F**inancial-grade** API **Security** **Profile** 1.0 - Part 2: Advanced

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.

FAPI **security** **profile** 1.0 - Part 2: Advanced **- draft 10 incorporating errata set 1 - 1**

Abstract

FAPI 1.0 security profile - part 2: Baseline is an OAuth profile that aims to provide specific implementation guidelines for security and interoperability. It provides highly secure options.

Warning

This document is not an OIDF International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

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 **participant**). 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 subject to patent rights. OIDF shall not be held responsible for identifying any or all such patent rights.

Notational Conventions

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.

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 [ISODIR2]. 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. They are all in lower case to rule out the dictionary meaning use of these words so that they can be translated easily. Following is a summary of the keywords.

Kind Keywords

Requirement shall, shall not

Recommendation should, should not

Permission may

Possibility can, cannot

FAPI 1.0 consists of the following parts:

FAPI Security Profile 1.0 - Part 1: Baseline

FAPI Security Profile 1.0 - Part 2: Advanced

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

Introduction

FAPI is a highly secured OAuth profile that aims to provide specific implementation guidelines for security and interoperability. The FAPI 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 **impersonating** a user through password sharing. This brittle, inefficient, and insecure practice creates security vulnerabilities which require 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.

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**

RF**C6749 - The OAuth 2**.0 **Authorization Framework**

RFC67**50** - The OAuth 2.0 Authorization Framework**: Bearer Token Usage**

RFC7636 - Proof Key for Code Exchange by OAuth Public Clients

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.

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 documents are **referred** **to** **in** the **text in such a way that some or all of their content constitutes requirements** of this document. For dated references, only the edition cited **applies**. For undated references, the latest edition of the referenced document (including any amendments) applies.

Part**1** F**API** **Security** **Profile** **1**.0 **-** **Part 1: Baseline**

RFC67**49** - The OAuth 2.0 Authorization Framework

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

OIDC - OpenID Connect Core 1.0 incorporating errata set 1

RFC8705 - OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

JARM - 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

DN – Distinguished Name

HTTP – Hyper Text Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

JAR – JWT-Secured Authorization Request

JARM – JWT Secured Authorization Response Mode

JOSE – Javascript Object Signing and Encryption

JSON – JavaScript Object Notation

JWE – JSON Web Encryption

JWK – JSON Web Key

JWKS – JSON Web Key Sets

JWS – JSON Web Signature

JWT – JSON Web Token

MTLS – Mutual Transport Layer Security

OIDF - OpenID Foundation

OIDF – OpenID Foundation

PAR – Pushed Authorization Requests

PII – Personally Identifiable Information

PKCE – Proof Key for Code Exchange

REST – Representational State Transfer

REST – Representational State Transfer

RP – Relying Party

TLS – Transport Layer Security

TLS – Transport Layer Security

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

5. Advanced security profile

5. Advanced security profile

5.1 Authorization response security

5.1. Introduction

5.1.**1** Introduction

The OIDF F**inancial-grade API (F**API**)** 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.

The OIDF 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

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.

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 **highly secured** APIs by defining the measures to mitigate:

attacks that leverage the weak binding of endpoints in RFC6749 **(e.g. malicious endpoint attacks, IdP mix-up attacks)**, 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 fow** 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. **2** 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

5.1.3 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 increased 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

5.2.2.0 Authorization server provisions

The authorization server shall support the provisions specified in clause 5.2.2.0 of FAPI Security Profile 1.0 - Part 1: Baseline, with the exception that Section 5.2.2.0-7 (enforcement of RFC7636) is not required.

In addition, the authorization server

shall require a JWS signed JWT request object passed by value with the request parameter or by reference with the request_uri parameter;

shall require

the response_type value code id_token, or

the response_type value code in conjunction with the response_mode value jwt;

(moved to 5.2.2.1);

(moved to 5.2.2.1);

shall only issue sender-constrained access tokens;

shall support **MTLS** as mechanism for constraining the legitimate senders of access tokens;

(withdrawn);

(moved to 5.2.2.1);

(moved to 5.2.2.1);

shall only use the parameters included in the signed request object passed via the request or request_uri parameter;

may support the pushed authorization request endpoint as described in PAR;

(withdrawn);

shall require the request object to contain an exp claim that has a lifetime of no longer than 60 minutes after the nbf claim;

