Web Authorization Protocol
Internet-Draft
Intended status: Informational
Expires: 26 December 2025

P. Kasselman
SPIRL
D. Sneeggen
Signicat

24 June 2025

OAuth Client Registration on First Use with SPIFFE
draft-kasselman-oauth-spiffe-01

Abstract

The OAuth framework is a widely deployed authorization protocol standard that enables applications to obtain limited access to user resources. OAuth clients must be registered with the OAuth authorization server, which poses significant operational challenges in dynamically scaling environments. The Secure Production Identity Framework For Everyone (SPIFFE) is a graduated Cloud Native Compute Foundation project designed to dynamically attest and verify workload identity. This draft describes how workloads with SPIFFE credentials can be used with OAuth to lessen the operational burden of client registration through a "register on first use" principle.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://PieterKas.github.io/OAuth-and-SPIFFE/draft-kasselman-oauth_spiffe.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-kasselman-oauth-spiffe/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/PieterKas/OAuth-and-SPIFFE.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

The OAuth framework [RFC6749] is popular and broadly deployed. It defines separate roles for the authorization server, resource server, resource owner and the client. In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner [RFC6749]. The OAuth client is identified with a client identifier and supports multiple authentication methods, such as the commonly-used client secret. The management of client identifiers and client secrets represents operational challenges, including manual registrations, lifecycle management and the increasing burden of managing client secrets used for authenticating OAuth clients. Modern cloud-native architectural patterns, like micro-services and other ephemeral compute concepts, allows for on-demand scaling of OAuth clients, which in turn increases the burden on managing the lifecycle of the client and client secrets.

The Secure Production Identity Production Framework (SPIFFE) is a graduated project in the Cloud Native Compute Foundation (CNCF) that defines a standard for managing the identity lifecycle of workloads in modern cloud-native compute environments. It is designed to dynamically attest and verify a workloads, assign identifiers, issue credentials and manage the lifecycle of those credentials in dynamically scaled environments without manual intervention. The identifier is an URI, while the credentials include profiles of JWT and X.509 certificates, known as JWT-SVID and X.509-SVIDs. These credentials are used by workloads to authenticate to each other or to services that they need to access. SPIFFE is commonly used to secure workload identities in large-scale production systems that need to scale dynamically.

Workloads provisioned with SPIFFE identifiers and credentials often need to access OAuth-protected resources. When they interact with OAuth-protected resources, they assume the role of an OAuth client and need to be registered with the OAuth authorization server. This requries an additional manual step, or an additional registration flow (e.g. Dynamic Client Registration [RFC7591]). The registration step results in the provisioning of an addditional identifier (the client identifier), and frequently an additional secret (the Client Secret) used for client authentication and must be secured. The additional secrets add to the overall challenge of secrets sprawl in large-scale environments, and degrade the risk profile for an environment.

OAuth deployments in which SPIFFE is already deployed can leverage the SPIFFE identifiers and credentials already issued to workloads by establishing a trust relationship with the SPIFFE issuer. When a workload acts as an OAuth client, it presents a SPIFFE credential to authenticate itself to the OAuth server. The OAuth authorization server verifies the JWT or X.509 certificate, ensuring it was issued by a trusted issuer. The authorization server registers the SPIFFE ID as a client identifier, along with additional metadata, if needed, before issuing an Access Token.

This "register on first use" pattern removes the need for manual pre-registration or additional roundtrips and credential registration. It leverages existing credentials that were issued dynamically and reduces the dependence on static secrets which are at higher risk of compromise than ephemeral dynamic credentials.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Client Registration On First Use

The SPIFFE deployment is responsible for bootstrapping the identifiers and credentials used by workloads in an environment. The SPIFFE deployment dynamically attests workloads, issues identifiers, provisions ephemeral credentials, and rotates those credentials. The attestation techniques used by a SPIFFE issuer is deployment-specific. SPIFFE supports multiple credential formats, including JWT-SVID [SPIFFE_JWT] and JWT-X.509 [SPIFFE_X509]. Workloads can use these credentials with any protocol that is compatible with these credential types to authenticate themselves. The use of SPIFFE

credentials to authenticate to an OAuth authorization server is described in [SPIFFE-OAUTH-CLIENT-AUTH].

