted, signed authentication requests are not supported by the OP.
- backchannel_user_code_parameter_supported: **OPTIONAL**. Boolean value specifying whether the OP supports use of the user_code parameter, with true indicating support. If omitted, the default value is false.

The CIBA grant type is used in the grant_types_supported field of discovery metadata for OPs that support the ping or poll delivery modes.

The supported client authentication methods and, when applicable, the associated JWS signing algorithms of the OP's Backchannel Authentication Endpoint are the same as those indicated by
the token_endpoint_auth_methods_supported and token_endpoint_auth_signing_alg_values_supported metadata parameters respectively.

Client Metadata

Clients registering to use CIBA **MUST** indicate a token delivery mode. When using the ping or poll mode, the Client **MUST** include the CIBA grant type in the "grant_types" field. When using the ping or push mode, the Client **MUST** register a client notification endpoint. Clients intending to send signed authentication requests **MUST** register the signature algorithm that will be used. The following parameters are introduced by this specification:

- backchannel_token_delivery_mode: **REQUIRED**. One of the following values: poll, ping or push.
- backchannel_client_notification_endpoint: **REQUIRED** if the token delivery mode is set to ping or push. This is the endpoint to which the OP will post a notification after a successful or failed end-user authentication. It **MUST** be an HTTPS URL.
- backchannel_authentication_request_signing_alg: **OPTIONAL**. The JWS algorithm alg value that the Client will use for signing authentication request, as described in Section 7.1.1. When omitted, the Client will not send signed authentication requests.
- backchannel_user_code_parameter: **OPTIONAL**. Boolean value specifying whether the Client supports the user_code parameter. If omitted, the default value is false. This parameter only applies when OP parameter backchannel_user_code_parameter_supported is true.

The token_endpoint_auth_method indicates the registered authentication method for the client to use when making direct requests to the OP, including requests to both the token endpoint and the backchannel authentication endpoint.

Poll and Ping Modes with Pairwise Identifiers

In order to use the Poll or Ping mode with Pairwise Pseudonymous Identifiers (PPIDs), the Client needs to register a URI that is of its ownership and use it during the authentication process in a way that demonstrates that the URI belongs to it, which allows the OP to consider the host component of that URI as the Sector Identifier for the pairwise identifier calculation per Section 8.1 of OpenID Connect Core.

In OpenID Connect Core the sector_identifier_uri contains a document with a list of redirect_uris and the Sector Identifier is defined as either the host component of the sector_identifier_uri or if this is not provided then the host component of the redirect_uri.

In CIBA Poll and Ping modes the jwks_uri is used in place of the redirect_uri. In CIBA Push mode the backchannel_client_notification_endpoint is used in place of the redirect_uri. In situations where the PPID must be shared among multiple RPs, then a sector_identifier_uri can be registered. This specification extends the purpose of the sector_identifier_uri such that it can contain jwks_uris and backchannel_client_notification_endpoints as well as redirect_uri.

In order to support Pairwise Pseudonymous Identifiers in Ping and Poll modes, the RP **MUST** provide either a sector_identifier_uri or a jwks_uri at the registration phase when the `urn:openid:params:grant-type:ciba` grant type is registered. In that way the OpenID Provider can use the host component of the sector_identifier_uri or jwks_uri as the Sector Identifier to generate the PPIDs for the Client.

When an OpenID Provider that supports PPIDs receives a dynamic registration request for a Client that indicates that it wishes to use the Poll or Ping CIBA modes, it **MUST** check if a valid jwks_uri is set when the subject_type is pairwise. If a sector_identifier_uri is explicitly provided, then the jwks_uri **MUST** be included in the list of URIs pointed to by the sector_identifier_uri.

But having registered a "jwks_uri" is not enough to use PPIDs, Client needs somehow to demonstrate that such "jwks_uri" belongs to it, which can be accomplished by proving possession of a private key corresponding to one of the public keys published at the "jwks_uri". Such proof can be demonstrated with signed authentication requests using the asymmetric keys provided by the "jwks_uri" or by authenticating to the OP using one of the following two mechanisms in conjunction with a key from its "jwks_uri":

1. Using the Self-Signed Certificate Mutual TLS OAuth Client Authentication Method as defined in section 2.2 of [I-D.ietf-oauth-mtls].
2. Using the private_key_jwt method as per the section 9 Client Authentication of [OpenID.Core].

Push Mode with Pairwise Identifiers

When using the Push mode, the PPIDs will use the host component of the "backchannel_client_notification_endpoint" as the Sector Identifier. In case a "sector_identifier_uri" is explicitly provided, then the "backchannel_client_notification_endpoint" **MUST** be included in the list of URIs pointed to by the "sector_identifier_uri".

The following is a non-normative example from a dynamic registration request that contains the CIBA grant type as required and a "jwks_uri" (with line wraps within values for display purposes only).

```
POST /connect/register HTTP/1.1

Content-Type: application/json

Accept: application/json

Host: server.example.com

Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ...



{

"application_type": "web",

    "client_name": "My Example",

    "logo_uri": "https://client.example.org/logo.png",

    "subject_type": "pairwise",

    "token_endpoint_auth_method": "private_key_jwt",

    "grant_types": ["urn:openid:params:grant-type:ciba"],

    "backchannel_token_delivery_mode": "poll",

    "jwks_uri":
"https://client.example.org/my_public_keys.jwks",

    "contacts": ["ve7jtb@example.org", "mary@example.org"]

}
```

5. Poll, Ping and Push Modes

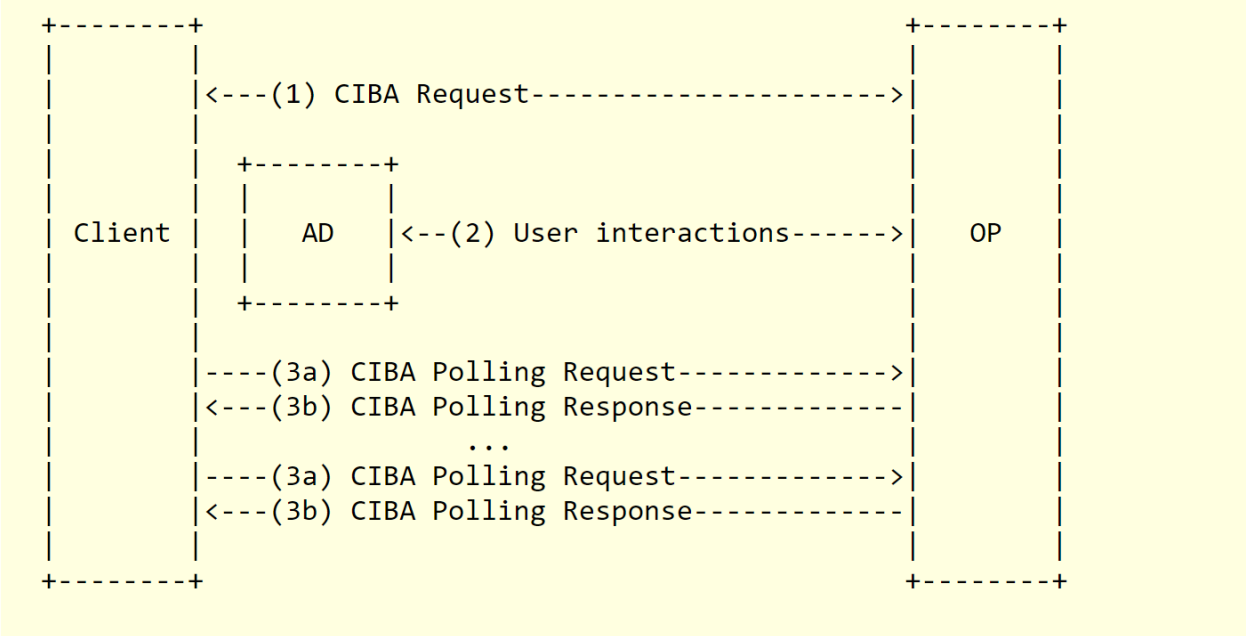
This specification allows the Client to get the authentication result in three ways: poll, ping or push.