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):

tls_client_auth or self_signed_tls_client_auth as specified in section 2 of **MTLS**, or

private_key_jwt as specified in section 9 of OIDC;

shall require the aud claim in the request object to be, or to be an array containing, the **OP'**s **Issuer Identifier** URL;

shall not support public clients;

shall require the request object to contain an nbf claim that is no longer than 60 minutes in the past; and

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.

shall require a JWS signed JWT request object passed by value with the request parameter or by reference with the request_uri parameter;

shall require

the response_type value code id_token, or

the response_type value code in conjunction with the response_mode value jwt;

(moved to 5.2.2.1);

(moved to 5.2.2.1);

shall only issue sender-constrained access tokens;

shall support **RFC8705** as mechanism for constraining the legitimate senders of access tokens;

(withdrawn);

(moved to 5.2.2.1);

(moved to 5.2.2.1);

shall only use the parameters included in the signed request object passed via the request or request_uri parameter;

may support the pushed authorization request endpoint as described in PAR;

(withdrawn);

shall require the request object to contain an exp claim that has a lifetime of no longer than 60 minutes after the nbf claim;

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**.0**-4):

tls_client_auth or self_signed_tls_client_auth as specified in section 2 of **RFC8705**, or

private_key_jwt as specified in section 9 of OIDC;

shall require the aud claim in the request object to be, or to be an array containing, the **authorization server'**s **issuer identifier** URL;

shall not support public clients;

shall require the request object to contain an nbf claim that is no longer than 60 minutes in the past; and

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

shall support OIDC;

shall support signed ID Tokens;

should support signed and encrypted ID Tokens;

shall return ID Token as a detached signature to the authorization response;

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

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

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 F**inancial-grade** API Security Profile 1.0 - Part 1: Baseline, except for RFC7636 support.

In addition, the confidential client

shall support **MTLS** as mechanism for sender-constrained access tokens;

shall include the request or request_uri parameter as defined in Section 6 of OIDC in the authentication request;

shall ensure the **Authorization Server** has authenticated the user to an appropriate **Level** of **Assurance** for the client's intended purpose;

(moved to 5.2.3.1);

5.2.2.1 ID Token as detached signature

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

shall support OIDC;

shall support signed ID Tokens;

should support signed and encrypted ID Tokens;

shall return ID Token as a detached signature to the authorization response;

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

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

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 FAPI Security Profile 1.0 - Part 1: Baseline, except for RFC7636 support.

In addition, the confidential client

shall support **RFC8705** as mechanism for sender-constrained access tokens;

shall include the request or request_uri parameter as defined in Section 6 of OIDC in the authentication request;

shall ensure the **authorization server** has authenticated the user to an appropriate **level** of **assurance** for the client’s intended purpose;

(moved to 5.2.3.1);

(withdrawn);	(withdrawn);
(withdrawn);	(withdrawn);
(moved 5.2.3.1);	(moved 5.2.3.1);
shall send all parameters inside the authorization request's signed request object;	shall send all parameters inside the authorization request's signed request object;
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;	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;
shall send the aud claim in the request object as the **OP'**s **Issuer Identifier** URL;	shall send the aud claim in the request object as the **authorization server'**s **issuer identifier** URL;
shall send an exp claim in the request object that has a lifetime of no longer than 60 minutes;	shall send an exp claim in the request object that has a lifetime of no longer than 60 minutes;
(moved to 5.2.3.1);	(moved to 5.2.3.1);
(moved to 5.2.3.1);	(moved to 5.2.3.1);
shall send a nbf claim in the request object;	shall send a nbf claim in the request object;
shall use RFC7636 with S256 as the code challenge method if using PAR; and	shall use RFC7636 with S256 as the code challenge method if using PAR; and
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.	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	5.2.3.1 ID Token as detached signature
In addition, if the response_type value code id_token is used, the client	In addition, if the response_type value code id_token is used, the client
shall include the value openid into the scope parameter in order to activate OIDC support;	shall include the value openid into the scope parameter in order to activate OIDC support;
shall require JWS signed ID Token be returned from endpoints;	shall require JWS signed ID Token be returned from endpoints;
shall verify that the authorization response was not tampered using ID Token as the detached signature;	shall verify that the authorization response was not tampered using ID Token as the detached signature;
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.	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.
shall support both signed and signed & encrypted ID Tokens.	shall support both signed and signed & encrypted ID Tokens.
5.2.3.2. JARM	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	In addition, if the response_type value code is used in conjunction with the response_mode value jwt, the client

shall verify the authorization responses as specified in JARM, Section 4.4.