An OAuth deployment leverages the SPIFFE identifiers and credentials already issued to workloads by establishing a trust relationship with the SPIFFE issuer. As a result, the OAuth authorization server is able to verify any credential issued for a specific SPIFFE trust domain when it is presented to the OAuth authorization server. This assures the OAuth authorization server that the workload was attested and verified in acccordance with the SPIFFE issuer’s policies. In addition, it is assured that the identifier was correctly assigned to the workload, and that the lifecycle of the credentials and the underlying keys are managed in accordance with the SPIFFE issuer’s policies.

The OAuth authorization server avoids operational overheads of registering the client and client secrets by accepting SPIFFE credentials from a workload acting as an OAuth client. The credentials must be signed by a trusted issuer. If this is the first time the authorization server verifies a credential with a specific SPIFFE ID, it registers a new client, removing the need for an out-of-band registration flow. The OAuth authorization server may derive a unique client identifier from the SPIFFE ID, or it may use the SPIFFE ID as the client identifier. Since the workload is already in possession of a credential that is cryptographically bound to its identifier, no addditional client secret is needed, removing the operational overhead and security risks of managing long-lived secrets in large-scale deployments that already have SPIFFE credentials. The OAuth authorization server may register additional metadata if needed. The authorization server may use a number of mechanissms for obtaining additional metadata. Metadata may be preconfigured, automatically created, retrieved from a configuration management system, or retrieved from the JWT-SVID, if it was included as additional claims, similar to how Dynamic Client Registration ([RFC7591]) describes how a software statement may be used to convey metadata.

"Register on first use" varies depending whether or not the OAuth flows includes redirection. In non-redirecting flows (such as Client Credentials or Resource Owner Password Credentials), the OAuth client authenticates and requests tokens directly from the authorization server without the need for user redirection. Flows with redirection (like Authorization Code Flow) temporarily send users to an external identity provider before returning them to the application with a token or authorization code, which is then used by the OAuth client. These redirect-based flows rely on the user agent (typically a web browser) to maintain the authentication context and handle the navigation between the application and the identity provider. These

differences require distinct implementation considerations when implementing client registration on first use. Redirecting flows need state management to reconnect returning users with their original session after authentication, while non-redirecting flows can create a client identifier inline within the same request context. Properly handling both OAuth patterns ensures a smooth registration experience regardless of the authentication method selected.

3.1. Client Registration On First Use for Non-Redirecting OAuth Flows

In OAuth flows where there is no redirection, the client initiates interaction with the authorization server and accesses the token endpoint directly without any preceding redirects. Examples include the Client Credentials (see Section 4.4 of [RFC6749]) and Resource Owner Password Credentials (see Section 4.3 of [RFC6749]).

The diagram below shows the process for "register on first use" in the Client Credential flow.