In the Poll mode, the authentication result is retrieved by the Client by polling the OP's token endpoint using the new grant type.

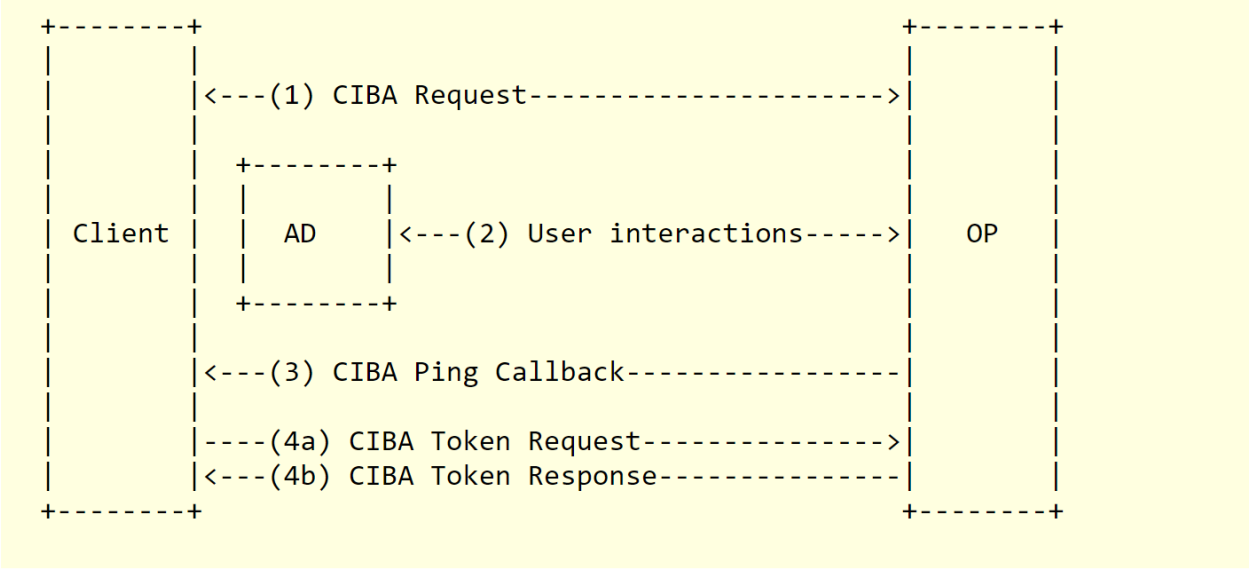
In the Ping mode, the OP will post the unique identifier of the authentication session to the Client, the Client will then retrieve the authentication result from the token endpoint using the new grant type.

In the Push mode, the OP will post the full authentication result to the Client.

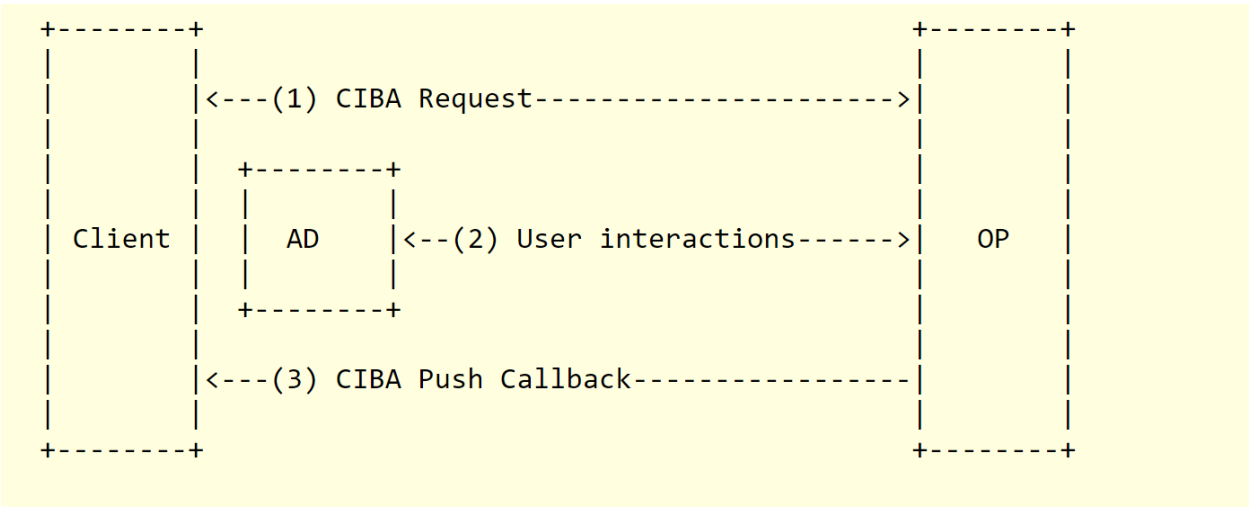
CIBA Poll Mode is illustrated in the following diagram:



CIBA Ping Mode is illustrated in the following diagram:



CIBA Push Mode is illustrated in the following diagram:



6. Example Use Cases

The following use cases are non-normative examples to illustrate the usage of this specification.

1. A call center agent wants to authenticate a caller. Using additional scopes like e.g. "profile" or "phone" the call center agent would get access to claims about the user like "phone_number" and "phone_number_verified".
2. A bank teller wants to authenticate a customer in a bank branch - using CIBA for authentication in a face-to-face scenario.
3. A user wants to use their smartphone to authorize a payment they are making at a point of sale terminal.

7. Backchannel Authentication Endpoint

The Backchannel Authentication Endpoint is used to initiate an out-of-band authentication of the end-user. This is done by sending an HTTP POST message directly from the Client to the OpenID Provider's Backchannel Authentication Endpoint, using a request defined in the following subsections.

Communication with the Backchannel Authentication Endpoint **MUST** utilize TLS. See Section 16.17 [OpenID.Core] for more information on using TLS.

7.1. Authentication Request

Client Initiated Backchannel Authentication defines an authentication request that is requested directly from the Client to the OpenID Provider without going through the user's browser. The Client **MUST** send an authentication request to the OpenID Provider by building an "HTTP POST" request that will take to the OpenID Provider all the information needed to authenticate the user without asking them for their identifier.

The Client **MUST** authenticate to the Backchannel Authentication Endpoint using the authentication method registered for its client_id, such as the authentication methods from Section 9 of [OpenID.Core] or authentication methods defined by extension in other specifications. Note that there's some potential ambiguity around the appropriate audience value to use when JWT client assertion based authentication is employed. To address that ambiguity the Issuer Identifier of the OP **SHOULD** be used as the value of the audience. In order to facilitate interoperability the OP **MUST** accept its Issuer Identifier, Token Endpoint URL, or Backchannel Authentication Endpoint URL as values that identify it as an intended audience.