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. 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.

6.1 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 Advanced access provisions

6.2.1. Protected resources provisions

The protected resources supporting this document

shall support the provisions specified in clause 6.2.1 F**inancial-grade** API Security Profile 1.0 - Part 1: Baseline; and

shall adhere to the requirements in **MTLS**.

6.2.1 Protected resources provisions

The protected resources supporting this document

shall support the provisions specified in clause 6.2.1 FAPI Security Profile 1.0 - Part 1: Baseline; and

shall adhere to the requirements in **RFC8705**.

6.2.2. Client provisions

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

6.2.2 Client provisions

The client supporting this document shall support the provisions specified in clause 6.2.2 of FAPI Security Profile 1.0 - Part 1: Baseline.

7. (Withdrawn)

7. (Withdrawn)

8. Security considerations

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

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 **RFC8705**.

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 **the attacker can then replay**.

When the FAPI Security Profile 1.0 client uses **RFC8705**, 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.

authenticate towards the token endpoint of the authorization server. **However, there is still the potential for a phished code be injected into a different flow involving an honest client.**

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 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 **RFC8705** 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 **authorization server** or misconfigured token endpoint.

For the "Access Token Injection with ID Token Replay" attack, the attacker tricks a client under his control to start a normal authorization flow to obtain an authorization

response with an ID Token. The ID Token is replayed along with a phished access token at the token endpoint (which is misconfigured in the client to point to an attacker-controlled URL). The attacker then gains access to resources of the honest resource owner through the client.

Misconfigured endpoints are mitigated by using metadata in the authorization server's published metadata document as defined in OIDD or RFC8414.

ID Token replay can be mitigated by requiring the at_hash in the token endpoint's ID Token response to verify the validity of the access token.

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

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 F**inancial-grade** API Security Profile 1.0 - Part 1: Baseline shall apply, with the following additional requirements:

For TLS versions below 1.3, only the **following 4** cipher suites **shall be permitted:**

TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

TLS_EC**DHE_RSA_WITH_AES_**1**28_GCM_SHA256**

TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

TLS_ECDHE_RSA_WITH_AES_25**6_GCM_SHA384**

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.

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

shall use PS256 or ES256 algorithms;

should not use algorithms that use RSASSA-PKCS1-v1_5 (e.g. RS256); and

shall not use none.

8.6.1. Encryption algorithm considerations

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 FAPI Security Profile 1.0 - Part 1: Baseline shall apply, with the following additional requirements:

Only the cipher suites **recommended in B**C**P**1**9**5 **shall be permitted.**

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.

8.6 Algorithm considerations

For JWS, both clients and authorization servers

shall use PS256 or ES256 algorithms;

should not use algorithms that use RSASSA-PKCS1-v1_5 (e.g. RS256); and

shall not use none.

8.6.1 Encryption algorithm considerations

For JWE, both clients and authorization servers

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

For JWE, both clients and authorization servers

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

requires that jwks_uri endpoints shall be served over TLS;

recommends that JOSE headers for x5u and jku should not be used; and

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:

find keys with a kid that matches the kid in the JOSE header;

if a single key is found, use that key;

This profile requires both **clients** and **authorization servers** to verify payloads with keys from the other party. The **authorization server** verifies request objects and private_key_jwt assertions. The **client** verifies ID Tokens and authorization response JWTs. For **authorization servers,** 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 **authorization server** jwks_uri can be found in RFC8414, while the definition of the **client** jwks_uri can be found in RFC7591.

In addition, this profile

requires that jwks_uri endpoints shall be served over TLS;

recommends that JOSE headers for x5u and jku should not be used; and

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 **RFC8705** 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 **RFC8705** 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:

find keys with a kid that matches the kid in the JOSE header;

if a single key is found, use that key;

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**/O**P**) A**S**/O**P** 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.

