MITRE

Enterprise Mission Tailored OAuth 2.1 Profile

Bedford, MA

Beth Abramowitz
Kelley Burgin
Ben Cresitello-Dittmar
Neil McNab
Roger Westman

December 2022

This page intentionally left blank.

Table of Contents

List of Figures

1 Introduction

This document profiles *The OAuth 2.1 Authorization Framework* [OAuth2-1] for use in the context of securing web-facing application programming interfaces (APIs), particularly Representational State Transfer (RESTful) APIs. OAuth 2.1 is an update to OAuth 2.0 [RFC6749] that incorporates best current practices, including addressing security issues. The OAuth 2.1 client, authorization server, and protected resource profiles defined in this document serve two purposes:

1. Define a mandatory baseline set of security controls, while maintaining reasonable ease of implementation and functionality.
2. Define objective requirements for use of features that provide stronger security properties but are not yet widely available in OAuth implementations.

Readers are expected to be familiar with [OAuth2-1]. All components MUST comply with all requirements in that specification; this profile document levies additional requirements for the enterprise environment.

Section 5 of this document provides detailed security rationale for the profiling decisions made.

1.1 Requirements Notation and Convention

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in [RFC2119].

All uses of JSON Web Signature (JWS) and JSON Web Encryption (JWE) data structures in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used.

1.2 Terminology

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Authorization Server", "Client", "Client Authentication", "Client Identifier", "Client Secret", "Grant Type", "Protected Resource", "Redirection URI", "Refresh Token", "Resource Owner", "Resource Server", "Response Type", and "Token Endpoint" defined by OAuth 2.1, the terms "Claim Name", "Claim Value", and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [RFC7519], and the terms defined by OpenID Connect Core 1.0 [OIDC-Core].

1.3 Conformance

This specification defines requirements for the following components:

- OAuth 2.1 clients.
- OAuth 2.1 authorization servers.
- OAuth 2.1 protected resources.

The requirements include details of interaction between these components:

- Client to authorization server.

- Client to protected resource.
- Protected resource to authorization server.

When a profile-compliant component is interacting with other profile-compliant components, in any valid combination, all components MUST implement the requirements as stated in this specification. All interaction with non-profile components is outside the scope of this specification.

A profile-compliant OAuth 2.1 client MUST support and utilize certain features as described in Section 2 of this specification.

A profile-compliant OAuth 2.1 authorization server MUST support and utilize certain features as described in Section 3 of this specification.

A profile-compliant OAuth 2.1 protected resource MUST support and utilize certain features as described in Section 4 of this specification.

1.4 Environment Overview

This profile is intended for use in enterprise environments, not consumer-facing environments. In enterprise environments, users do not "own" their data, the enterprise does. However, the user may have some level of responsibility for ensuring that unauthorized entities do not access data that the user has permission to access. In general, users need to be strongly authenticated in enterprise environments and not be able to act anonymously when accessing data.

The enterprise is assumed to have a deployed Public Key Infrastructure (PKI). The PKI issues non-person entity (NPE) certificates to clients, protected resources, and authorization servers. As discussed later, the PKI can be leveraged to provide greater assurance than is present in current typical non-enterprise OAuth deployments.

1.5 Use Cases

This profile is oriented around two primary use cases: user authorization delegation to a web application, and user authorization delegation to a native application.

This profile is not intended to describe user authentication to a web application / server. OpenID Connect (OIDC), which builds upon OAuth, is intended for that use case. OpenID Connect is profiled in a separate document.

This use case section is non-normative and is intended to provide examples to set the stage for the rest of the profile document. An example OAuth protocol flow with the additional requirements in this document is shown in Figure 1.

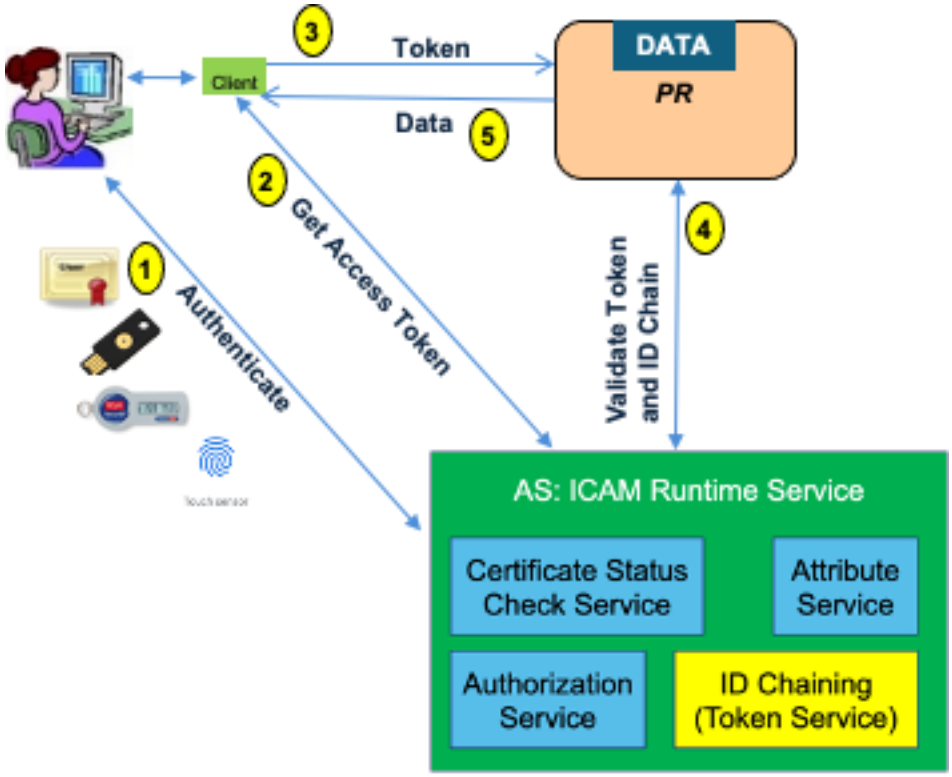

Figure 1: Example OAuth Protocol Flow

1.5.1 User Authorization Delegation to a Web Application

In this use case, a web application requires the ability to access a protected resource on behalf of a user, making use of some subset of the user's privileges. A web application is a capability provided by a web server running on a separate endpoint system than the user.

In a naïve approach, the web application could simply be given the ability to impersonate any user to the protected resource solely by authenticating itself and providing the user's identity. However, this approach does not prove to the protected resource that the user was involved in the transaction. Another naïve approach would be for the user to provide authentication credentials (e.g., username/password or PKI private key) to the web application. However, this approach provides the web application with full, unfettered ability to act as if it is the user with any resource.

OAuth enables a safer, limited approach for delegating user authorization to a web application to act on behalf of the user. With OAuth (when using the authorization code flow), the web application constructs an authorization request and redirects the user's web browser to an authorization server. The user authenticates to the authorization server (or the user's web browser makes use of an existing, authenticated session), and the authorization server redirects the user back to the web application with a one-time-use authorization code. The web application provides the one-time-use authorization code to the authorization server and receives an access token that it then uses to access the protected resource on the user's behalf. The access token is issued based on authentication to the authorization server of both the web application and the user. The access token can be limited to only allow a subset of the user's privileges, although the

details of how to represent authorization attributes within access tokens are out of scope of this profile. The access token can be limited to only be valid at a particular protected resource.

In OAuth terminology, the user is known as a “resource owner,” and the web application is known as a “client.” Web applications that have the ability to security store credentials with which to authenticate themselves to the authorization server are known in the OAuth specification as “confidential clients.”

1.5.2 User Authorization Delegation to a Native Application

In this use case, a native application requires the ability to access a protected resource on behalf of a user, making use of some subset of the user's privileges. For example, an email client may need the ability to access a user's mailbox on an email server.