An authentication request is composed of the following parameters and **MAY** contain additional parameters defined by extension or profile:

Scope

REQUIRED. The scope of the access request as described by Section 3.3 of RFC6749. OpenID Connect implements authentication as an extension to OAuth 2.0 by including the openid scope value in the authorization requests. Consistent with that, CIBA authentication requests **MUST** therefore contain the openid scope value. The behavior when the openid scope value is not present is left unspecified by this document, thus allowing for the potential definition of the behavior in a non OpenID context, such as an OAuth authorization flow. Other scope values **MAY** be present, including but not limited to the profile, email, address and phone scope values from Section 5.4 of [OpenID.Core].

client_notification_token

REQUIRED if the Client is registered to use Ping or Push modes. It is a bearer token provided by the Client that will be used by the OpenID Provider to authenticate the callback request to the Client. The length of the token **MUST NOT** exceed 1024 characters and it **MUST** conform to the syntax for Bearer credentials as defined in Section 2.1 of RFC6750. Clients **MUST** ensure that it contains sufficient entropy (a minimum of 128 bits while 160 bits is recommended) to make brute force guessing or forgery of a valid token computationally infeasible - the means of achieving this are implementation specific, with possible

approaches including secure pseudorandom number generation or cryptographically secured self-contained tokens.

acr_values

OPTIONAL. Requested Authentication Context Class Reference values. Space-separated string that specifies the acr values that the OpenID Provider is being requested to use for processing this Authentication Request, with the values appearing in order of preference. The actual means of authenticating the end-user, however, are ultimately at the discretion of the OP and the Authentication Context Class satisfied by the authentication performed is returned as the acr Claim Value of the ID Token. When the acr_values parameter is present in the authentication request, it is highly **RECOMMENDED** that the resulting ID Token contain an acr Claim.

login_hint_token

OPTIONAL. A token containing information identifying the end-user for whom authentication is being requested. The particular details and security requirements for the login_hint_token as well as how the end-user is identified by its content are deployment or profile specific.

id_token_hint

OPTIONAL. An ID Token previously issued to the Client by the OpenID Provider being passed back as a hint to identify the end-user for whom authentication is being requested. If the ID Token received by the Client from the OP was asymmetrically encrypted, to use it as an id_token_hint, the client **MUST** decrypt the encrypted ID Token to extract the signed ID Token contained in it.

login_hint

OPTIONAL. A hint to the OpenID Provider regarding the end-user for whom authentication is being requested. The value may contain an email address, phone number, account number, subject identifier, username, etc., which identifies the end-user to the OP. The value may be directly collected from the user by the Client before requesting authentication at the OP, for example, but may also be obtained by other means.

binding_message

OPTIONAL. A human readable identifier or message intended to be displayed on both the consumption device and the authentication device to interlock them together for the transaction by way of a visual cue for the end-user. This interlocking message enables the end-user to ensure that the action taken on the authentication device is related to the request initiated by the consumption device. The value **SHOULD** contain something that enables the end-user to reliably discern that the transaction is related across the consumption device and the authentication device, such as a random value of reasonable entropy (e.g. a transactional approval code). Because the various devices involved may have limited display abilities and the message is intending for visual inspection by the end-user, the binding_message value **SHOULD** be relatively short and use a limited set of plain text

characters. The invalid_binding_message defined in Section 13 is used in the case that it is necessary to inform the Client that the provided binding_message is unacceptable.

user_code

OPTIONAL. A secret code, such as password or pin, known only to the user but verifiable by the OP. The code is used to authorize sending an authentication request to user's authentication device. This parameter **SHOULD** only be present if client registration parameter backchannel_user_code_parameter indicates support for user code.

requested_expiry

OPTIONAL. A positive integer allowing the client to request the expires_in value for the auth_req_id the server will return. The server **MAY** use this value to influence the lifetime of the authentication request and is encouraged to do so where it will improve the user experience, for example by terminating the authentication when as it knows the client is no longer interested in the result.

As in the CIBA flow the OP does not have an interaction with the end-user through the consumption device, it is **REQUIRED** that the Client provides one (and only one) of the hints specified above in the authentication request, that is "login_hint_token", "id_token_hint" or "login_hint".

An authentication request is made using the HTTP POST method with the aforementioned parameters in the application/x-www-form-urlencoded format and a character encoding of UTF-8 in the HTTP request entity-body. When applicable, additional parameters required by the given client authentication method are also included (e.g. JWT assertion based client authentication uses client_assertion and client_assertion_type while Mutual TLS client authentication uses client_id).

The following is a non-normative example of an authentication request (with line wraps within values for display purposes only):

```
POST /bc-authorize HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded



scope=openid%20email%20example-scope&

client_notification_token=8d67dc78-7faa-4d41-aabd-67707b374255&

binding_message=W4SCT&

login_hint_token=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJ
```

```
zdWJfaWQiOnsic3ViamVjdF90eXBlIjoicGhvbmUiLCJwaG9uZSI6IisxMzMwMjg
xODAwNCJ9fQ.Kk8jcUbHjJAQkRSHyDuFQr3NMEOSJEZc85VfER74tX6J9CuUllr8
9WKUHUR7MA0-mWlptMRRhdgW1ZDt7g1uwQ&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJ
pc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0dHB
zOi8vc2VydmVyLmV4YW1wbGUuY29tIiwianRpIjoiYmRjLVhzX3NmLTNZTW80RlN
6SUoyUSIsImlhdCI6MTUzNzgxOTQ4NiwiZXhwIjoxNTM3ODE5Nzc3fQ.Ybr8mg_3
E2OptOSsA8rnelYO_y1L-yFaF_j1iemM3ntB61_GN3APe5cl_-5a6cvGlP154XAK
7fL-GaZSdnd9kg
```

7.1.1. Signed Authentication Request

A signed authentication request is made by encoding all of the authentication request parameters as claims of a signed JWT with each parameter name as the claim name and its value as a JSON string. An exception to this is requested_expiry, which may be sent as either a JSON string or a JSON number, the OP **MUST** accept either type. An extension or profile may define additional authentication request parameters, these may be defined to be any JSON type.

The JWT **MUST** contain all of the authentication request parameters. The JWT **MUST** be secured with an asymmetric signature and follow the guidance from Section 10.1 of [OpenID.Core] regarding asymmetric signatures. The JWT **MUST** also contain the following RFC7519 registered claims:

aud

The Audience claim **MUST** contain the value of the Issuer Identifier for the OP, which identifies the Authorization Server as an intended audience.

iss

The Issuer claim **MUST** be the client_id of the OAuth Client.

exp

An expiration time that limits the validity lifetime of the signed authentication request.

iat

The time at which the signed authentication request was created.

nbf

The time before which the signed authentication request is unacceptable.

jti

A unique identifier for the signed authentication request.

The signed authentication request JWT is passed as an application/x-www-form-urlencoded HTTP request parameter with the name request. Authentication request parameters **MUST NOT** be present outside of the JWT, in particular they **MUST NOT** appear as HTTP request parameters. Additional HTTP request parameters as required by the given client authentication method, however, **MUST** be included as application/x-www-form-urlencoded parameters (e.g. Mutual TLS client authentication uses client_id while JWT assertion based client authentication uses client_assertion and client_assertion_type).