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 **authorization server**, **resource server** 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 **authorization server**/O**penID provider**) A**uthorization server**/O**penID provider** 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 **authorization server**) User misunderstands who the RP is due to a confusing representation of the RP at the **authorization server'**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**) A**S** 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)

(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 **authorization server** 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 **authorization server**) A**uthorization server** stores personal data. If **authorization server** 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 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

RFC**7636 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

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

ISODIR2 - ISO/IEC Directives, Part 2 - Principles and rules for the structure and drafting of ISO and IEC documents

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

RFC6750 The OAuth 2.0 Authorization Framework: Bearer Token Usage

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

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)

SoK: Single Sign-On Security – An Evaluation of OpenID Connect 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 FAPI Working Group - openid-specs-fapi@lists.openid.net

Reference: Section 5 of [[this specification]]

Appendix A. Examples

A.0 JWK for 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_znJkKXsuSIH735xnSmx74CC8TOe6G-V

0Wi_wVSJ9bHPphSki_kWUtjVGcnyjYuQVE0LRj3qrGPAX9bsVKSqs8T9AM41TB9oV5Sjz5YhggwICvvC

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

xlmFIrkniMHRzjyYQTHRU0gW3DRV9MrQQrmP71McvfLPeMBPPgsHgLo7KmUBDoUjsgnwgycEOWPm8MWJ

13dpTsVnoWNIFQqVNz1L5pRU3Uoknl0MGoE6v0M9lfgQgzxIX9gSB1VGp5zZRcsnZGU3MFpwBhOWwiCU

wqztoX0H5P0g7OWocspHrDn6YOgxHw"

}

A.1. Example request object

eyJraWQiOiJjbGllbnQtMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJhdWQiOiJodHRwczpcL1wv

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_znJkKXsuSIH735xnSmx74CC8TOe6G-V

0Wi_wVSJ9bHPphSki_kWUtjVGcnyjYuQVE0LRj3qrGPAX9bsVKSqs8T9AM41TB9oV5Sjz5YhggwICvvC

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

xlmFIrkniMHRzjyYQTHRU0gW3DRV9MrQQrmP71McvfLPeMBPPgsHgLo7KmUBDoUjsgnwgycEOWPm8MWJ

13dpTsVnoWNIFQqVNz1L5pRU3Uoknl0MGoE6v0M9lfgQgzxIX9gSB1VGp5zZRcsnZGU3MFpwBhOWwiCU

wqztoX0H5P0g7OWocspHrDn6YOgxHw"

}

A.1 Example request object

eyJraWQiOiJjbGllbnQtMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJhdWQiOiJodHRwczpcL1wv

ZmFwaS1hcy5leGFtcGxlLmNvbVwvIiwibmJmIjoxNTk0MTQwMDMwLCJzY29wZSI6Im9wZW5pZCBwYXlt

ZW50cyIsImlzcyI6IjUyNDgwNzU0MDUzIiwicmVzcG9uc2VfdHlwZSI6ImNvZGUgaWRfdG9rZW4iLCJy

ZWRpcmVjdF91cmkiOiJodHRwczpcL1wvZmFwaS1jbGllbnQuZXhhbXBsZS5vcmdcL2FhcGktYXMtY2Fs

bGJhY2siLCJzdGF0ZSI6IlZnU1VJRW5mbG5EeFRlMXZBdHI1NG8iLCJleHAiOjE1OTQxNDAzOTAsIm5v

bmNlIjoiN3hEQ0h2aXVQTVNYSklpZ2tIT2NEaSIsI
mNsaWVudF9pZCI6IjUyNDgwNzU0MDUzIn0.VSo5

VWN3lOiCry2KItU5RI62i9KG2KQlBdpsDT0DI0vSMK-q85aJZvsMiHBNBv1PQ9qAWmU3oJS-yi-Ks_lD

lP6lIMFrOL_Ym3VxJ_SM6lrc8JSZH_nNx6sqxPpeMQTF4SFPx30vHrlBVJaCGfnCMVC6Nbzwef0vOEpN

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.eyJzdWIiOiIxMDAxIiwiYXVk

IjoiNTI0ODA3NTQwNTMiLCJjX2hhc2giOiJRUjJ6dWNmWVpraUxyYktCS0RWcGdRIiwic19oYXNoIjoi

OXM2Q0JiT3hpS0U2NWQ5LVFyMFFJUSIsImF1dGhfdGltZSI6MTU5NDE0MDA5MCwiaXNzIjoiaHR0cHM6

ZmFwaS1hcy5leGFtcGxlLmNvbVwvIiwibmJmIjoxNTk0MTQwMDMwLCJzY29wZSI6Im9wZW5pZCBwYXlt

ZW50cyIsImlzcyI6IjUyNDgwNzU0MDUzIiwicmVzcG9uc2VfdHlwZSI6ImNvZGUgaWRfdG9rZW4iLCJy

ZWRpcmVjdF91cmkiOiJodHRwczpcL1wvZmFwaS1jbGllbnQuZXhhbXBsZS5vcmdcL2FhcGktYXMtY2Fs

bGJhY2siLCJzdGF0ZSI6IlZnU1VJRW5mbG5EeFRlMXZBdHI1NG8iLCJleHAiOjE1OTQxNDAzOTAsIm5v

bmNlIjoiN3hEQ0h2aXVQTVNYSklpZ2tIT2NEaSIsI
mNsaWVudF9pZCI6IjUyNDgwNzU0MDUzIn0.VSo5