In a naïve approach, the native application could simply be given the user's authentication credentials (e.g. username/password or private key). However, this approach requires the native application to store those credentials, and if stolen, provides an attacker with the full, unfettered ability to act as if he or she is the user with any resource. In addition, this approach limits the flexibility to introduce new authentication methods or perform adaptive authentication (e.g. based on dynamic risk decisions), as those methods would need to be supported by all native applications and all protected resources. For example, TLS client certificate authentication is widely used in some enterprise environments but requiring every app developer to implement client certificate authentication within each app is not feasible.

OAuth enables a safer, limited approach for delegating user authorization to a native application to act on behalf of the user. With OAuth, using the authorization code flow, the native application constructs an authorization request and redirects the user's web browser to an authorization server. The user authenticates to the authorization server through the web browser (or the user's web browser makes use of an existing, authenticated session). Any authentication method supported by both the web browser and the authorization server can be used, without specific support needed in the application (the web browser could also redirect to an authentication broker app on the user’s endpoint device for more advanced capabilities, for instance to provide device security posture assessments). The authorization server redirects the user back to the native application with a one-time-use authorization code. The native application provides the one-time-use authorization code to the authorization server and receives an access token that it then uses to access the protected resource on the user's behalf. The access token can be limited to only allow a subset of the user's access, and the access token can be limited to only be valid at a particular protected resource. For example, an access token issued to an email client could be valid only for accessing the email server, not other enterprise servers.

In OAuth terminology, the user is known as a “resource owner,” and the native application is known as a “client.” Unlike web applications, native applications typically do not have the ability to securely store credentials with which to authenticate the application itself to the authorization server. The access token is generally issued by the authorization server based on just the user's authentication, not the native application's authentication (the native application provides a client ID, but it typically can be easily captured and spoofed). Applications that do not possess secure credentials with which to authenticate themselves to the authorization server are known in the OAuth specification as “public clients.”

In some cases, rather than use a separate web browser, the native application embeds its own web browser. This approach eliminates the complexity of redirecting the authorization response

(containing the one-time-use authorization code) from the web browser back to the native application. However, this approach is generally not appropriate, as it directly exposes the native application to the user's credentials. It may also limit the types of authentication methods that can be used, as the native application may not have functionality for as wide a range of authentication methods as a dedicated web browser.

1.5.3 User Authorization Delegation to a Browser-Embedded Client

In this use case, a client application running entirely within the user's web browser requires the ability to access a protected resource on behalf of a user. These applications are typically written in JavaScript and are often referred to as "Single-Page Applications" (SPAs).

At this time, this use case is out of scope for this profile. The IETF Internet-Draft *OAuth 2.0 for Browser-Based Apps* [Parecki] provides potentially useful details and guidance for this use case, but an examination of its feasibility and security properties would first be necessary.

1.5.4 Identity Chaining

Token exchange [RFC8693] is currently out of scope for this profile but is addressed in the two documents 1) *Token and Identity Chaining Between Protected Resources in a Single ICAM Ecosystem Using OAuth Token Exchange* [Chaining-1] and 2) *Token and Identity Chaining Between Protected Resources in a Multiple ICAM Ecosystem Using OAuth Token Exchange* [Chaining-2].

This section provides an initial description of the token exchange use case.

A protected resource (PR1) may need to call a second protected resource (PR2) such as a second backend database in order to satisfy a query received from a client presenting Token 1. PR1 cannot simply replay Token 1 at PR2 since this OAuth 2.1 profile requires that the tokens be sender and audience constrained, so PR1 must request a new access token, Token 2, that is valid for PR1 to use at PR2 (in this usage, PR1 is acting as an OAuth client). If PR2 needs to access a third protected resource (PR3), then PR2 must request a new access token, Token 3, and so on. This process of exchanging Token 1 (which grants access to PR1) to obtain a new access token, Token 2 (which grants access to PR2) is called ***token chaining***. The identity chaining profiles additionally enable ***identity chaining*** by ensuring that the identities of the user, client, and protected resources are propagated in the exchanged tokens, so that each protected resource can, as necessary, use the set of identities to make appropriate access decisions.

If the protected resources are operated by different organizations, each of which relies on a different authorization server, then the situation is more complex. This case is considered in [Chaining-2] (see Figure 2).

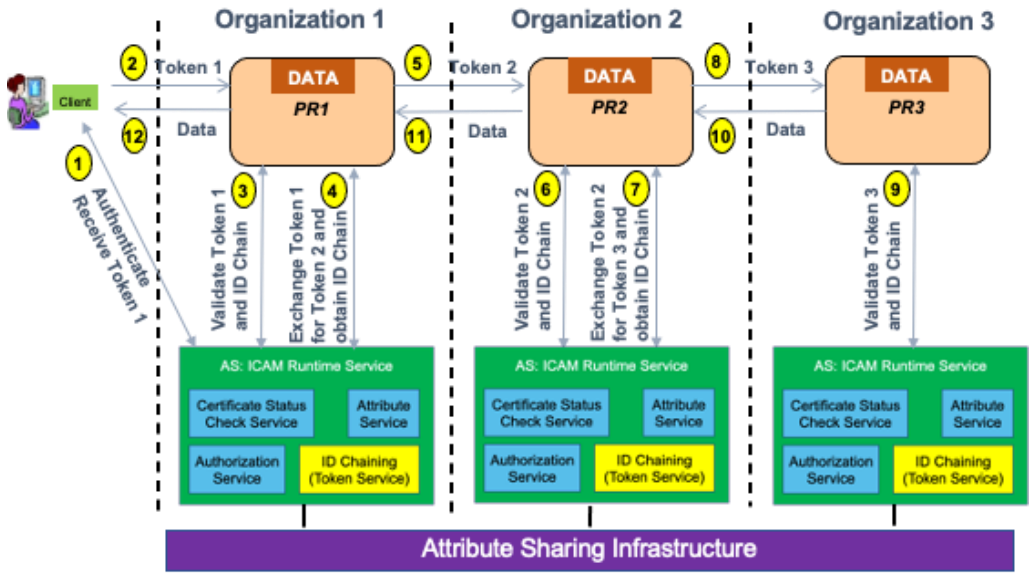

Figure 2: Identity Chaining in a Multiple ICAM Ecosystem Example

1.6 Global Requirements

This section contains requirements that apply to all components described in this profile.

Connections between a client and authorization server, between a client and resource server, and between a resource server and authorization server in the case of token introspection must use TLS 1.2 or above to prevent authorization codes and access tokens from being disclosed. Each TLS originator MUST have a capability to limit the certification authorities (CAs) trusted for verifying the TLS destination's PKI certificate [RFC6125]. The capability may be provided by the originator itself or by the originator’s underlying platform (e.g., operating system on which it is running).

2 Client Profiles

This section profiles the expected behavior of OAuth clients.

2.1 Client Types

This section, and overall profile, distinguishes between two types of clients: confidential clients and public clients.

2.1.1 Confidential Clients

The term “confidential client” applies to clients that act on behalf of a particular user and require delegation of that user’s authority to access protected resources. Confidential clients use their own credentials to authenticate themselves to the authorization server during the OAuth flow, so both the client and the user are authenticated by the authorization server as part of an authorization request.

Typically, confidential clients are web server backends, running on a separate system than the user, as described in Section 1.5.1.

Confidential clients MUST possess their own client certificate used for authentication to the authorization server.

2.1.2 Public Client

The term “public client” applies to clients that act on behalf of a particular user and require delegation of that user's authority to access the protected resource. Unlike confidential clients, public clients do not use their own credentials to authenticate themselves to the authorization server. Instead, only a client ID (which often can be easily captured) is used. Public clients are typically native applications running on the user's endpoint device, often leading to many identical instances of a piece of software operating in different environments and running simultaneously for different end users. With public clients, generally only the user, not the client, is authenticated by the authorization server as part of an authorization request. It may be possible to supplement user authentication with device security posture and/or application identity attestations, for example by using Android’s SafetyNet API [SafetyNet].