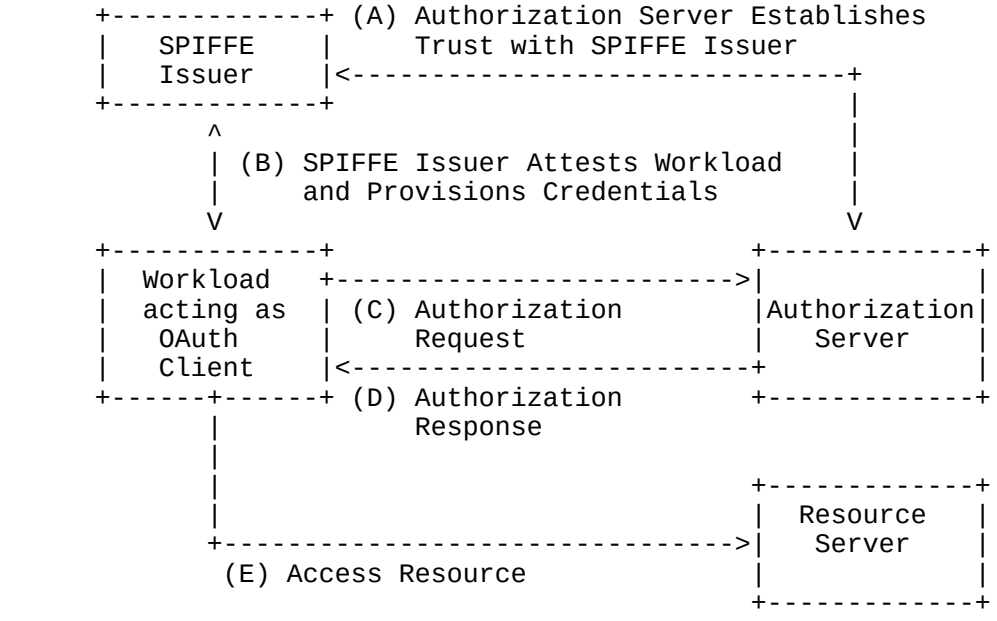

Figure 1: Client Registration on First Use: Client Credentials Grant

* (A) The OAuth authorization server establishes a trust relationship with the SPIFFE Issuer. Once trust is established, the OAuth authorization server accepts credentials issued by the SPIFFE Issuer.

* (B) The workload is attested and credentialed by the SPIFFE Issuer [SPIFFE]. The SPIFFE Issuer assigns a SPIFFE ID [SPIFFE_ID] as an identifier for the workload, along with SPIFFE credentials (e.g. JWT-SVID [SPIFFE_JWT], X.509-SVID [SPIFFE_X509] or other SPIFFE credential types).

* (C) The workload, acting as an OAuth client, makes a request to the authorization server's token endpoint as specified in Section 4.4 of [RFC6749]. It authenticates itself using either a JWT-SVID or X.509-SVID as defined in [SPIFFE-OAUTH-CLIENT-AUTH]

* (D) The OAuth authorization server authenticates the workload acting as an OAuth client by verifying the credentials it presented (see [SPIFFE-OAUTH-CLIENT-AUTH]). If the authentication is successful, the authorization server accepts the SPIFFE ID as a valid client identifier and checks if it has been registered previously. If it has not been registered, it registers the new SPIFFE ID as a valid client. The authorization server adds additional metadata if needed, before completing the request and returning an access token.

* (E) The Oauth client use the access token it retrieved in the previous step and use it to acccess the resource server.

3.2. Client Registration On First Use for Redirecting OAuth Flows

In OAuth flows that rely on redirection, the initial interaction with the authorization server is not performed by the client, but is instead performed by another component, such as the user agent. These flows typically include interaction with the Resource Owner in order to perform user authentication and grant authorization. Once the Resource Owner authenticated and granted authorization, the flow is redirected back to the client. The client then interacts with the authorization server token endpoint to complete the flow and obtain tokens. Examples include the Authorization Grant Flow (see Section 4.1 of [RFC6749]).