For example, a signed authentication request using the same authentication request parameters and values as the example from the previous section would look like the following (with line wraps within values for display purposes only):

```
POST /bc-authorize HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded


request=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJpc3MiOiJz

NkJoZFJrcXQzIiwiYXVkIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJl

eHAiOjE1Mzc4MjAwODYsImlhdCI6MTUzNzgxOTQ4NiwibmJmIjoxNTM3ODE4ODg2

LCJqdGkiOiI0TFRDcUFDQzJFU0M1QldDbk4zajU4RW5BIiwic2NvcGUiOiJvcGVu

aWQgZW1haWwgZXhhbXBsZS1zY29wZSIsImNsaWVudF9ub3RpZmljYXRpb25fdG9r

ZW4iOiI4ZDY3ZGM3OC03ZmFhLTRkNDEtYWFiZC02NzcwN2IzNzQyNTUiLCJiaW5k

aW5nX21lc3NhZ2UiOiJXNFNDVCIsImxvZ2luX2hpbnRfdG9rZW4iOiJleUpyYVdR

aU9pSnNkR0ZqWlhOaWR5SXNJbUZzWnlJNklrVlRNalUySW4wLmV5SnpkV0pmYVdR

aU9uc2ljM1ZpYW1WamRGOTBlWEJsSWpvaWNHaHZibVVpTENKd2FHOXVaU0k2SWlz

eE16TXdNamd4T0RBd05DSjlmUS5LazhqY1ViSGpKQVFrUlNIeUR1RlFyM05NRU9T

SkVaYzg1VmZFUjc0dFg2SjlDdVVsbHI4OVdLVUhVUjdNQTAtbVdscHRNUlJoZGdX
```

```
MVpEdDdnMXV3USJ9.RB-iFvzpkQ_gUzg0eutoviViCKyLugjVYfVqdjDZ63U1MZR
Z-KcUNSsBjCVptc-QdljCSNCUyULIzT2R5Nmg4Q&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJ
pc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0dHB
zOi8vc2VydmVyLmV4YW1wbGUuY29tIiwianRpIjoiY2NfMVhzc3NmLTJpOG8yZ1B
6SUprMSIsImlhdCI6MTUzNzgxOTQ4NiwiZXhwIjoxNTM3ODE5Nzc3fQ.PWb_VMzU
IbD_aaO5xYpygnAlhRIjzoc6kxg4NixDuD1DVpkKVSBbBweqgbDLV-awkDtuWnyF
yUpHqg83AUV5TA
```

Where the following is the JWT payload (with line wraps and added whitespace for display purposes only):

```
{
 "iss": "s6BhdRkqt3",
 "aud": "https://server.example.com",
 "exp": 1537820086,
 "iat": 1537819486,
 "nbf": 1537818886,
 "jti": "4LTCqACC2ESC5BWCnN3j58EnA",
 "scope": "openid email example-scope",
 "client_notification_token": "8d67dc78-7faa-4d41-aabd-
67707b374255",
 "binding_message": "W4SCT",
 "login_hint_token": "eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2I
   n0.eyJzdWJfaWQiOnsic3ViamVjdF90eXBlIjoicGhvbmUiLCJwaG9uZSI6I
   isxMzMwMjgxODAwNCJ9fQ.Kk8jcUbHjJAQkRSHyDuFQr3NMEOSJEZc85VfER
```

```
    74tX6J9CuUllr89WKUHUR7MA0-mWlptMRRhdgW1ZDt7g1uwQ"

}
```

Note that encrypted JWT authentication requests are not supported.

7.1.2. User Code

CIBA supports the optional use of "user codes" as a mechanism to prevent unsolicited authentication requests from appearing on a user's authentication device.

A "user code" is a secret known to the user, but which is not the user's password at the OP. When an OP supports this feature then Clients are required to ask the user for their "user code" in order to start a CIBA flow. This prevents malicious Clients or malicious end-users from starting unsolicited CIBA flows for a particular user for whom they know the login_hint or equivalent.

It is optional for the OP to implement User Code functionality. If the OP implements user code functionality then it may allow 1) clients without user code and 2) users without user code.

Typically clients that establish a security context with the user prior to sending a CIBA request **SHOULD** be allowed without the User Code mechanism (e.g. web applications that use CIBA for step-up authentication or call center applications where a caller ID is used to identify the user). In addition clients who don't use static user identifiers as login hints **SHOULD** be allowed without requiring the User Code mechanism.

The OP declares support for user code with the provider metadata parameter backchannel_user_code_parameter_supported.

The Client registration parameter backchannel_user_code_parameter specifies if support for user code is required from the Client.

A client may detect users who require a user code from the authentication request error code. For example a client may first attempt an authentication request without a user code, and only prompt for a user code if it receives the error code missing_user_code.

The Client **MUST NOT** store the user code, but **SHOULD** rather request it from the user for each CIBA flow.

Registering a user code for a user is not in scope of this specification. Examples include a facility provided by the authentication device or another service provided by the OP. OPs **SHOULD** provide a method for the user to change the user code.

7.2. Authentication Request Validation

The OpenID Provider **MUST** validate the request received as follows:

1. Authenticate the Client per the authentication method registered or configured for its client_id. It is **RECOMMENDED** that Clients not send shared secrets in the Authentication Request but rather that public key cryptography be used.
2. If the authentication request is signed, validate the JWT sent with the request parameter, which includes verifying the signature and ensuring that the JWT is valid in all other respects per RFC7519.
3. Validate all the authentication request parameters. In the event the request contains more than one of the hints specified in Authentication Request, the OpenID Provider **MUST** return an "invalid_request" error response as per Section 13.
4. The OpenID Provider **MUST** process the hint provided to determine if the hint is valid and if it corresponds to a valid user. The type, issuer (where applicable) and maximum age (where applicable) of a hint that an OP accepts **SHOULD** be communicated to Clients. How the OP validates hints and informs Clients of its hint requirements is out-of-scope of this specification.
5. If the hint is not valid or if the OP is not able to determine the user then an error **SHOULD** be returned to the Client as per Section Authentication Error Response.
6. The OpenID Provider **MUST** verify that all the **REQUIRED** parameters are present and their usage conforms to this specification.

OpenID Providers **SHOULD** ignore unrecognized request parameters.

If the OpenID Provider encounters any error, it **MUST** return an error response, per Section 13.

7.3. Successful Authentication Request Acknowledgement

If the Authentication Request is validated as per Section Authentication Request Validation, the OpenID Provider will return an HTTP 200 OK response to the Client to indicate that the authentication request has been accepted and it is going to be processed. The body of this response will contain:

auth_req_id

REQUIRED. This is a unique identifier to identify the authentication request made by the Client. It **MUST** contain sufficient entropy (a minimum of 128 bits while 160 bits is recommended) to make brute force guessing or forgery of a valid auth_req_id computationally infeasible - the means of achieving this are implementation specific, with possible approaches including secure pseudorandom number generation or cryptographically secured self-contained tokens. The OpenID Provider **MUST** restrict the characters used to 'A'-'Z', 'a'-'z', '0'-'9', '.', '-' and '_', to reduce the chance of the client incorrectly decoding or re-encoding the auth_req_id; this character set was chosen to allow the server to use unpadded base64url if it wishes. The identifier **MUST** be treated as opaque by the client.

expires_in

REQUIRED. A JSON number with a positive integer value indicating the expiration time of the "auth_req_id" in seconds since the authentication request was received. A Client calling the token endpoint with an expired auth_req_id will receive an error, see Token Error Response.

interval

OPTIONAL. A JSON number with a positive integer value indicating the minimum amount of time in seconds that the Client **MUST** wait between polling requests to the token endpoint. This parameter will only be present if the Client is registered to use the Poll or Ping modes. If no value is provided, clients **MUST** use 5 as the default value.