2.2 Connection to the Authorization Server

Clients MUST support the OAuth authorization code grant. Confidential clients MAY support the OAuth client credentials grant.

OAuth authorization servers provide both an authorization endpoint and a token endpoint. This section profiles connections to these two endpoints from clients. Both the authorization endpoint and token endpoint are used with the authorization code grant. Only the token endpoint is used with the client credentials grant.

OAuth confidential and public clients do not connect directly to the authorization endpoint. Rather, as described by the OAuth authorization code flow in [OAuth2-1], the client performs its request by redirecting the user's web browser to the authorization endpoint with appropriate parameters. The user authenticates to the authorization endpoint, and the user's web browser is redirected back to a URI hosted by the client, from which the client obtains an authorization code. The client then presents the authorization code to the authorization server's token endpoint to obtain an access token.

2.2.1 Discovery

Clients MAY use *OAuth 2.0 Authorization Server Metadata* [RFC8414] to retrieve configuration information from the authorization server, including supported options, endpoint URIs, and public keys.

Alternatively, clients MAY configure some or all of this information in an out-of-band manner.

2.2.2 Requests to the Authorization Endpoint

Clients MUST include their full redirect URI in authorization requests. Clients MUST use distinct redirect URIs for each authorization server as a means to identify the authorization server that a particular response came from. *OAuth 2.0 Authorization Server Issuer Identification* [RFC9207] defines the “iss” parameter that identifies the authorization server in authorization responses. Clients MUST check the “iss” parameter identifies the authorization server that the client made the authorization request to.

2.2.3 Requests to the Token Endpoint

Clients connect directly to the token endpoint to retrieve access tokens (and optionally refresh tokens). When the authorization code grant is used, clients provide the authorization code they receive as described in the previous section. When the client credentials grant is used, confidential clients do not provide an authorization code (as stated in [OAuth2-1], public clients cannot use the client credentials grant).

Confidential clients MUST support authentication to the authorization server's token endpoint using mutually authenticated TLS (mTLS). Public clients MAY support use of mutually authenticated TLS to the authorization server’s token endpoint. In the case of public clients, mutually authenticated TLS is not used to authenticate the client to the authorization server, it is used to enable cryptographically binding the access token issued by the authorization server to a private key held by the public client.

Mutually authenticated TLS connections by confidential clients MUST comply with [RFC8705]. The self-signed certificate option described in Section 2.2 "Self-Signed Certificate Mutual TLS OAuth Client Authentication Method" MUST NOT be used. Rather, the Section 2.1 "PKI Mutual TLS OAuth Client Authentication Method" MUST be used, where the subject distinguished name (DN) of the client's certificate is registered with the authorization server.

Mutually authenticated TLS connections by public clients, if used, MUST comply with Section 4 of [RFC8705].

2.2.4 Client Registration

Clients MUST be registered with the authorization server.

Client registration MUST be completed by out-of-band configuration. Clients MUST NOT register using dynamic registration [RFC7592].

2.2.4.1 Redirect URI

Android provides a feature called Android App Links [AppLinks], and Apple iOS provides a similar feature called Universal Links [UniversalLinks]. These features provide the ability to enforce a strong binding between a HTTPS URI and a specific mobile app installed on the Android or Apple device. Clients running on the user’s endpoint device SHOULD use [AppLinks], [UniversalLinks], or a similar capability enforced by the endpoint device platform to protect their redirect URIs as discussed in Section 8.4.1 of [OAuth2-1].

2.2.4.2 Client Keys

Clients using mTLS MUST register their TLS certificate's subject DN with the authorization server.

2.3 Connection to the Protected Resource

2.3.1 Requests to the Protected Resource

Clients MUST support TLS connections to the protected resource. Clients MUST provide access tokens to protected resources using the authorization header as described in [RFC6750].

3 Authorization Server Profile

This section details the expected behavior of OAuth Authorization Servers.

3.1 Connections with Clients

3.1.1 Grant Types

The authorization server MUST support the authorization code grant type as described in Section 2 and MAY support the client credentials grant type.

Authorization codes issued by the authorization server MUST contain a minimum of 128 bits of entropy. Authorization code lifetimes MUST be configurable on the authorization server. The authorization server MUST support lifetimes at least as short as 60 seconds.

If an authorization code is inadvertently accepted more than once, it MUST be noted in an audit log, any refresh token issued based on the authorization code MUST be revoked, and any access token issued based on the authorization code SHOULD be revoked (if it is possible to do so).

3.1.2 Client Authentication

The authorization server MUST enforce client authentication for confidential clients.

The authorization server MUST support TLS client certificate authentication of confidential clients as in [RFC8705] or JWT assertion [RFC7521]. The self-signed certificate option described in [RFC8705] Section 2.2 "Self-Signed Certificate Mutual TLS OAuth Client Authentication Method" MUST NOT be used. Rather, the [RFC8705] Section 2.1 "PKI Mutual TLS OAuth Client Authentication Method" MUST be used, where the subject distinguished name (DN) of the client's certificate is registered with the authorization server.

The authorization server MUST support mTLS connections from public clients as specified in [RFC8705]. In the case of public clients, mTLS is not used to authenticate the client to the authorization server, it is used to enable cryptographically binding the access token issued by the authorization server to a private key held by the public client.

3.1.3 User Authentication to the Authorization Server

The authorization server MUST support the following mechanism for users to authenticate themselves to the authorization server:

- TLS client certificate authentication

The authorization server SHOULD support the following mechanisms for users to authenticate themselves to the authorization server:

- RSA SecurID
- FIDO 2.0 / W3C Web Authentication
- Username and password
- Federated authentication to a user’s home organization using OpenID Connect (described below as identity brokering)

The authorization server MAY support other user authentication mechanisms. The authorization server MAY also support the ability to authenticate (and assess security properties of) the user’s endpoint device in addition to the user. Such support may be detailed further in a future profile.

The authorization server MUST provide the ability for an administrator to configure which user authentication mechanisms are acceptable.

To authenticate users from organizations outside the authorization server's organization, authorization servers MUST support identity brokering using OpenID Connect. With identity brokering, the authorization servers associated with each protected resource act as an OpenID Connect Relying Party (RP), delegating authentication to an OpenID Connect Identity Provider (IdP) operated by the user's home organization. The user authenticates to their home Identity Provider, and that IdP asserts to the authorization server that the authentication successfully occurred. If needed, the protected resource's authorization server can obtain attributes about the user from the user's IdP or through some other mechanism. If implemented, identity brokering MUST be performed in accordance with the Enterprise OpenID Connect Profile.

In non-enterprise environments, it is typically desired that the authorization server presents the user with the client's authorization request and require the user to explicitly approve the request. However, in this profile, the authorization server MUST provide the ability to disable such functionality. This profile is intended for enterprise environments where individual users do not "own" data. Additionally, this profile requires clients to be approved by the enterprise as part of the client registration process, which provides protection from malicious clients.

If the end user is prompted with an interactive approval page, the authorization server MUST indicate to the user:

- A human readable name of the client
- What kind of access the client is requesting (including scope, target resource, etc.)

3.1.4 Discovery

The authorization server MUST provide an OAuth authorization server metadata endpoint as specified by [RFC8414]. The endpoint MAY be shared with an OpenID Connect discovery endpoint. The endpoint's response MUST contain at least the following fields and MAY contain additional fields:

issuer	The fully qualified issuer URL of the server
authorization_endpoint	The fully qualified URL of the server's authorization endpoint
token_endpoint	The fully qualified URL of the server's token endpoint
jwks_uri	The fully qualified URI of the server's public key in JWK Set format
introspection_endpoint	The fully qualified URL of the server's introspection endpoint defined by OAuth Token Introspection
revocation_endpoint	The fully qualified URL of the server's revocation endpoint defined by [RFC7009] (only included if a revocation endpoint exists)

Note that if the authorization server is also an OpenID Connect Provider, its discovery endpoint must additionally meet the requirements listed in the Enterprise OpenID Connect Profile.