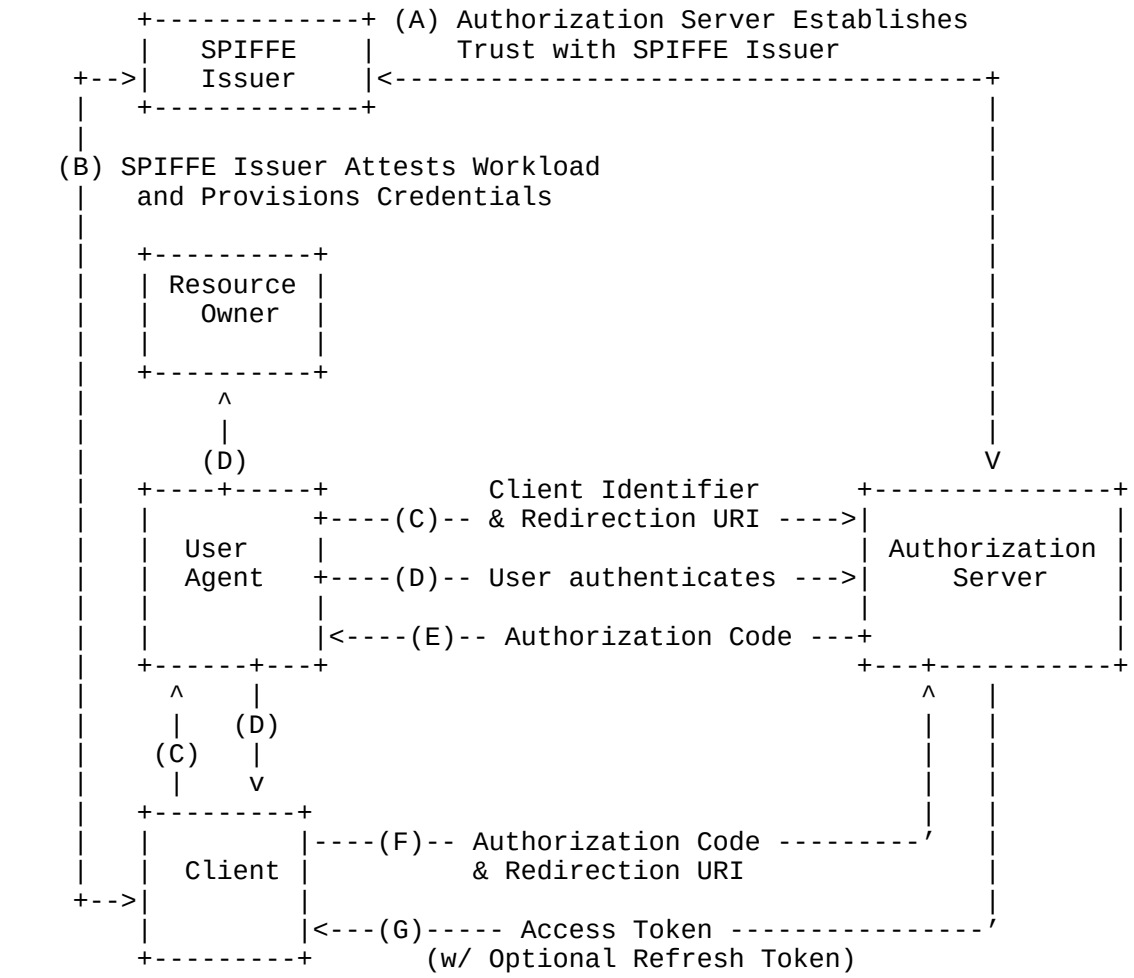

Figure 2: Client Registration on First Use: Authorization Code Grant

* (A) The OAuth authorization server establishes a trust relationship with the SPIFFE Issuer. Once trust is established, the OAuth authorization server accepts credentials issued by the SPIFFE Issuer.

* (B) The workload is attested and credentialed by the SPIFFE Issuer [SPIFFE]. The SPIFFE Issuer assigns a SPIFFE ID [SPIFFE_ID] as an identifier for the workload, along with SPIFFE credentials (e.g. JWT-SVID [SPIFFE_JWT] and X.509-SVID [SPIFFE_X509]).

* (C) The client initiates the flow by directing the resource owner's user agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user agent back once access is granted (or denied).

* (D) The authorization server authenticates the resource owner (via the user agent) and establishes whether the resource owner grants or denies the client's access request. If the client identifier (client_id) is a SPIFFE ID, it registers the SPIFFE ID as the client identifier and register additional metadata, including a redirection URI (redirect_uri). The additional metadata may be obtained from configuration servers or other out-of-band mechanisms that is trusted by the authorization server.

* (E) Assuming the resource owner grants access, the authorization server redirects the user agent back to the client using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier.

* (F) The client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. It authenticates itself using either a JWT-SVID or X.509-SVID as defined in [SPIFFE-OAUTH-CLIENT-AUTH]. The client includes the redirection URI used to obtain the authorization code for verification.

* (G) The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step (E). If valid, the authorization server responds back with an access token and, optionally, a refresh token.

3.3. Client Registration Error Response

When the registration attempt has failed or been rejected, the authorization server MUST return an error response that conforms to the expected error response for the given OAuth 2.0 endpoint. Additional error information may or may not be included in the error response.

TODO: Generally improve error section, inline with RFC 6749. Consider also how to return information which can aid the workload or workload owners to resolve failed registration or complete an incomplete registration.

3.3.1. Authentication Failed

When a SPIFFE credential is missing, invalid, or for any reason rejected, the authorization server returns an error response with an HTTP 401 status code.