The following is a non-normative example of an authentication response:

```
HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store


{

  "auth_req_id": "1c266114-a1be-4252-8ad1-04986c5b9ac1",

  "expires_in": 120,

  "interval": 2

}
```

7.4. Authentication Request Acknowledgment Validation

If the Client receives an HTTP 200 OK, it **MUST** validate that all the required parameters are received.

The Client **MUST** keep the auth_req_id in order to validate the callbacks received in Ping and Push modes or to use when making a token request in Poll and Ping modes.

The Client **SHOULD** store the expiration time in order to clean up authentication requests for which no Ping Callback or Push Callback is received.

8. OpenID Provider Obtains End-User Consent/Authorization

After the OP has validated the Authentication Request, the OP identifies the user and chooses a channel to best authenticate the user and authorize the request, in line with the Client's requests regarding acr_values. Typically this involves the OP initiating an interactive session on the user's authentication device.

Once the end-user is authenticated, the OpenID Provider **MUST** obtain an authorization decision as described in Section 3.1.2.4 of [OpenID.Core].

When using the Client Initiated Backchannel Authentication flow, there is no interactive dialogue between the OpenID Provider and the end-user through the user's consumption device. There might be an agent of the Client involved who transfers the binding_message to the user.

9. Client Notification Endpoint

The Client Notification Endpoint is set by the Client during Registration and Discovery Metadata. It is the endpoint the OP will call after a successful or failed end-user authentication.

It **MUST** be an HTTPS URL and Communication with the Client Notification Endpoint **MUST** utilize TLS. See Section 16.17 [OpenID.Core] for more information on using TLS.

When the Client is configured in Ping mode, the endpoint receives a notification from the OP that an Authentication Result is ready to be retrieved from the Token Endpoint.

When the Client is configured in Push mode, the endpoint receives the Authentication Result (an ID Token, an Access Token and, optionally, a Refresh Token or in the event that the user did not grant authorization, an error).

Requests to the Client Notification Endpoint **MUST** be authenticated using a "bearer token" created by the Client and sent to the OP in the Authentication request as the value of the parameter "client_notification_token".

10. Getting the Authentication Result

The token delivery mode for the Client (Poll, Ping or Push) is determined at registration time.

A Client can only register a single token delivery method and the OP **MUST** only deliver the Authentication Result to the Client through the registered mode.

10.1. Token Request Using CIBA Grant Type

If the Client is registered to use Poll or Ping modes, the Client will retrieve the Authentication Result from the token endpoint.

The Client **MUST** be authenticated as specified in Section 9 of [OpenID.Core].

If the Client is registered to use the Poll mode, then the Client polls the token endpoint at reasonable interval, which **MUST NOT** be more frequent than the minimum interval provided by the OpenID Provider via the "interval" parameter (if provided).

For polling requests the OP may implement long polling, where the OP responds to the token request only when the authentication result has become available or a timeout has occurred. A timeout of 30 seconds is recommended for long polling requests, as specified in Section 5.5 of RFC6202, Best Practices for Long Polling.

Clients **SHOULD** be prepared to wait at least 30 seconds for a response when using polling mode. The OP **SHOULD** not take longer than 30 seconds to respond to a request, even when using long polling.

The polling interval is measured from the instant when a polling request is sent. To implement long polling the OP may respond slower than interval. A Client **MUST NOT** send two overlapping requests with same auth_req_id. Rather the client **MUST** always wait until receiving the response to the previous request before sending out the next request. If the interval has passed when the response is received, then the client may immediately send out the next request.

An OP in a meltdown type situation may return an HTTP 503 with a Retry-After response header as described in Section 7.1.3 of HTTP/1.1. Clients **SHOULD** respect such headers and only retry the token request after the time indicated in the header.

Here are some illustations of how a Client **SHOULD** behave based on different OP responses, assuming a default interval of 5s:

Long Polling

The Client makes a token request and the OP returns an authorization_pending error after 30s. In this case the Client can immediately make the next token request.

Standard Polling

The Client makes a token request and the OP returns an authorization_pending error after 2s. In this case the Client **MUST** wait 3s before making the next request.

OP responds slower than 30s

The Client makes a token request and the OP doesn't return any response within 30s. In this case the Client may cancel the request and make a new request.

If the Client is registered to use the Ping mode, then when the Client receives a notification to its Client Notification Endpoint containing an auth_req_id that is verified against a client_notification_token, it **MUST** call the token endpoint to retrieve the authentication result.

NOTE: A Client configured in Ping mode may also poll the token endpoint. The OpenID Provider **MUST** treat such a Client as if it had been registered to use the Poll mode.

The Client makes an "HTTP POST" request to the token endpoint by sending the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type

REQUIRED. Value **MUST** be `urn:openid:params:grant-type:ciba`

auth_req_id

REQUIRED. It is the unique identifier to identify the authentication request (transaction) made by the Client. The OP **MUST** check whether the auth_req_id was issued to this Client in response to an Authentication Request. Otherwise, an error **MUST** be returned.

The following is a non-normative example of a token request (with line wraps within values for display purposes only).

```
POST /token HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded


grant_type=urn%3Aopenid%3Aparams%3Agrant-type%3Aciba&

auth_req_id=1c266114-a1be-4252-8ad1-04986c5b9ac1&

client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A

client-assertion-type%3Ajwt-bearer&

client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.ey

Jpc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0d

HBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwianRpIjoiLV9wMTZqNkhj

aVhvMzE3aHZaMzEyYyIsImlhdCI6MTUzNzgxOTQ5MSwiZXhwIjoxNTM3ODE5Nzg

yfQ.BjaEoqZb-81gE5zz4UYwNpC3QVSeX5XhH176vg35zjkbq3Zmv_UpHB2ZugR

Va344WchTQVpaSSShLbvha4yziA
```

10.1.1. Successful Token Response

After receiving and validating a valid and authorized Token Request from the Client and when the end-user associated with the supplied auth_req_id has been authenticated and has authorized the request, the OpenID Provider returns a successful response as specified in Section 3.1.3.3 of [OpenID.Core]. Once redeemed for a successful token response, the auth_req_id value that was used is no longer valid. If the end-user associated with the supplied auth_req_id has not been authenticated or has not authorized the request, an error response **MUST** be sent as defined in Token Error Response.