The following non-normative example shows the JSON document found at an authorization server metadata endpoint for an authorization server:

```
{
```

```
  "token_endpoint": "https://as.example.com/token",
  "token_endpoint_auth_methods_supported": [
    "tls_client_auth"[1]
  ],
  "jwks_uri": "https://as.example.com/jwk",
  "authorization_endpoint": "https://as.example.com/authorize",
  "introspection_endpoint": "https://as.example.com/introspect",
  "service_documentation": "https://as.example.com/about",
  "response_types_supported": [
    "code"
  ],
  "revocation_endpoint": "https://as.example.com/revoke",
  "grant_types_supported": [
    "authorization_code",
    "client_credentials",
  ],
  "scopes_supported": [
    "profile", "openid", "email", "address", "phone",
"offline_access"
  ],
  "op_tos_uri": "https://as.example.com/about",
  "issuer": "https://as.example.com/",
  "op_policy_uri": "https://as.example.com/about"
}
```

The authorization server MUST provide its public key (whose associated private key is used by the authorization server to sign tokens) in JWK Set format. The key MUST contain the following fields:

kid	The key ID of the key pair used to sign this token
kty	The key type
alg	The default algorithm used for this key

The authorization server MUST provide an RS256 key with a modulus of at least 2048 bits. The authorization server MAY provide additional keys using the following algorithms: RS384, RS512, ES256, ES384, ES512, PS256, PS384, PS512.

The following is a non-normative example of a 2048-bit RSA public key:

```
{
"keys": [
    {
      "alg": "RS256",
      "e": "AQAB",
      "n": "o80vbR0ZfMhjZWfqwPUGNkcIeUcweFyzB2S2T-
hje83IOVct8gVg9FxvHPK1R
```

[1] Note: The “tls_client_auth” authentication method name has not yet been finalized by the IETF.

```
eEW3-p7-A8GNcLAuFP_8jPhiL6LyJC3F10aV9KPQFF-
w6Eq6VtpEgYSfzvFegNiPtpMWd7C43
EDwjQ-GrXMVCLrBYxZC-
P1ShyxVBOzeR_5MTC0JGiDTecr_2YT6o_3aE2SIJu4iNPgGh9Mnyx
dBo0Uf0TmrqEIabquXA1-
V8iUihwfI8qjf3EujkYi7gXXelIo4_gipQYNjr4DBNlE0__RI0kD
U-27mb6esswnP2WgHZQPsk779fTcNDBIcYgyLujlcUATEqfCaPDNp00J6AbY6w",
      "kty": "RSA",
      "kid": "rsa1"
    }
] }
```

3.1.5 Redirect URIs

The authorization server MUST ensure that each redirect URI is one of the following:

- An HTTPS URI referring to a website with Transport Layer Security (TLS) protection or an app installed on the user's endpoint using [AppLinks], [UniversalLinks], or similar capability (this method is the preferred choice)
- Hosted on the user's endpoint without involving remote network connectivity (e.g., http://localhost/), however an HTTPS URI protected using [AppLinks], [UniversalLinks], or similar capability is preferred when possible
- Hosted on a client-specific non-remote-protocol URI scheme (e.g., myapp://), however an HTTPS URI protected using [AppLinks], [UniversalLinks], or similar capability is preferred when possible

3.2 Token Issuance Policy

The authorization server MUST be capable of enforcing an authorization policy that must be met in order for tokens to be issued. This policy MUST be customizable by the administrator. This profile does not enforce specific requirements upon capabilities of the authorization policy, but it is RECOMMENDED that at least the following attributes be considered:

- Attributes associated with the user's account, such as:
  - Personnel type (e.g. employee vs. contractor)
  - Citizenship
- The user's method(s) of authenticating to the authorization server
- The protected resource being accessed
- Security posture and other properties of the user's endpoint device
- IP address from which the user's endpoint device is connecting

3.3 JWT Access Tokens

To facilitate interoperability with protected resources, the authorization server MUST issue JWT access tokens compliant with [RFC9068]. The information carried in the JWT is intended to allow a protected resource to verify the authenticity and parse the contents of the token without additional network calls. If the protected resource is not capable of performing these operations, it can make use of token introspection [RFC7662] to request information about the token's authenticity and contents.

The authorization server MUST include the following claims in issued tokens:

iss	The issuer URL of the server that issued the token.
exp	The expiration time (integer number of seconds since from 1970-01-01T00:00:00Z UTC), after which the token MUST be considered invalid.
aud	The audience of the token. An array containing the identifier(s) of protected resource(s) for which the token is valid, if this information is known. The aud claim may contain multiple values if the token is valid for multiple protected resources.
sub	The identifier of the end-user that authorized this client, or in the case of the client credentials grant, the client id of a client acting on its own behalf.
client_id	The client id of the client to whom this token was issued.
iat	Issue timestamp
jti	A unique JWT Token ID value with at least 128 bits of entropy. This value MUST NOT be re-used in another token.
cnf	Capability required for requests from confidential clients, optional for requests from public clients. Specified by Section 3 of [RFC8705] (and by Section 4 for public clients). Hash of the client's PKI certificate that was presented using TLS mutual authentication between the client and authorization server. This field binds the access token to the client's certificate, enabling the protected resource to ensure that only the authorized client can present the access token (over a mutually authenticated TLS connection).

The authorization server SHOULD be capable of including additional fields in issued tokens, including the following:

amr	The user's authentication method to the AS when the user authorized issuance of this access token.
auth_time	Timestamp of when the user authenticated to the AS in order to authorize issuance of this access token (often not the same as iat, as the token may be issued based on a prior user authentication).

The access tokens MUST be signed with JWS. The authorization server MUST support the RS256 signature method for tokens. It MAY support the following additional asymmetric signing methods defined in the IANA JSON Web Signatures and Encryption Algorithms registry: RS384, RS512, ES256, ES384, ES512, PS256, PS384, PS512. The JWS header MUST contain the following field:

kid	The key ID of the key pair used to sign this token

The authorization server MAY encrypt access tokens using JWE. Encrypted access tokens MUST be encrypted using a public key designated for encryption (i.e., not used for digital signatures) of the protected resource identified in the "aud" claim.

3.4 Refresh Tokens

Mandates on the specific format of the refresh token are out of scope of this profile, as the refresh token is for the internal use of the authorization server, which both generates and consumes the token.

The authorization server MUST sign refresh tokens using JWS and MAY encrypt refresh tokens using JWE. If refresh tokens are encrypted using JWE, they MUST be encrypted either using the authorization server's public key or symmetrically encrypted using a secret key held by the authorization server.

3.5 Introspection

The authorization server MUST provide a token introspection endpoint. Token introspection [RFC7662] allows a protected resource to query the authorization server for metadata about a token. The server responds to an introspection request with a JSON object containing the following fields as defined in the token introspection specification:

active	Boolean value indicating whether or not this token is currently active at this authorization server. Tokens that have been revoked, have expired, or were not issued by this authorization server are considered non-active.
scope	Space-separated list of OAuth 2.0 scope values represented as a single string.
exp	Timestamp of when this token expires (integer number of seconds since from 1970-01- 01T00:00:00Z UTC)
sub	An opaque string that uniquely identifies the user who authorized this token at this authorization server (if applicable).
client_id	An opaque string that uniquely identifies the OAuth 2.0 client that requested this token

The server MAY include additional fields in its token introspection response.

The authorization server MUST require mTLS authentication for the introspection endpoint.

