OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 13, 2019

B. Campbell
Ping Identity
J. Bradley
Yubico
N. Sakimura
Nomura Research Institute
T. Lodderstedt
YES.com AG

April 11, 2019

OAuth 2.0 Mutual TLS Client Authentication and Certificate-Bound Access Tokens
draft-ietf-oauth-mtls-14

Abstract

This document describes OAuth client authentication and certificate-bound access and refresh tokens using mutual Transport Layer Security (TLS) authentication with X.509 certificates. OAuth clients are provided a mechanism for authentication to the authorization server using mutual TLS, based on either self-signed certificates or public key infrastructure (PKI). OAuth authorization servers are provided a mechanism for binding access tokens to a client's mutual TLS certificate, and OAuth protected resources are provided a method for ensuring that such an access token presented to it was issued to the client presenting the token.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 13, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The OAuth 2.0 Authorization Framework [RFC6749] enables third-party client applications to obtain delegated access to protected resources. In the prototypical abstract OAuth flow, illustrated in Figure 1, the client obtains an access token from an entity known as an authorization server and then uses that token when accessing protected resources, such as HTTPS APIs.

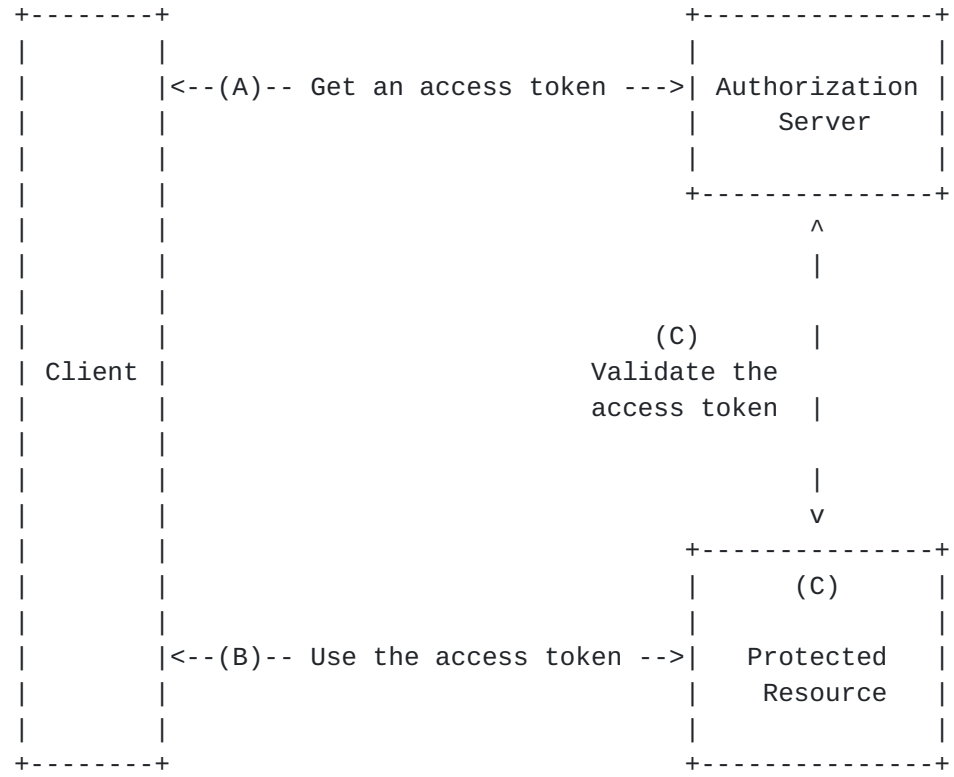

Figure 1: Abstract OAuth 2.0 Protocol Flow

The flow illustrated in Figure 1 includes the following steps:

(A) The client makes an HTTPS "POST" request to the authorization server and presents a credential representing the authorization grant. For certain types of clients (those that have been issued or otherwise established a set of client credentials) the request must be authenticated. In the response, the authorization server issues an access token to the client.

(B) The client includes the access token when making a request to access a protected resource.

(C) The protected resource validates the access token in order to authorize the request. In some cases, such as when the token is self-contained and cryptographically secured, the validation can be done locally by the protected resource. While other cases require that the protected resource call out to the authorization server to determine the state of the token and obtain meta-information about it.