VWN3lOiCry2KItU5RI62i9KG2KQlBdpsDT0DI0vSMK-q85aJZvsMiHBNBv1PQ9qAWmU3oJS-yi-Ks_lD

lP6lIMFrOL_Ym3VxJ_SM6lrc8JSZH_nNx6sqxPpeMQTF4SFPx30vHrlBVJaCGfnCMVC6Nbzwef0vOEpN

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.eyJzdWIiOiIxMDAxIiwiYXVk

IjoiNTI0ODA3NTQwNTMiLCJjX2hhc2giOiJRUjJ6dWNmWVpraUxyYktCS0RWcGdRIiwic19oYXNoIjoi

OXM2Q0JiT3hpS0U2NWQ5LVFyMFFJUSIsImF1dGhfdGltZSI6MTU5NDE0MDA5MCwiaXNzIjoiaHR0cHM6

6

XC9cL2ZhcGktYXMuZXhhbXBsZS5jb21cLyIsImV4cCI6MTU5NDE0MDM5MCwiaWF0IjoxNTk0MTQwMDkw

LCJub25jZSI6Ijd4RENIdml1UE1TWEpJaWdrSE9jRGkifQ.Z-LpQRuYoiTqEBfVfctn-e6bLwSMqi8wA

3TuARGW6GyD05gPF6TVlUwHgJnSUlhETrzhEUAKKiyGDxGspuBU0OAnB4qepgrEBizk980NjCEVXNkog

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

eyJraWQiOiJjbGllbnQtZW5jLTIwMjAtMDgtMjgiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NNIiwi

YWxnIjoiUlNBLU9BRVAtMjU2In0.LFvxFCzJ-1NRl48pXTUs8f2axm5MRe9Cv0dgV6sXTRKwkT3nC2SJ

QlutOol36VARLd3uaIoj4Z7LVV_MrdIYYvDci2WLlKSlI_NRgR3qJ25N3S6fCqNEYRgDDbNzSr15MDRc

WQR5Jdl3VP8g748cowD_2gaopaCzZWTa3r_J2VOEETfcBAIMX0NbtVA3hHW-rQ0aCC7UIbP0_oEB2YF0

u6qAXCXuC02nO6coMSpSHTDZwkqkmFiFEKERM_Gayz3lVddlgfcPR2k76bCUjWy934-rOrOBGcLyS1Ww

aTIqMUS3WEIsAwCDr1Jt4pAioryRLZfLmWNff4QZSBxWejRqpw.uRANzseIWYB9YeAW.sJGqF2ERkMEE

jm8h62tUA4UeZIBqvVRpkQqjTuae7-4ac-4sSth0A3zeERvlyC5GcP0W2tj7uxMi0I4gpN33OfAOR-tA

9E_47oCHXrOH-7cpLgVIxxWZFx43dhxUh5QHuBfli4nHErMVUsFq6CzQj8Z5SHvBD2Qx3suPEeCNo_M2

XC9cL2ZhcGktYXMuZXhhbXBsZS5jb21cLyIsImV4cCI6MTU5NDE0MDM5MCwiaWF0IjoxNTk0MTQwMDkw

LCJub25jZSI6Ijd4RENIdml1UE1TWEpJaWdrSE9jRGkifQ.Z-LpQRuYoiTqEBfVfctn-e6bLwSMqi8wA

3TuARGW6GyD05gPF6TVlUwHgJnSUlhETrzhEUAKKiyGDxGspuBU0OAnB4qepgrEBizk980NjCEVXNkog

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

eyJraWQiOiJjbGllbnQtZW5jLTIwMjAtMDgtMjgiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NNIiwi