A protected resource MAY cache the response from the introspection endpoint for a period of time no greater than half the lifetime of the token. A protected resource MUST NOT accept a token that is not active according to the response from the introspection endpoint.

3.5 Token Lifetimes

This profile provides lifetimes for different types of tokens issued to different types of clients. Specific applications MAY issue tokens with different lifetimes. Any active token MAY be revoked at any time.

The authorization server MUST either limit access tokens to one hour, or (preferably) provide a capability to configure the access token lifetime (potentially on a per-protected resource and/or per-client basis) to a value of one hour or less. The authorization server SHOULD provide a capability to similarly configure refresh token lifetimes.

3.6 Scopes

Scopes define individual pieces of authority that can be requested by clients, granted by users, and enforced by protected resources. Specific scope values will be highly dependent on the specific types of resources being protected. OpenID Connect, for example, defines scope values to enable access to different attributes of user profiles.

Authorization servers MUST define and document default scope values that will be used if an authorization request does not specify a requested set of scopes.

Authorization servers MAY assert different scopes and authorization claims in the access token depending on the method used to authenticate the user.

Authorization servers MAY allow a refresh token issued for multiple scopes to be used to obtain an access token for just a subset of those scopes.

To facilitate general use across a wide variety of protected resources, authorization servers MUST allow for the definition of arbitrary scope values during client registration.

3.7 Viewing and Revoking Client Accesses and Tokens

The authorization server MUST provide an interface for end users to view a list of clients that have been granted access to resources on the user's behalf, and for end users to revoke this access.

Revocation MUST revoke any currently valid refresh tokens issued to the client to access resources on the user's behalf, SHOULD revoke currently valid access tokens, and MUST prevent the client from obtaining new tokens without the authorization server receiving a new authorization request via the user.

Note that revocation of access tokens may not have an immediate impact, as protected resources may not always check the revocation status of access tokens. However, this profile limits access tokens to a lifetime of 60 minutes, and revocation of the corresponding refresh token will prevent the client from obtaining a new access token upon the access token's expiration. The Continuous Access Evaluation Protocol (CAEP) under development in the OpenID Foundation’s Shared Signals and Events WG [CAEP] may, when complete, provide a mechanism for informing protected resources when access tokens are revoked.

The authorization server SHOULD provide an interface compliant with [RFC7009] for clients to request token revocation.

The authorization server MUST automatically revoke refresh tokens and SHOULD revoke access tokens under the following conditions:

1. User's account has been locked or deleted.
2. User's account credentials under which the tokens were issued have been reported lost or compromised (e.g. password, private key, hardware token, etc.).

3.8 Audit

The authorization server MUST record at least the following activities in an audit log:

1. Issuance of refresh tokens and access tokens to clients.
2. Attempted or successful use of an authorization code more than once.
3. Failed authentication attempts.

4 Protected Resource Profile

Protected resources grant access to clients if they present a valid access token with appropriate authorization claims (e.g. the token's scope claim and potentially other claims conveying detailed authorization information). Access tokens are not required to contain scopes or other claims conveying detailed authorization information. If they do not, the access token asserts the identity of the user (the token's sub claim) and the client (the token's client_id claim), and the protected

resource can make use of applicable enterprise authorization services to determine the allowed access.

Protected resources trust the authorization server to authenticate the end user appropriately for the importance, risk, and value level of the protected resource and requested scopes. The authorization server MAY assert different scopes and authorization claims in the access token depending on the method used to authenticate the user.

This section describes the expected behavior of OAuth protected resources (also known as resource servers). The connections with both clients and authorization servers are detailed below.

4.1 Connections from Clients

A protected resource MUST be capable of receiving access tokens passed in the authorization header as described in [RFC6750].

Protected resources MUST define and document which scopes are required for access to the resource.

Protected resources MUST verify access tokens by locally inspecting the JWT, using token introspection [RFC7662], or a combination of the two.

The protected resource MUST check the audience claim “aud” to ensure that it specifies the protected resource's identifier. The protected resource's identifier is the full subject distinguished name (DN) in the protected resource's certificate. The protected resource MUST ensure that the rights associated with the token are sufficient to grant access to the resource. The protected resource SHOULD enforce the authorization policy that is appropriate for the resource and not depend solely on OAuth.

Protected resources MUST support mutual TLS client certificate bound access tokens as specified in [RFC8705] Section 3.

If the protected resource uses token introspection, the protected resource MAY cache the response from the introspection endpoint for a period of time no greater than half the lifetime of the token, and a protected resource MUST NOT accept a token that is not active according to the response from the introspection endpoint.

4.2 Connections to Authorization Servers

Protected resources MAY use [RFC8414] to retrieve configuration information from the authorization server, including supported options, endpoint URIs, and public keys.

Alternatively, protected resources MAY configure some or all of this information in an out-of-band manner.

Protected resources MAY use token introspection [RFC7662] to connect to the authorization server to retrieve information about an access token presented by a client. If the protected resource uses token introspection, it MUST connect to the authorization server using mTLS.

5 Security Rationale for Profile Requirements

This section is intended to provide rationale behind this profile's requirements to help the reader understand why certain decisions were made.

This profile requires that clients be registered with authorization servers in an out-of-band manner, rather than allowing dynamic registration of clients. Clients must have some level of trust placed in them, as they are given the capability to access resources on behalf of the user. Phishing attacks have been demonstrated in environments that allow open registration of OAuth clients. For example, in a past incident, an attacker registered a fake "Google Docs" application with Google, and tricked users into granting the application access to their Google-hosted resources [Reddit]. Additionally, unlike in typical consumer-facing environments, this profile (since it is for enterprise use) does not require users to explicitly consent to granting clients access to their resources, making it even more critical that clients be trusted.

This profile requires use of TLS 1.2 or above for all OAuth interactions, as [OAuth2-1] does not explicitly require that all interactions be protected with TLS. For example, the initial interaction between the user's web browser and an OAuth client could occur over plaintext HTTP, and Fett et al. (Section 3.2 of [Fett]) describe how this property could be leveraged to carry out an authorization server mix-up attack.

This profile provides some degree of resilience in case server certificate validation is not sufficient. For example, an attacker may thwart server certificate validation by illegitimately obtaining a valid certificate from a trusted Certification Authority (CA) [Birge-Lee], somehow injecting new trusted Certificate Authority (CA) certificates into endpoints [Goodin], or exploiting unforeseen vulnerabilities in certificate validation routines. Resilience is provided by requiring that clients and protected resources have the capability of limiting the trusted CAs for connections to the authorization server. Additionally, mutually authenticated TLS connections are required by this profile for many network connections. In a mutually authenticated TLS connection, an attacker could potentially still impersonate the TLS server to the TLS client as described above, but would likely be unable to impersonate the TLS client to the TLS server.

This profile describes use of Mutual TLS Client Certificate Bound Access Tokens as specified by Section 3 of [RFC8705], mandating its support on authorization servers, confidential clients, and protected resources. This approach cryptographically binds the access token to the client that obtained it, requiring the client to authenticate to protected resources using mutually authenticated TLS in order for the protected resource to accept the access token. This approach prevents stolen access tokens (e.g. from the client's storage or from an insufficiently protected network connection) from being used without access to the client's private key. This approach (along with the token's "aud" field) also prevents a protected resource from replaying an access token that a client presented to it into another protected resource.

This profile requires that confidential clients authenticate to authorization servers using mTLS with a client certificate as described in [RFC8705]. In enterprise environments envisioned by this profile, confidential clients (typically front-end web servers) already possess and use non-person-entity (NPE) PKI certificates. These NPE PKI certificates and the associated private keys are ideal to use to authenticate clients to the authorization server rather than using a shared secret. mTLS also provides resilience against man-in-the-middle attacks, as even if an attacker can impersonate the server to the client, an attacker would additionally have to impersonate the client to the server (rather than just pass through an intercepted client_secret value).