3.3.2. Failed Registration

When a request to register a client fails due to disallowed metadata (e.g., http redirect URIs if only https is allowed, or an access token lifetime exceeding maximum allowed) or invalid metadata, the authorization server returns an error response with an HTTP 400 status code.

3.3.3. Incomplete Registration

When additional post-registration setup is required, there can be cases where the client cannot be used. In such scenarios, the authorization server returns an error response with an HTTP 403 status code. This error could also be returned in cases where there is an asynchronous event or manual operation that is not yet completed.

4. SPIFFE and OAuth Trust Relationship

SPIFFE makes provision for multiple Trust Domains, which are represented in the workload identifier. Trust Domains offers additional segmentation withing a SPIFFE deployment and each Trust Domain has its own keys for signing credentials. The OAuth authorization server may choose to trust one or more trust domains as defined in [SPIFFE-OAUTH-CLIENT-AUTH].

4.1. Client Authentication

The client MUST authenticate itself using either a JWT-SVID or X.509-SVID as defined in [SPIFFE-OAUTH-CLIENT-AUTH].

4.2. Authorization Server Processing

When presented with a SPIFFE ID that is used as a client identifier, the authorization server SHOULD register it as a valid client identifier. There are two cases:

4.2.1. Non-redirect flows

In non-redirect flows, the client is intereacting directly with the token endpoint. The authorization server MUST authenticate the client as described in JWT-SVID or X.509-SVID as defined in [SPIFFE-OAUTH-CLIENT-AUTH]. If the JWT-SVID or X.509-SVID used to authenticate the client was issued by a trusted SPIFFE issuer, and the authentication succeeds, the SPIFFE ID SHOULD be registered as the client identifier.

4.2.2. Redirect flows

Redirect flows typically require interaction with the Resource Owner to perform user authentication. In redirect flows, another component such as the user agent, interacts with the authorization server and relies on a redirection back to the client. Since the client does not interact directly with the authorization server in this initial interaction, it is not in a position to authenticate itself using a JWT-SVID or X.509-SVID. The user agent includes a client identifier (the SPIFFE ID) which the authorization server registers as a new client after obtaining additional metadata. The metadata may be derived from the SPIFFE ID, or it may be retrieved from a configuration server. Details of obtaining the additional metadata is implementation-specific and beyond the scope of this document. If one or more redirection URIs are obtained from another source, it MUST be the same as that presented by the user agent.

Following the redirect back to the client, the client must authenticate to the authorization server as described in JWT-SVID or X.509-SVID as defined in [SPIFFE-OAUTH-CLIENT-AUTH]. If the authentication fails, or the SPIFFE ID in the JWT-SVID or X.509-SVID does not match the client identifier, the request should be denied and the authorization server MAY de-register the client identifier used. If the authentication succeeds, it confirms that the client was issued a credential by a trusted SPIFFE issuer and the authorization server issues the requested tokens.

5. Additional metadata

Additional metadata may be required when a new client is first registered. The OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] defines client metadata that may be used. If required, client metadata SHOULD be added at the time of registration. Metadata may be dynamically generated or derived at the time of registration, or it may be retrieved from a system of record using the SPIFFE ID to locate the metadata in an enterprise environment.

5.1. Metadata as claims in the JWT-SVID

A SPIFFE issuer MAY include additional metadata as claims in the JWT-SVID, similar to how it may be included in a software statement as defined in the OAuth 2.0 Dynamic Registration Protocol [RFC7591]. If the additional metadata claims are used, the claims in the JWT-SVID should be processed in accordance with [RFC7591] as if they were included in a software statement.

5.2. The redirect URI

Flows that depend on redirection requires that a redirect URI is registered along with every client identifier. Another component, like a user agent, may contact the authorization server, before being redirected to the client. In order to avoid attacks such as cross-site request forgery, open redirector attacks and similar attacck described in OAuth 2.0 Threat Model and Security Considerations [RFC6819] and OAuth 2.0 Security Best Current Practice [RFC9700], the redirect_uri MUST be provisioned from a trustworthy source if it is required.

5.3. Scopes

An authorization server MAY register a client with a default set of scopes or retrieve client-specific scopes from a system of record, such as a configuration management system. Details of how to retrieve additional scope data is out of scope for this document.