The following is a non-normative example of a successful token response (with line wraps within values for display purposes only).

```
HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store



{

 "access_token": "G5kXH2wHvUra0sHlDy1iTkDJgsgUO1bN",

 "token_type": "Bearer",

 "refresh_token": "4bwc0ESC_IAhflf-ACC_vjD_ltc11ne-8gFPfA2Kx16",

 "expires_in": 120,

 "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjE2NzcyNiJ9.eyJpc3MiOiJo


dHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsInN1YiI6IjI0ODI4OTc2MTAwMSIs


ImF1ZCI6InM2QmhkUmtxdDMiLCJlbWFpbCI6ImphbmVkb2VAZXhhbXBsZS5jb20i


LCJleHAiOjE1Mzc4MTk4MDMsImlhdCI6MTUzNzgxOTUwM30.aVq83mdy72ddIFVJ

   LjlNBX-5JHbjmwK-Sn9Mir-
blesfYMceIOw6u4GOrO_ZroDnnbJXNKWAg_dxVynv


MHnk3uJc46feaRIL4zfHf6Anbf5_TbgMaVO8iczD16A5gNjSD7yenT5fslrrW-NU


_vtmi0s1puoM4EmSaPXCR19vRJyWuStJiRHK5yc3BtBlQ2xwxH1iNP49rGAQe_LH

   fW1G74NY5DaPv-V23JXDNEIUTY-jT-
NbbtNHAxnhNPyn8kcO2WOoeIwANO9BfLF1


EFWtjGPPMj6kDVrikec47yK86HArGvsIIwk1uExynJIv_tgZGE0eZI7MtVb2UlCw

   DQrVlg"

}
```

10.2. Ping Callback

If the Client is registered in Ping mode, the OpenID Provider will send an HTTP POST Request to the Client Notification Endpoint after a successful or failed end-user authentication.

In this mode the OP sends the client_notification_token as a bearer token in the Authorization header field and sends only the auth_req_id in the body of the request. The request uses the application/json media type.

The following is a non-normative example of a Ping callback sent as an HTTP POST request to the Client's Notification Endpoint (with line wraps within values for display purposes only).

```
POST /cb HTTP/1.1

Host: client.example.com

Authorization: Bearer 8d67dc78-7faa-4d41-aabd-67707b374255

Content-Type: application/json



{

 "auth_req_id": "1c266114-a1be-4252-8ad1-04986c5b9ac1"

}
```

The Client **MUST** verify the client_notification_token used to authenticate the request is valid and is associated with the auth_req_id received in the Ping callback. If the bearer token is not valid, the Client **SHOULD** return an HTTP 401 Unauthorized response.

For valid requests, the Client Notification Endpoint **SHOULD** respond with an HTTP 204 No Content. The OP **SHOULD** also accept responses with HTTP 200 OK and any body in the response **SHOULD** be ignored.

The Client **MUST NOT** return an HTTP 3xx code. The OP **MUST NOT** follow redirects.

How the OP handles HTTP error codes in the ranges of 4xx and 5xx is out-of-scope of this specification. Administrative action is likely to be needed in these cases.

For valid requests, the Client can now use the received auth_req_id to make a Token Request using the CIBA Grant Type to the Token Endpoint as described in Token Request Using CIBA Grant Type.

10.3. Push Callback

10.3.1. Successful Token Delivery

If the Client is registered in Push mode and the user is well authenticated and has authorized the request, the OpenID Provider delivers a payload that includes an ID Token, an Access Token and, optionally, a Refresh Token to the Client Notification Endpoint.

Error conditions associated with the authentication request are delivered to the Client by sending a Push Error Payload to the Client Notification Endpoint.

The Push Callback uses the parameters defined for a successful token response in Section 4.1.4 of OAuth 2.0 and Section 3.1.3.3 of [OpenID.Core]. In addition a new parameter "auth_req_id" is included in the payload. This is the authentication request identifier as defined in Successful Authentication Response.

The Push Callback uses the application/json media type.

In order to bind together the ID Token, the Access Token and the auth_req_id, the OP **MUST** include the hash value of the Access Token and the auth_req_id within the ID Token using the at_hash and urn:openid:params:jwt:claim:auth_req_id claims respectively. In case a Refresh Token is sent to the Client, the hash value of it **MUST** also be added to the ID token using the urn:openid:params:jwt:claim:rt_hash claim. Section 3.1.3.6 of [OpenID.Core] shows how to calculate the hash value of the access_token for at_hash, the same method can also be applied to calculate the Refresh Token hash value. Note that these claims are only required in Push mode.

The following is a non-normative example of a Push Callback sent as an HTTP POST request to the Client's notification endpoint (with line wraps within values for display purposes only). The request is authenticated through a bearer token that is the value of the "client_notification_token" provided by the Client in the Authentication Request.

```
POST /cb HTTP/1.1

Host: client.example.com

Authorization: Bearer 8d67dc78-7faa-4d41-aabd-67707b374255

Content-Type: application/json


{

 "auth_req_id": "1c266114-a1be-4252-8ad1-04986c5b9ac1",

 "access_token": "G5kXH2wHvUra0sHlDy1iTkDJgsgUO1bN",

 "token_type": "Bearer",

 "refresh_token": "4bwc0ESC_IAhflf-ACC_vjD_ltc11ne-8gFPfA2Kx16",
```

```
 "expires_in": 120,

 "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjE2NzcyNiJ9.eyJpc3MiOiJ


odHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsInN1YiI6IjI0ODI4OTc2MTAwMS


IsImF1ZCI6InM2QmhkUmtxdDMiLCJlbWFpbCI6ImphbmVkb2VAZXhhbXBsZS5jb


20iLCJleHAiOjE1Mzc4MTk4MDMsImlhdCI6MTUzNzgxOTUwMywiYXRfaGFzaCI6


Ild0MGtWRlhNYWNxdm5IZXlVMDAwMXciLCJ1cm46b3BlbmlkOnBhcmFtczpqd3Q


6Y2xhaW06cnRfaGFzaCI6InNIYWhDdVNwWENSZzVta0REdnZyNHciLCJ1cm46b3


BlbmlkOnBhcmFtczpqd3Q6Y2xhaW06YXV0aF9yZXFfaWQiOiIxYzI2NjExNC1hM


WJlLTQyNTItOGFkMS0wNDk4NmM1YjlhYzEifQ.SGB5_a8E7GjwtoYrkFyqOhLK6

   L8-Wh1nLeREwWj30gNYOZW_ZB2mOeQ5yiXqeKJeNpDPssGUrNo-3N-
CqNrbmVCb

   XYTwmNB7IvwE6ZPRcfxFV22oou-NS4-
3rEa2ghG44Fi9D9fVURwxrRqgyezeD3H

   HVIFUnCxHUou3OOpj6aOgDqKI4Xl2xJ0-kKAxNR8LljUp64OHgoS-
UO3qyfOwIk

   IAR7o4OTK_3Oy78rJNT0Y0RebAWyA81UDCSf_gWVBp-
EUTI5CdZ1_odYhwB9OWD


W1A22Sf6rmjhMHGbQW4A9Z822yiZZveuT_AFZ2hi7yNp8iFPZ8fgPQJ5pPpjA7u

   dg"

}
```

The Client **MUST** verify the client_notification_token used to authenticate the request is valid and is associated with the auth_req_id received in the Push Callback. If the bearer token is not valid the Client **SHOULD** return an HTTP 401 Unauthorized response.

The Client **MUST** validate the ID Token, which acts as a detached signature, as per Section 3.1.3.7 of [OpenID.Core].

The Client **MUST** ensure that the value of the urn:openid:params:jwt:claim:auth_req_id claim in the ID Token matches the auth_req_id in the request.