YWxnIjoiUlNBLU9BRVAtMjU2In0.LFvxFCzJ-1NRl48pXTUs8f2axm5MRe9Cv0dgV6sXTRKwkT3nC2SJ

QlutOol36VARLd3uaIoj4Z7LVV_MrdIYYvDci2WLlKSlI_NRgR3qJ25N3S6fCqNEYRgDDbNzSr15MDRc

WQR5Jdl3VP8g748cowD_2gaopaCzZWTa3r_J2VOEETfcBAIMX0NbtVA3hHW-rQ0aCC7UIbP0_oEB2YF0

u6qAXCXuC02nO6coMSpSHTDZwkqkmFiFEKERM_Gayz3lVddlgfcPR2k76bCUjWy934-rOrOBGcLyS1Ww

aTIqMUS3WEIsAwCDr1Jt4pAioryRLZfLmWNff4QZSBxWejRqpw.uRANzseIWYB9YeAW.sJGqF2ERkMEE

jm8h62tUA4UeZIBqvVRpkQqjTuae7-4ac-4sSth0A3zeERvlyC5GcP0W2tj7uxMi0I4gpN33OfAOR-tA

9E_47oCHXrOH-7cpLgVIxxWZFx43dhxUh5QHuBfli4nHErMVUsFq6CzQj8Z5SHvBD2Qx3suPEeCNo_M2

woohCprwjOKhE-Q_VkWUJb-Elrq9HxJcBtadw0spolqgYYTIWvV4fcKmbtGANYLac29oKWd5-jyDAsSF

FZrSCNxv-BtJUiUVWUn5eVufjJYCx62Ju-MZ8vsPNTE-_I5em9RTBja6ylcivjzhW9Ncl6yKVfnB0XJN

cSSHQSFhc6Gvy7oYMBXx1C5G31OsiklkKQX2gsAZlxFQ_X25AXpMoV8-5xsUwdMdTaPxIIsccbrK2dfA

aP0rUruSV8zrlrbsN3ftjTJSka2XGG3kra76EPAlzSwxy6XdFVtEV31hirV3f9g04Gj_e-Q7J7HR62eY

3_09WyARShQL3DVXWOcK_8YrLr58JjNAbm0s5dAUq-zt9cMv8rl05t_dE59Gi6Hnl2YAiRdYG6B71FxJ

CE2Uqciy2jLe6mCDFDfqkog4G5R9FzNz5VzhVpmZVm3OJkug-UzayN7nwZ7jsmxQ2ucCM03xq-0MLdsk

H-cleahkFw5S-W40cn5hLrRXSqUoYfKmVSd9RltOZ6T0VrYpw2LaF2uUYEO9w9bMmg2zzfxft4WHsEbD

OlJVb5SE8mUjzBBZAcgaHYSv0Wii70lEJvLSdnVI1r9kuu9ae_j1Tu08RVyFGfgixYjI9z2L_sc8uOoO

HJ-Tq1iuncL3lCQJBuwBFoxyINlFgz4YV2AgreNsX8bDfE9XbRB9TnfvSd6rmes9lO0-3VQFlsC0C5dx

VXgp5o05E8nisPwuLmlGO5BTtBzCQ3tIH2SuTLTG-gohTEUVn4fACwIiyuXdPXcF4GxJNRNgNOH7xwxx

55qEM0xl2GuSseV59FiZR-WKMMs.kScy0JLB4XEClDAwTIVNA

which when decrypted using the following key:

{

"kty": "RSA",

"d": "OjDe8EkZXgvB-Gy5A4EdU8fBuAjdHLMyHKAtMaS_W_joEJHDvZRhIYbh1jAyHYoR3kFMXu

tCIYpRjDrsUEhjYuVKLm90CVtysoRjjkiXyupcEW3o--X_HBJhKm1Y-0I7LQ-cA7CotJpTVMR2fRTqP1

T4FsORAjg9l-fbdpVmeDiZBRbL2zCWmKWhtDpHyy7vbSCRghntihz_M5Hrchk7r8ito_K3dFrV9IZSF9

RoEY7kyK5bL36Kpgai44PYCzqOzqP2fteO_rZ9fn-uK59pI3ySo_PgSbJ55n14Nd9Z8m70zE9Z4aIeND

EFspZUhavngRwc7MuJ7f_hVGQ9RFbbkQ",

"e": "AQAB",

"use": "enc",

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

"n": "jVc92j0ntTV0V1nwZ3mpGaV2bME4d6AMS2SRrJBM0fLehaTEqDNzGu0warz2SC9bhcBOB5

_q3mYBFjmTwWzSbsk6RYETnAgViXg67PgH7Vkx2NCtwgQW3cNdnUZWRNYHsoevkx_Ta1X6Vi9ulebU_B

CKjrF-6CjVcGgEsO_S5DKcukGHdf81WlQOq3zGQg4h7MLArrbPSTHHORDsu_87qY9m2EhiYSOBSF5rHs

woohCprwjOKhE-Q_VkWUJb-Elrq9HxJcBtadw0spolqgYYTIWvV4fcKmbtGANYLac29oKWd5-jyDAsSF

FZrSCNxv-BtJUiUVWUn5eVufjJYCx62Ju-MZ8vsPNTE-_I5em9RTBja6ylcivjzhW9Ncl6yKVfnB0XJN

cSSHQSFhc6Gvy7oYMBXx1C5G31OsiklkKQX2gsAZlxFQ_X25AXpMoV8-5xsUwdMdTaPxIIsccbrK2dfA

aP0rUruSV8zrlrbsN3ftjTJSka2XGG3kra76EPAlzSwxy6XdFVtEV31hirV3f9g04Gj_e-Q7J7HR62eY

3_09WyARShQL3DVXWOcK_8YrLr58JjNAbm0s5dAUq-zt9cMv8rl05t_dE59Gi6Hnl2YAiRdYG6B71FxJ

CE2Uqciy2jLe6mCDFDfqkog4G5R9FzNz5VzhVpmZVm3OJkug-UzayN7nwZ7jsmxQ2ucCM03xq-0MLdsk

H-cleahkFw5S-W40cn5hLrRXSqUoYfKmVSd9RltOZ6T0VrYpw2LaF2uUYEO9w9bMmg2zzfxft4WHsEbD

OlJVb5SE8mUjzBBZAcgaHYSv0Wii70lEJvLSdnVI1r9kuu9ae_j1Tu08RVyFGfgixYjI9z2L_sc8uOoO

HJ-Tq1iuncL3lCQJBuwBFoxyINlFgz4YV2AgreNsX8bDfE9XbRB9TnfvSd6rmes9lO0-3VQFlsC0C5dx

VXgp5o05E8nisPwuLmlGO5BTtBzCQ3tIH2SuTLTG-gohTEUVn4fACwIiyuXdPXcF4GxJNRNgNOH7xwxx

55qEM0xl2GuSseV59FiZR-WKMMs.kScy0JLB4XEClDAwTIVNA

which when decrypted using the following key:

{

"kty": "RSA",

"d": "OjDe8EkZXgvB-Gy5A4EdU8fBuAjdHLMyHKAtMaS_W_joEJHDvZRhIYbh1jAyHYoR3kFMXu

tCIYpRjDrsUEhjYuVKLm90CVtysoRjjkiXyupcEW3o--X_HBJhKm1Y-0I7LQ-cA7CotJpTVMR2fRTqP1

T4FsORAjg9l-fbdpVmeDiZBRbL2zCWmKWhtDpHyy7vbSCRghntihz_M5Hrchk7r8ito_K3dFrV9IZSF9

RoEY7kyK5bL36Kpgai44PYCzqOzqP2fteO_rZ9fn-uK59pI3ySo_PgSbJ55n14Nd9Z8m70zE9Z4aIeND

EFspZUhavngRwc7MuJ7f_hVGQ9RFbbkQ",

"e": "AQAB",

"use": "enc",

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

"n": "jVc92j0ntTV0V1nwZ3mpGaV2bME4d6AMS2SRrJBM0fLehaTEqDNzGu0warz2SC9bhcBOB5

_q3mYBFjmTwWzSbsk6RYETnAgViXg67PgH7Vkx2NCtwgQW3cNdnUZWRNYHsoevkx_Ta1X6Vi9ulebU_B

CKjrF-6CjVcGgEsO_S5DKcukGHdf81WlQOq3zGQg4h7MLArrbPSTHHORDsu_87qY9m2EhiYSOBSF5rHs

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.eyJhdWQiOiI0NjkxODA2NDgw

MzkwNTEiLCJjb2RlIjoiendrR2FjOWp1TFg4RjhmcmFwRElTaTNLMkZ3bG40cXh3eWZOSUkzQ2p6MCIs

ImlzcyI6Imh0dHBzOlwvXC9mYXBpLWFzLmV4YW1wbGUuY29tXC8iLCJzdGF0ZSI6IlZnU1VJRW5mbG5E

eFRlMXZBdHI1NG8iLCJleHAiOjE1OTQxNDEwOTB9.k_3df0dIDX6watKxQkzAHOLgf4FBi_xIPN-n8aT

5hMX3gaBbeDqdUA5NR764L4ugdDgXyQm8dNcZrZldKIPfSfRcjBTtSx9PEdiffn_xUkwnS18YNAfEoq0

HjvkOQ59F21ImKn113kon00uC2dqBGByRrZcaUYOnvW2DdHCVA0VTW2je5nzbI02z9csLa8uGGGwjWRP

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

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.eyJhdWQiOiI0NjkxODA2NDgw

MzkwNTEiLCJjb2RlIjoiendrR2FjOWp1TFg4RjhmcmFwRElTaTNLMkZ3bG40cXh3eWZOSUkzQ2p6MCIs

ImlzcyI6Imh0dHBzOlwvXC9mYXBpLWFzLmV4YW1wbGUuY29tXC8iLCJzdGF0ZSI6IlZnU1VJRW5mbG5E

eFRlMXZBdHI1NG8iLCJleHAiOjE1OTQxNDEwOTB9.k_3df0dIDX6watKxQkzAHOLgf4FBi_xIPN-n8aT

5hMX3gaBbeDqdUA5NR764L4ugdDgXyQm8dNcZrZldKIPfSfRcjBTtSx9PEdiffn_xUkwnS18YNAfEoq0

HjvkOQ59F21ImKn113kon00uC2dqBGByRrZcaUYOnvW2DdHCVA0VTW2je5nzbI02z9csLa8uGGGwjWRP

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.eyJzdWIiOiI1MjQ4MDc1NDA1

MyIsImF1ZCI6Imh0dHBzOlwvXC9mYXBpLWFzLmV4YW1wbGUuY29tXC9hcGlcL3Rva2VuIiwiaXNzIjoi

NTI0ODA3NTQwNTMiLCJleHAiOjE1OTQxNDAxNTEsImlhdCI6MTU5NDE0MDA5MSwianRpIjoiNHZCY3RN

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

}

A.5 Example private_key_jwt client assertion

eyJraWQiOiJjbGllbnQtMjAyMC0wOC0yOCIsImFsZyI6IlBTMjU2In0.eyJzdWIiOiI1MjQ4MDc1NDA1

MyIsImF1ZCI6Imh0dHBzOlwvXC9mYXBpLWFzLmV4YW1wbGUuY29tXC9hcGlcL3Rva2VuIiwiaXNzIjoi

NTI0ODA3NTQwNTMiLCJleHAiOjE1OTQxNDAxNTEsImlhdCI6MTU5NDE0MDA5MSwianRpIjoiNHZCY3RN

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 Changes

2023-06-25

Applied changes needed to convert to pandoc

Changed the title to incorporate “errata”

#600 - Changed Financial-grade API to FAPI

#468 - Reference final versions of JAR, JARM, PAR

#409 - Rename [MTLS] as RFC8705

#405 - Use https for document references

#527,624 - Added security consideration for Access Token Injection with ID Token Replay

#613 - Remove empty subclauses 5.2.4 and 5.2.5

#612 - Fixed hanging paragraph in 5.1 and renumbered subclauses in 5.1.x

Copyright (c) 2021 The OpenID Foundation.

The OpenID Foundation (OIDF) grants to any Contributor, developer, implementer, or other interested party a non-exclusive, royalty-free, worldwide copyright license to reproduce, prepare derivative works from, distribute, perform and display, this Implementers Draft or Final Specification solely for the purposes of (i) developing specifications, and (ii) implementing Implementers Drafts and Final Specifications based on such documents, 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.

文字数: 38000
空白数: 5560 空白込み文字数: 43560
改行数: 744 改行込み文字数: 44304
単語数: 5788

#611 - 8.3.5 content moved to previously empty 8.3.4

Notices

Copyright (c) 2023 The OpenID Foundation.

The OpenID Foundation (OIDF) grants to any Contributor, developer, implementer, or other interested party a non-exclusive, royalty free, worldwide copyright license to reproduce, prepare derivative works from, distribute, perform and display, this Implementers Draft or Final Specification solely for the purposes of (i) developing specifications, and (ii) implementing Implementers Drafts and Final Specifications based on such documents, 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.

文字数: 39517
空白数: 6343 空白込み文字数: 45860
改行数: 625 改行込み文字数: 46485
単語数: 6129