5.4. Grant Types

Authorization servers MAY choose to limit the grant types for which the "register on first use" pattern is supported. This may be recorded as the "grant_type" metadata field.

6. Post-Registration Client Lifecycle Management

After registration, there MUST be an initial client record with a direct link between the SPIFFE identifier in the SPIFFE credentials and the OAuth client identifier. However, additional steps MAY be required to make the client operational, such as missing configuration or entitlement information. This is particularly relevant for complex enterprise environments.

6.1. Configuration

After registration, a client MUST be configured with the necessary operational metadata to function correctly and securely. An authorization server may use a number of mechanisms for obtaining metadata. Metadata may be statically preconfigured, automatically or manually created, or retrieved from a configuration management system. This can happen instantaneously after registration or it can be asynchronous. If metadata is added asynchronously, the authorization server MUST return an "Incomplete Registration" error whenever the client interacts with the authorization server, until the additional metadata has been added.

6.2. Entitlement

Entitlement defines the specific permissions and/or access rights granted to the client. This is a critical stage that determines what the configured client is permitted to do and request. Entitlements can include; grant types, scopes, audience restrictions, or fine-grained permissions.

In enterprise environments, the permissions and access rights granted to a client can be highly dynamic and complex. As such, there might be several out-of-band operations; creating a principal for the client, assigning permissions and roles to the client in a PRP system, assigning attributes to the workload, or any other mechanism used for making access rights evaluations.

6.3. Updates

Updates to client configuration or entitlement may occur using the same "register on first use" mechanism and can even be entirely opaque to the workload. However, special care must be taken to ensure this happens securely and in line with organizational policies.

6.4. Deregistration

An authorization server MAY automatically expire clients. This avoids a large number of unused clients identifiers from accumulating. If a client identifier is deleted, it can re-register using the "register-on-first-use" pattern described in this document.

7. Security Considerations

TODO: Security. Consider discussing error responses (include enough info to be helpful, but not too much to aid attackers.)

7.1. Entitlement Updates

If entitlement updates, such as client permissions, can be updated dynamically, be aware that this can lead to potential privilege escalation if a workload is compromised. Similarly, adding new redirects to an existing client can also lead to potential issues. The implementer needs to make clear decisions on whether updates to clients are allowed and, if so, what types of updates are permitted. Risk analysis could also be introduced to determine what types of client updates are allowed.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[Headless_JWT]
"Headless-JWT", n.d., <foo>.

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

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

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/rfc/rfc6755>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/rfc/rfc7521>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/rfc/rfc7523>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/rfc/rfc7591>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/rfc/rfc8705>.

[SPIFFE] "SPIFFE", n.d., <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE.md>.

[SPIFFE-OAUTH-CLIENT-AUTH]
"OAuth SPIFFE Client Authentication", n.d., <foo>.

[SPIFFE_BUNDLE]
"SPIFFE Bundle", n.d., <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE_Trust_Domain_and_Bundle.md#4-spiffe-bundle-format>.

[SPIFFE_FEDERATION]
"SPIFFE Federation", n.d., <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE_Federation.md>.

[SPIFFE_ID]
"SPIFFE-ID", n.d., <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE-ID.md>.

[SPIFFE_JWT]
"JWT-SVID", n.d., <https://github.com/spiffe/spiffe/blob/main/standards/JWT-SVID.md>.

[SPIFFE_X509]
"X509-SVID", n.d., <https://github.com/spiffe/spiffe/blob/main/standards/X509-SVID.md>.

9.2. Informative References

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/rfc/rfc6819>.

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/rfc/rfc9700>.

Appendix A. Acknowledgments

The author would like to thank Dmitry Telegin (Backbase), Dean Saxe, Arndt Schwenkschuster (SPIRL) and Marcel Levy (SPIRL) and others (please let us know, if you’ve been mistakenly omitted) for their valuable input, feedback and general support of this work.

Appendix B. Document History

[[To be removed from the final specification]] -latest * Initial draft submitted to Datatracker

Authors’ Addresses

Pieter Kasselman
SPIRL
Email: pieter@spirl.com

Dag Sneeggen
Signicat
Email: dagsne@signicat.com