The Client **MUST** validate the access token received using the at_hash in the ID Token as per Section 3.2.2.9 of [OpenID.Core]. If a refresh token is present, the Client **MUST** validate it using the urn:openid:params:jwt:claim:rt_hash in the ID Token in a similar manner as the access token is validated.

For valid requests, the Client Notification Endpoint **SHOULD** respond with an HTTP 204 No Content. The OP **SHOULD** also accept HTTP 200 OK and any body in the response **SHOULD** be ignored.

The Client **MUST NOT** return an HTTP 3xx code. The OP **MUST NOT** follow redirects.

How the OP handles HTTP error codes in the ranges of 4xx and 5xx is out-of-scope of this specification. Administrative action is likely to be needed in these cases.

The following is a non-normative example of a base64url decoded ID Token sent to the client notification endpoint:

```
{
  "iss": "https://server.example.com",
  "sub": "248289761001",
  "aud": "s6BhdRkqt3",
  "email": "janedoe@example.com",
  "exp": 1537819803,
  "iat": 1537819503,
  "at_hash": "Wt0kVFXMacqvnHeyU0001w",
  "urn:openid:params:jwt:claim:rt_hash":
"sHahCuSpXCRg5mkDDvvr4w",
  "urn:openid:params:jwt:claim:auth_req_id":
    "1c266114-a1be-4252-8ad1-04986c5b9ac1"
}
```

11. Token Error Response

If the Token Request is invalid or unauthorized, the OpenID Provider constructs an error response according to Section 3.1.3.4 Token Error Response of [OpenID.Core]. In addition to the error codes defined in Section 5.2 of RFC6749, the following error codes defined in the OAuth Device Flow are also applicable:

authorization_pending

The authorization request is still pending as the end-user hasn't yet been authenticated.

slow_down

A variant of "authorization_pending", the authorization request is still pending and polling **SHOULD** continue, but the interval **MUST** be increased by at least 5 seconds for this and all subsequent requests.

expired_token

The auth_req_id has expired. The Client will need to make a new Authentication Request.

access_denied

The end-user denied the authorization request.

If the auth_req_id is invalid or was issued to another Client, an invalid_grant error **MUST** be returned as described in Section 5.2 of RFC6749.

If a Client continually polls quicker than the interval, the OP may return an invalid_request error.

If a Client receive an invalid_request error it **MUST NOT** make any further requests for the same auth_req_id.

If the Client is registered to use the Push Mode then it **MUST NOT** call the Token Endpoint with the CIBA Grant Type and the following error is returned.

unauthorized_client

The Client is not authorized as it is configured in Push Mode

> **Note:** When a Client receives a 4xx response code with a JSON payload then it **SHOULD** inspect the payload in order to determine the error rather than relying on the HTTP status code alone.

12. Push Error Payload

When Clients are configured to use the Push token delivery mode they can receive error payloads at their Client Notification Endpoint. These errors will be sent using the application/json media type with the following three parameters:

error_description

OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the

"error_description" parameter **MUST NOT** include characters outside the set %x20-21 / %x23-5B / %x5D-7E

error

REQUIRED. A single ASCII error code from one present in the list below.

auth_req_id

REQUIRED. The authentication request identifier.

Error codes applicable to the push error payload:

access_denied

The end-user denied the authorization request.

expired_token

The auth_req_id has expired. The Client will need to make a new Authentication Request. OpenID Providers are not required to send this error, but Clients **SHOULD** support receiving this error.

transaction_failed

The OpenID Provider encountered an unexpected condition that prevented it from successfully completing the transaction. This general case error code can be used to inform the Client that the CIBA transaction was unsuccessful for reasons other than those explicitly defined by access_denied and expired_token.

13. Authentication Error Response

An Authentication Error Response is returned directly from the Backchannel Authentication Endpoint in response to the Authentication Request sent by the Client. The applicable error codes are detailed below (some of which are repurposed from OAuth 2.0 Sections 4.1.2.1 and 5.2).

Authentication Error Responses are sent in the same format as Token Error Responses, i.e. the HTTP response body uses the application/json media type with the following parameters:

error

REQUIRED. A single ASCII error code from one present in the list below.

error_description

OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter **MUST NOT** include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

error_uri

OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the "error_uri" parameter **MUST** conform to the URI-reference syntax and thus **MUST NOT** include characters outside the set %x21 / %x23-5B / %x5D-7E.

List of authentication error codes associated to HTTP Errors.

HTTP 400 Bad Request

invalid_request

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, contains more than one of the hints, or is otherwise malformed.

invalid_scope

The requested scope is invalid, unknown, or malformed.

expired_login_hint_token

The login_hint_token provided in the authentication request is not valid because it has expired.

unknown_user_id

The OpenID Provider is not able to identify which end-user the Client wishes to be authenticated by means of the hint provided in the request (login_hint_token, id_token_hint or login_hint).

unauthorized_client

The Client is not authorized to use this authentication flow.

missing_user_code

User code is required but was missing from the request.

invalid_user_code

User code was invalid.

invalid_binding_message

The binding message is invalid or unacceptable for use in the context of the given request.

HTTP 401 Unauthorized

invalid_client

Client authentication failed (e.g., invalid client credentials, unknown client, no client authentication included, or unsupported authentication method).

HTTP 403 Forbidden

access_denied

The resource owner or OpenID Provider denied the request. Note that as the authentication error response is received prior to any user interaction, such an error would only be received if a resource owner or OpenID Provider had made a decision to deny a certain type of request or requests from a certain type of client. The mechanism for such a decision to be made is outside the scope of this specification.

The following is a non-normative example from an authentication error response:

```
HTTP/1.1 400 Bad Request

Content-Type: application/json



{

  "error": "unauthorized_client",

  "error_description":
```

```
    "The client 'client.example.org' is not allowed to use
CIBA."

}
```

14. Security Considerations

The login_hint_token **SHOULD** be digitally signed by the issuer. This ensures authenticity of the data and reduces the threat of an injection attack. The signature allows the OP to authenticate and authorize the sender of the hint and prevent collecting of user identifiers by rogue Clients.

The OP **SHOULD** ensure that the "backchannel_client_notification_endpoint" configured at registration time is in the administrative authority of the Client. Otherwise, the OP would post authentication results to the wrong Client. How this check is done is outside the scope of this specification.

An id_token_hint cannot be validated using standard JWT processing rules because the token is being used in a context (sent from the Client back to the OP) that is different than that for which it was originally issued (typically a short lived token issued by the OP intended to convey claims about the authentication of an end-user to the Client). An expired ID Token therefore could still be considered valid as an id_token_hint so an OP **SHOULD**, for some reasonable period, accept id_token_hints with an expiration time that has passed. The OP **SHOULD** ensure that it is the issuer of the token and that the Client presenting the id_token_hint is listed in the audience claim. The OP **SHOULD** verify the signature to ensure that the token was, in fact, issued by it and hasn't been modified since. Note that due to key rotation, however, the OP may not necessarily have access to the key used to sign the token, so the length of time an ID Token is considered a valid hint will be likely to be limited by the OP's key rotation interval and retention period. Given these restrictions, implementers may consider not verifying the signature at all and only accepting ID Tokens with pairwise subject identifiers as hints. The OP could then validate that the Client authenticated at the Backchannel Authentication Endpoint was issued the pairwise subject identifier (or shares a Sector Identifier with the Client who was issued the pairwise subject identifier).