Another asymmetric authentication method called "private_key_jwt" is defined by the OpenID Connect Core specification for authentication of the OAuth client to the authorization server. This profile does not allow its use. private_key_jwt has the advantage over client_secret that the private key is not exposed over the network to an attacker. However, it is not as secure as TLS

mutual authentication. With private_key_jwt, the client signs an assertion using its private key and attaches the assertion to its request. The assertion is not tied to the content of the client's request, so the client's request is not resilient against man-in-the-middle attacks if the attacker is able to impersonate the server to the client. The assertion could potentially be replayed if the authorization server does not store previously seen "jti" values until the assertion's expiration (a nonce is placed in the assertion to prevent replay). Additionally, private_key_jwt uses JSON Web Keys (JWKs) rather than X.509 certificates [RFC5280], so this may require the client to generate and manage another key pair, including ensuring that the authorization server has the client's public key.

OpenID Foundation's Financial-grade API Part 2 [FAPI2] provides a mechanism to use an OpenID Connect ID token to bind each received access token to a client authorization request. A future version of this profile may adopt that mechanism. If this threat is a concern, it can be addressed by having the client request and verify an ID token in accordance with the Enterprise OpenID Connect Profile.

6 Security Considerations

All transactions MUST be protected in transit by TLS as described in BCP195.

Security considerations are addressed throughout this profile and by [OAuth2-1].

7 Normative Reference

[AppLinks] Google. "Handling Android App Links", https://developer.android.com/training/app-links

[OAuth2-1] D. Hardt, A. Parecki, and T. Lodderstedt. "The OAuth 2.1 Authorization Framework", March 2021, Work-in-Progress, https://tools.ietf.org/html/draft-ietf-oauth-v2-1-07

[OIDC-Core] OpenID Foundation. "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, https://openid.net/specs/openid-connect-core-1_0.html

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119

[RFC5280] D. Cooper, et al. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, http://www.rfc-editor.org/info/rfc5280

[RFC6125] P. Saint-Andre and J. Hodges, "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)", RFC 6125, DOI 10.17487/RFC6125, March 2011, http://www.rfc-editor.org/info/rfc6125

[RFC6749] D. Hardt, Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, http://www.rfc-editor.org/info/rfc6749

[RFC6750] M. Jones. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012, http://www.rfc-editor.org/info/rfc6750

[RFC7009] T. Lodderstedt, Ed., S. Dronia, and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, http://www.rfc-editor.org/info/rfc7009

[RFC7519] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT)”, RFC 7519, DOI 10.17487/RFC7519, May 2015, http://www.rfc-editor.org/info/rfc7519

[RFC7521] B. Campbell, C. Mortimore, M. Jones, and Y. Goland, “Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants”, RFC7521, May 2015, http://www.rfc-editor.org/info/rfc7521

[RFC7592] M. Jones, J. Bradley, and M. Machulak, “OAuth 2.0 Dynamic Client Registration Management Protocol”, RFC7592, July 2015, http://www.rfc-editor.org/info/rfc7592

[RFC7662] Richer, J., Ed., “OAuth 2.0 Token Introspection”, RFC 7662, October 2015, http://www.rfc-editor.org/info/rfc7662

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, “OAuth 2.0 Authorization Server Metadata”, RFC 8414, DOI 10.17487/RFC8414, June 2018, http://www.rfc-editor.org/info/rfc8414

[RFC8705] Campbell, B., Bradley,J., Sakimura, N., and T. Lodderstedt, “OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens”, February 2020, https://tools.ietf.org/html/rfc8705

[RFC9068] V. Bertocci, “JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens.”, October 2021, https://tools.ietf.org/html/rfc9068

[RFC9207] K. Meyer zu Selhausen and D. Fett. “OAuth 2.0 Authorization Server Issuer Identifier”, March 2022, https://tools.ietf.org/html/rfc9207

[Parecki] Parecki, A., and D. Waite, “OAuth 2.0 for Browser-Based Apps”, May 2021, https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps-12

[UniversalLinks] Apple, “Universal Links for Developers”, https://developer.apple.com/ios/universal-links/

8 Informative Reference

[Birge-Lee] H. Birge-Lee, et al. "Bamboozling Certificate Authorities with BGP." USENIX Security Symposium 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/birge-lee

[CAEP] T. Cappalli and A. Tulshibagwale, OpenID Continuous Access Evaluation Profile 1.0 - draft 02, https://openid.net/specs/openid-caep-specification-1_0.html

[Fett] D. Fett, et al. "A Comprehensive Formal Security Analysis of OAuth 2.0." ACM CCS 2016, https://arxiv.org/pdf/1601.01229.pdf

[Goodin] D. Goodin, ArsTechnica. "Sennheiser discloses monumental blunder that cripples HTTPS on PCs and Macs", https://arstechnica.com/information-technology/2018/11/sennheiser-discloses-monumental-blunder-that-cripples-https-on-pcs-and-macs/

[Chaining-1] B. Abramowitz, et al., “Token and Identity Chaining Between Protected Resources in a Single ICAM Ecosystem Using OAuth Token Exchange”, MITRE PR 21-1421, https://www.mitre.org/news-insights/publication/token-and-identity-chaining-between-protected-resources-single-icam

[Chaining-2] B. Abramowitz, et al., “Token and Identity Chaining Between Protected Resources in a Multiple ICAM Ecosystem Using OAuth Token Exchange”,

MITRE PR 21-1421, https://www.mitre.org/news-insights/publication/token-and-identity-chaining-between-protected-resources-multiple-icam

[FAPI2] N. Sakimura, et al., “Financial-grade API – Part 2: Read and Write API SecurityProfile”, October 2018, https://openid.net/specs/openid-financial-api-part-2.html>

[Reddit] Reddit. "New Google Docs phishing scam, almost undetectable." https://www.reddit.com/r/google/comments/692cr4/new_google_docs_phishing_scam_almost_undetectable/

[RFC8693] M. Jones, et al. “OAuth 2.0 Token Exchange.” January 2020, https://tools.ietf.org/html/rfc8693

[SafetyNet] “SafetyNet Attestation API.” https://developer.android.com/training/safetynet/attestation

Appendix A: Acronyms

API	Application programming interface
CA	Certificate authority
CSRF	cross-site request forgery
DN	Distinguished Name
HTTPS	Hypertext Transfer Protocol - Secure
iGov	International Government Assurance Profile
JSON	JavaScript Object Notation
JWA	JSON Web Algorithms
JWE	JSON Web Encryption
JWK	JSON Web Keys
JWS	JSON Web Signature
JWT	JSON Web Token
NPE	Non-person entity
OIDC	OpenID Connect
PKCE	Proof Key for Code Exchange
PoP	Proof-of-Possession
SAML	Security Assertion Markup Language
URL	Uniform Resource Locator

Appendix B: Table of Requirements

The following table includes the text that it referred to as “SHOULD”, “MAY”, “MUST”, and “MUST NOT”.