This specification defines a method of token delivery, "Push Callback", which differs from standard OAuth 2.0 flows. Most OAuth 2.0 and OpenID Connect profiles require the Client to authenticate at the token endpoint in order to retrieve access tokens, ID Tokens and refresh tokens. However with the CIBA Push mode, tokens are delivered directly to the Client at its Client Notification Endpoint. Implementers **SHOULD** ensure that appropriate security controls are in place for this endpoint and its registration with the OP. CIBA requires that a hash of the access token, a hash of the refresh token and the auth_req_id itself are included in the ID Token when the Push mode is used. This allows the Client to verify that the values in the Push Callback have not been tampered with.

In some scenarios it may be appropriate for the RP to pass metadata to the OP about the context of the session it has with the user. For example the RP could send the geolocation of the consumption device and the OP could verify that against the geolocation of the

authentication device. This specification does not require such metadata to be sent, nor does it define the method by which such metadata would be conveyed.

15. Privacy Considerations

This flow requires the Client to obtain an identifier for the end-user. When this identifier is a static global identifier, such as a phone number or email address, there are clear privacy implications. However this flow does not require the use of such identifiers and in deployments with higher privacy requirements, alternative identifiers could be used, such as:

ID Token containing a pairwise identifier

CIBA supports the use of ID Tokens as an id_token_hint in the authentication request. If the OP has previously issued an ID Token to the Client that contains a pairwise identifier and no personally identifiable information, then a CIBA flow can be initialized without the RP asking the user for a static identifier. This ID Token may have been obtained by the Client via a standard front-channel redirect flow.

Single-use user identifier, transferred from the AD to the CD

The OP could generate a single-use identifier which could be transferred from the Authentication Device to the Consumption Device at the start of the flow. For example, the user could authenticate on the AD and request an identifier for a CIBA flow. This could be displayed as a QR code, the user could scan this QR code at the CD. The RP would then decode the identifier and embed it in a login_hint_token which it would use to initiate a CIBA flow. The OP would then gain consent from the end-user for the access requested by the RP.

Discovery Service

For some ecosystems the RP may be able to send the user to a discovery service and receive back an encrypted login_hint_token which it can use in the authentication request. The OP would decrypt the token, identify the user and continue the CIBA flow.

16. IANA Considerations

16.1. OAuth Authorization Server Metadata Registration

This specification requests registration of the following values in the IANA "OAuth Authorization Server Metadata" registry of [IANA.OAuth.Parameters] established by RFC8414.

- Metadata Name: backchannel_token_delivery_modes_supported
- Metadata Description: Supported CIBA authentication result delivery modes
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document
- Metadata Name: backchannel_authentication_endpoint
- Metadata Description: CIBA Backchannel Authentication Endpoint
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net

- Specification Document(s): Section 4 of this document
- Metadata Name: backchannel_authentication_request_signing_alg_values_supported
- Metadata Description: JSON array containing a list of the JWS signing algorithms supported for validation of signed CIBA authentication requests
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document
- Metadata Name: backchannel_user_code_parameter_supported
- Metadata Description: Indicates whether the OP supports use of the CIBA user_code parameter.
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

16.2. OAuth Dynamic Client Registration Metadata Registration

This specification requests registration of the following client metadata definitions in the IANA "OAuth Dynamic Client Registration Metadata" registry of [IANA.OAuth.Parameters] established by RFC7591:

- Client Metadata Name: backchannel_token_delivery_mode
- Client Metadata Description: CIBA authentication result delivery mode
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document
- Client Metadata Name: backchannel_client_notification_endpoint
- Client Metadata Description: Endpoint to which a notification will be sent after a CIBA end-user authentication event
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document
- Client Metadata Name: backchannel_authentication_request_signing_alg
- Client Metadata Description: The JWS algorithm that the client will use to sign CIBA authentication requests
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net

- Specification Document(s): Section 4 of this document
- Client Metadata Name: backchannel_user_code_parameter
- Client Metadata Description: Indicates whether the Client **MUST** support the CIBA user_code parameter.
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification Document(s): Section 4 of this document

16.3. OAuth Parameters Registration

This specification requests registration of the following value in the IANA "OAuth Parameters" registry of [IANA.OAuth.Parameters] established by RFC6749.

- Parameter name: auth_req_id
- Parameter usage location: token response
- Change Controller: OpenID Foundation MODRNA Working Group - openid-specs-mobile-profile@lists.openid.net
- Specification document(s): Section 7.3 of this document

16.4. JSON Web Token Claims

This specification makes no request of IANA with respect to the "JSON Web Token Claims" registry at [IANA.JWT] established by RFC7519. Public Claim Names where used for the auth_req_id and refresh token hash in the ID Token in Section 10.3.1 in order to avoid further congestion of the registry with application specific claims that are unlikely to be of general applicability. For a Signed Authentication Request, where the the authentication request parameters are encoded as claims of the request JWT, the context of use is sufficiently constrained so as to safely consider those to be Private Claim Names.

17. References

17.1. Normative References

[I-D.ietf-oauth-mtls] Campbell, B., Bradley, J., Sakimura, N. and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", Internet-Draft draft-ietf-oauth-mtls-17, August 2019.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and C. Mortimore, "OpenID Connect Core 1.0", November 2014.

RFC2119 Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

RFC6750 Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012.

RFC7519 Jones, M., Bradley, J. and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015.

17.2. Informative References

[IANA.JWT] IANA, "JSON Web Token (JWT)"

[IANA.OAuth.Parameters] IANA, "OAuth Parameters"

RFC6749 Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012.

RFC7231 Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014.

RFC7591 Richer, J., Jones, M., Bradley, J., Machulak, M. and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015.

RFC8414 Jones, M., Sakimura, N. and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018.

RFC8628 Denniss, W., Bradley, J., Jones, M. and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019.

Appendix A. Acknowledgements

The following have contributed to the development of this specification.

John Bradley, Ralph Bragg, Geoff Graham, Joseph Heenan, Bjorn Hjelm, Takahiko Kawasaki, Torsten Lodderstedt, James Manger, Charles Marais, Nat Sakimura, and Petteri Stenius.

Appendix B. Notices

provided that attribution be made to the OIDF as the source of the material, but that such attribution does not indicate an endorsement by the OIDF.

The technology described in this specification was made available from contributions from various sources, including members of the OpenID Foundation and others. Although the OpenID Foundation has taken steps to help ensure that the technology is available for distribution, it takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this specification or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any independent effort to identify any such rights. The OpenID Foundation and the contributors to this specification make no (and hereby expressly disclaim any) warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this specification, and the entire risk as to implementing this specification is assumed by the implementer. The OpenID Intellectual Property Rights policy requires contributors to offer a patent promise not to assert certain patent claims against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

Authors' Addresses

Gonzalo Fernandez Rodriguez

Telefonica I+D

EMail: gonzalo.fernandezrodriguez@telefonica.com

Florian Walter

Deutsche Telekom AG

EMail: F.Walter@telekom.de

Axel Nennker

Deutsche Telekom AG

EMail: axel.nennker@telekom.de

Dave Tonge

Moneyhub

EMail: dave.tonge@moneyhub.com

Brian Campbell

Ping Identity

EMail: bcampbell@pingidentity.com