Section	Text
1.	All components MUST comply with all requirements in that specification [OAuth 2.1 I-D}; this profile document levies additional requirements for the enterprise environment.
1.3	When a profile-compliant component is interacting with other profile-compliant components, in any valid combination, all components MUST implement the requirements as stated in this specification. All interaction with non-profile components is outside the scope of this specification.
1.3	A profile-compliant OAuth 2.1 client MUST support and utilize certain features as described in Section 2 of this specification.
1.3	A profile-compliant OAuth 2.1 authorization server MUST support and utilize certain features as described in Section 3 of this specification.
1.3	A profile-compliant OAuth 2.1 protected resource MUST support and utilize certain features as described in Section 4 of this specification.
1.6	Each originator [of a TLS connection] MUST have a capability to limit the certification authorities (CAs) trusted for verifying the destination's PKI certificate.
2.1.1	Confidential clients MUST possess their own asymmetric key pair used for authentication to the authorization server.
2.2	Clients MUST support the OAuth authorization code grant. Confidential clients MAY support the OAuth client credentials grant.
2.2.1	Clients MAY use *OAuth 2.0 Authorization Server Metadata* [RFC8414] to retrieve configuration information from the authorization server, including supported options, endpoint URIs, and public keys. Alternatively, clients MAY configure some or all of this information in an out-of-band manner.
2.2.2	Clients MUST include their full redirect URI in authorization requests.
2.2.2	Clients MUST use distinct redirect URIs for each authorization server as a means to identify the authorization server that a particular response came from.
2.2.2	Clients MUST check the “iss” parameter identifies the authorization server that the client made the authorization request to.
2.2.3	Confidential clients MUST support authentication to the authorization server's token endpoint using mutually authenticated TLS.
2.2.3	Public clients MAY support use of mutually authenticated TLS to the authorization server’s token endpoint.
2.2.3	Mutually authenticated TLS connections by confidential clients MUST comply with [RFC8705].
2.2.3	The self-signed certificate option described in Section 2.2 "Self-Signed Certificate Mutual TLS OAuth Client Authentication Method" MUST NOT be used. Rather, the Section 2.1 "PKI Mutual TLS OAuth Client Authentication Method" MUST be used, where the subject distinguished name (DN) of the client's certificate is registered with the authorization server.

Section	Text
2.2.3	Mutually authenticated TLS connections by public clients, if used, MUST comply with Section 4 of [RFC8705].
2.2.4	Clients MUST be registered with the authorization server.
2.2.4	Client registration MUST be completed by out-of-band configuration. Clients MUST NOT register using dynamic registration [RFC7592].
2.2.4.1	Clients running on the user's endpoint device SHOULD use [AppLinks], [UniversalLinks], or a similar capability enforced by the endpoint device platform to protect their redirect URIs as discussed in Section 8.4.1 of [OAuth2-1].
2.2.4.2	Clients using mTLS MUST register their TLS certificate's subject DN with the authorization server.
2.3.1	Clients MUST support mutually authenticated TLS to the protected resource as specified in Section 3 "Mutual TLS Client Certificate Bound Access Tokens" of [RFC8705].
3.1.1	The authorization server MUST support the authorization code grant type as described in Section 2 and MAY support the client credentials grant type.
3.1.1	Authorization codes issued by the authorization server MUST contain a minimum of 128 bits of entropy.
3.1.1	Authorization code lifetimes MUST be configurable on the authorization server. The authorization server MUST support lifetimes at least as short as 60 seconds.
3.1.1	If an authorization code is inadvertently accepted more than once, it MUST be noted in an audit log, any refresh token issued based on the authorization code MUST be revoked, and any access token issued based on the authorization code SHOULD be revoked (if it is possible to do so).
3.1.2	The authorization server MUST enforce client authentication for confidential clients.
3.1.2	The authorization server MUST support TLS client certificate authentication of confidential clients as in [RFC8705] or JWT assertion [?]. The self-signed certificate option described in Section 2.2 "Self-Signed Certificate Mutual TLS OAuth Client Authentication Method" MUST NOT be used. Rather, the Section 2.1 "PKI Mutual TLS OAuth Client Authentication Method" MUST be used, where the subject distinguished name (DN) of the client's certificate is registered with the authorization server.
3.1.2	The authorization server MUST support mTLS connections from public clients as specified in [RFC8705].
3.1.3	The authorization server MUST support the following mechanism for users to authenticate themselves to the authorization server: • TLS client certificate authentication The authorization server SHOULD support the following mechanisms for users to authenticate themselves to the authorization server: • RSA SecurID • FIDO 2.0 / W3C Web Authentication • Username and password • Federated authentication to a user's home organization using OpenID Connect (described below as identity brokering)

<table>
<tr><th>Section</th><th>Text</th></tr>
<tr><td>3.1.3</td><td>The authorization server MAY support other user authentication mechanisms. The authorization server MAY also support the ability to authenticate (and assess security properties of) the user’s endpoint device in addition to the user.</td></tr>
<tr><td>3.1.3</td><td>The authorization server MUST provide the ability for an administrator to configure which user authentication mechanisms are acceptable.</td></tr>
<tr><td>3.1.3</td><td>To authenticate users from organizations outside the authorization server’s organization, authorization servers MUST support identity brokering using OpenID Connect. If implemented, identity brokering MUST be performed in accordance with the Enterprise OpenID Connect Profile.</td></tr>
<tr><td>3.1.3</td><td>In non-enterprise environments, it is typically desired that the authorization server presents the user with the client's authorization request and require the user to explicitly approve the request. However, in this profile, the authorization server MUST provide the ability to disable such functionality.
If the end user is prompted with an interactive approval page, the authorization server MUST indicate to the user:
A human readable name of the client
What kind of access the client is requesting (including scope, target resource, etc.)</td></tr>
<tr><td>3.1.4</td><td>The authorization server MUST provide an OAuth authorization server metadata endpoint as specified by [RFC8414]. The endpoint MAY be shared with an OpenID Connect discovery endpoint. The endpoint’s response MUST contain at least the following fields and MAY contain additional fields:
<table>
<tr><td>issuer</td><td>The fully qualified issuer URL of the server</td></tr>
<tr><td>authorization_endpoint</td><td>The fully qualified URL of the server's authorization endpoint</td></tr>
<tr><td>token_endpoint</td><td>The fully qualified URL of the server's token endpoint</td></tr>
<tr><td>jwks_uri</td><td>The fully qualified URI of the server's public key in JWK Set format</td></tr>
<tr><td>introspection_endpoint</td><td>The fully qualified URL of the server's introspection endpoint defined by OAuth Token Introspection</td></tr>
<tr><td>revocation_endpoint</td><td>(only included if a revocation endpoint exists) The fully qualified URL of the server's revocation endpoint defined by [RFC7009]</td></tr>
</table></td></tr>
<tr><td>3.1.4</td><td>The authorization server MUST provide an RS256 key with a modulus of at least 2048 bits. The authorization server MAY provide additional keys using the following algorithms: RS384, RS512, ES256, ES384, ES512, PS256, PS384, PS512.</td></tr>
<tr><td>3.1.4</td><td>The authorization server MUST provide its public key (used by the authorization server to sign tokens) in JWK Set format. The key MUST contain the following fields:
<table>
<tr><td>kid</td><td>The key ID of the key pair used to sign this token</td></tr>
<tr><td>kty</td><td>The key type</td></tr>
<tr><td>alg</td><td>The default algorithm used for this key</td></tr>
</table></td></tr>
<tr><td>3.1.5</td><td>The authorization server MUST ensure that each redirect URI is one of the following:</td></tr>
</table>

<table>
<tr><th>Section</th><th>Text</th></tr>
<tr><td></td><td>

An HTTPS URI referring to a website with Transport Layer Security (TLS) protection or an app installed on the user’s endpoint using [AppLinks], [UniversalLinks], or similar capability (this method is the preferred choice)
Hosted on the user's endpoint without involving remote network connectivity (e.g., http://localhost/), however an HTTPS URI protected using [AppLinks], [UniversalLinks], or similar capability is preferred when possible
Hosted on a client-specific non-remote-protocol URI scheme (e.g., myapp://), however an HTTPS URI protected using [AppLinks], [UniversalLinks], or similar capability is preferred when possible

</td></tr>
<tr><td>3.2</td><td>The authorization server MUST be capable of enforcing an authorization policy that must be met in order for tokens to be issued. This policy MUST be customizable by the administrator. This profile does not enforce specific requirements upon capabilities of the authorization policy, but it is RECOMMENDED that at least the following attributes be considered:

Attributes associated with the user’s account, such as:

Personnel type (e.g. employee vs. contractor)
Citizenship

The user’s method(s) of authenticating to the authorization server
The protected resource being accessed
Security posture and other properties of the user’s endpoint device
IP address from which the user’s endpoint device is connecting
</td></tr>
<tr><td>3.3</td><td>To facilitate interoperability with protected resources, the authorization server MUST issue JWT access tokens compliant with [RFC9068].</td></tr>
<tr><td>3.3</td><td>The authorization server MUST include the following claims in issued tokens:
<table>
<tr><td>iss</td><td>The issuer URL of the server that issued the token.</td></tr>
<tr><td>exp</td><td>The expiration time (integer number of seconds since from 1970-01-01T00:00:00Z UTC), after which the token MUST be considered invalid.</td></tr>
<tr><td>aud</td><td>The audience of the token. An array containing the identifier(s) of protected resource(s) for which the token is valid, if this information is known. The aud claim may contain multiple values if the token is valid for multiple protected resources.</td></tr>
<tr><td>sub</td><td>The identifier of the end-user that authorized this client, or in the case of the client credentials grant, the client id of a client acting on its own behalf.</td></tr>
<tr><td>client_id</td><td>The client id of the client to whom this token was issued.</td></tr>
<tr><td>iat</td><td>Issue timestamp</td></tr>
<tr><td>jti</td><td>A unique JWT Token ID value with at least 128 bits of entropy. This value MUST NOT be re-used in another token.</td></tr>
<tr><td>cnf</td><td>Capability required for requests from confidential clients, optional for requests from public clients. Specified by Section 3 of [RFC8705] (and by Section 4 for public clients). Hash of the client’s PKI certificate that was presented using TLS mutual authentication</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Section</th><th>Text</th></tr>
<tr><td></td><td><table><tr><td></td><td>between the client and authorization server. This field binds the access token to the client's certificate, enabling the protected resource to ensure that only the authorized client can present the access token (over a mutually authenticated TLS connection).</td></tr></table></td></tr>
<tr><td>3.3</td><td>The authorization server SHOULD be capable of including additional fields in issued tokens, including the following:<table><tr><td>amr</td><td>The user’s authentication method to the AS when the user authorized issuance of this access token.</td></tr><tr><td>auth_time</td><td>Timestamp of when the user authenticated to the AS in order to authorize issuance of this access token (often not the same as iat, as the token may be issued based on a prior user authentication).</td></tr></table></td></tr>
<tr><td>3.3</td><td>The access tokens MUST be signed with JWS. The authorization server MUST support the RS256 signature method for tokens. It MAY support the following additional asymmetric signing methods defined in the IANA JSON Web Signatures and Encryption Algorithms registry: RS384, RS512, ES256, ES384, ES512, PS256, PS384, PS512. The</td></tr>
<tr><td>3.3</td><td>JWS header MUST contain the following field:<table><tr><td>kid</td><td>The key ID of the key pair used to sign this token</td></tr></table></td></tr>
<tr><td>3.3</td><td>The authorization server MAY encrypt access tokens using JWE. Encrypted access tokens MUST be encrypted using a public key designated for encryption (i.e., not used for digital signatures) of the protected resource identified in the “aud” claim.</td></tr>
<tr><td>3.4</td><td>The authorization server MUST sign refresh tokens using JWS and MAY encrypt refresh tokens using JWE. If refresh tokens are encrypted using JWE, they MUST be encrypted either using the authorization server's public key or symmetrically encrypted using a secret key held by the authorization server.</td></tr>
<tr><td>3.5.1</td><td>The authorization server MUST provide a token introspection endpoint.</td></tr>
<tr><td>3.5.1</td><td>The server MAY include additional fields in its token introspection response.</td></tr>
<tr><td>3.5.1</td><td>The authorization server MUST require mTLS authentication for the introspection endpoint.</td></tr>
<tr><td>3.5.1</td><td>A protected resource MAY cache the response from the introspection endpoint for a period of time no greater than half the lifetime of the token.</td></tr>
<tr><td>3.5.1</td><td>A protected resource MUST NOT accept a token that is not active according to the response from the introspection endpoint.</td></tr>
<tr><td>3.6</td><td>The authorization server MUST either limit access tokens to one hour, or (preferably) provide a capability to configure the access token lifetime (potentially on a per-protected resource and/or per-client basis). The authorization server SHOULD provide a capability to similarly configure refresh token lifetimes</td></tr>
<tr><td>3.7</td><td>Authorization servers MUST define and document default scope values that will be used if an authorization request does not specify a requested set of scopes.</td></tr>
<tr><td>3.7</td><td>Authorization servers MAY assert different scopes and authorization claims in the access token depending on the method used to authenticate the user.</td></tr>
<tr><td>3.7</td><td>Authorization servers MAY allow a refresh token issued for multiple scopes to be used to obtain an access token for just a subset of those scopes.</td></tr>
</table>

Section	Text
3.7	To facilitate general use across a wide variety of protected resources, authorization servers SHOULD allow for the use of arbitrary scope values at runtime, such as allowing clients or protected resources to use arbitrary scope strings upon registration.
3.8	The authorization server MUST provide an interface for end users to view a list of clients that have been granted access to resources on the user's behalf, and for end users to revoke this access.
3.8	Revocation MUST revoke any currently valid refresh tokens issued to the client to access resources on the user's behalf, SHOULD revoke applicable currently valid access tokens, and MUST prevent the client from obtaining new tokens without the authorization server receiving a new authorization request via the user.
3.8	The authorization server SHOULD provide an interface compliant with [RFC7009] for clients to request token revocation.
3.8	The authorization server MUST automatically revoke refresh tokens and SHOULD revoke access tokens under the following conditions: 3. User's account has been locked or deleted. 4. User's account credentials under which the tokens were issued have been reported lost or compromised (e.g. password, private key, hardware token, etc.).
3.9	The authorization server MUST record at least the following activities in an audit log: 4. Issuance of refresh tokens and access tokens to clients. 5. Attempted or successful use of an authorization code more than once.
4	The authorization server MAY assert different scopes and authorization claims in the access token depending on the method used to authenticate the user.
4	Authorization servers MAY allow a refresh token issued for multiple scopes to be used to obtain an access token for just a subset of those scopes.
4.1	A protected resource MUST be capable of receiving access tokens passed in the authorization header as described in [RFC6780].
4.1	Protected resources MUST define and document which scopes are required for access to the resource.
4.1	Protected resources MUST verify and interpret access tokens using either JWT, token introspection [RFC7662], or a combination of the two.
4.1	The protected resource MUST check the audience claim “aud” to ensure that it specifies the protected resource's identifier. The protected resource's identifier is the full subject distinguished name (DN) in the protected resource's certificate. The protected resource MUST ensure that the rights associated with the token are sufficient to grant access to the resource. The protected resource SHOULD enforce whatever authorization policy is appropriate for the resource and not depend solely on OAuth.
4.1	Protected resources MUST support mutual TLS client certificate bound access tokens as specified in [RFC8705] Section 3.
4.1	If the protected resource uses token introspection, the protected resource MAY cache the response from the introspection endpoint for a period of time no greater

Section	Text
	than half the lifetime of the token, and a protected resource MUST NOT accept a token that is not active according to the response from the introspection endpoint.
4.2	Protected resources MAY use [RFC8414] to retrieve configuration information from the authorization server, including supported options, endpoint URIs, and public keys. Alternatively, protected resources MAY configure some or all of this information in an out-of-band manner.
4.2	Protected resources MAY use token introspection [RFC7662] to connect to the authorization server to retrieve information about an access token presented by a client. If the protected resource uses token introspection, it MUST connect to the authorization server using mTLS
5.	All transactions MUST be protected in transit by TLS as described in BCP195