Layering on the abstract flow above, this document standardizes enhanced security options for OAuth 2.0 utilizing client certificate based mutual TLS. Section 2 provides options for authenticating the request in step (A). While step (C) is supported with semantics to express the binding of the token to the client certificate for both local and remote processing in Section 3.1 and Section 3.2 respectively. This ensures that, as described in Section 3, protected resource access in step (B) is only possible by the legitimate client bearing the access token and holding the private key corresponding to the certificate.

OAuth 2.0 defines a shared secret method of client authentication but also allows for definition and use of additional client authentication mechanisms when interacting directly with the authorization server. This document describes an additional mechanism of client authentication utilizing mutual TLS certificate-based authentication, which provides better security characteristics than shared secrets. While [RFC6749] documents client authentication for requests to the token endpoint, extensions to OAuth 2.0 (such as Introspection [RFC7662], Revocation [RFC7009], and the Backchannel Authentication Endpoint in [OpenID.CIBA]) define endpoints that also utilize client authentication and the mutual TLS methods defined herein are applicable to those endpoints as well.

Mutual TLS certificate-bound access tokens ensure that only the party in possession of the private key corresponding to the certificate can utilize the token to access the associated resources. Such a constraint is sometimes referred to as key confirmation, proof-of-

possession, or holder-of-key and is unlike the case of the bearer token described in [RFC6750], where any party in possession of the access token can use it to access the associated resources. Binding an access token to the client's certificate prevents the use of stolen access tokens or replay of access tokens by unauthorized parties.

Mutual TLS certificate-bound access tokens and mutual TLS client authentication are distinct mechanisms, which are complementary but don't necessarily need to be deployed or used together.

1.1. Requirements Notation and Conventions

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.

1.2. Terminology

Throughout this document the term "mutual TLS" refers to the process whereby a client presents its X.509 certificate and proves possession of the corresponding private key to a server when negotiating a TLS session. In contemporary versions of TLS [RFC8446] [RFC5246] this requires that the client send the Certificate and CertificateVerify messages during the handshake and for the server to verify the CertificateVerify and Finished messages.

2. Mutual TLS for OAuth Client Authentication

This section defines, as an extension of OAuth 2.0, Section 2.3 [RFC6749], two distinct methods of using mutual TLS X.509 client certificates as client credentials. The requirement of mutual TLS for client authentication is determined by the authorization server based on policy or configuration for the given client (regardless of whether the client was dynamically registered, statically configured, or otherwise established).

In order to utilize TLS for OAuth client authentication, the TLS connection between the client and the authorization server MUST have been established or reestablished with mutual TLS X.509 certificate authentication (i.e. the Client Certificate and Certificate Verify messages are sent during the TLS Handshake).

For all requests to the authorization server utilizing mutual TLS client authentication, the client MUST include the "client_id" parameter, described in OAuth 2.0, Section 2.2 [RFC6749]. The

presence of the "client_id" parameter enables the authorization server to easily identify the client independently from the content of the certificate. The authorization server can locate the client configuration using the client identifier and check the certificate presented in the TLS Handshake against the expected credentials for that client. The authorization server MUST enforce the binding between client and certificate as described in either Section 2.1 or Section 2.2 below.

2.1. PKI Mutual TLS Method

The PKI (public key infrastructure) method of mutual TLS OAuth client authentication adheres to the way in which X.509 certificates are traditionally used for authentication. It relies on a validated certificate chain [RFC5280] and a single subject distinguished name (DN) or a single subject alternative name (SAN) to authenticate the client. Only one subject name value of any type is used for each client. The TLS handshake is utilized to validate the client's possession of the private key corresponding to the public key in the certificate and to validate the corresponding certificate chain. The client is successfully authenticated if the subject information in the certificate matches the single expected subject configured or registered for that particular client (note that a predictable treatment of DN values, such as the distinguishedNameMatch rule from [RFC4517], is needed in comparing the certificate's subject DN to the client's registered DN). Revocation checking is possible with the PKI method but if and how to check a certificate's revocation status is a deployment decision at the discretion of the authorization server. Clients can rotate their X.509 certificates without the need to modify the respective authentication data at the authorization server by obtaining a new certificate with the same subject from a trusted certificate authority (CA).

2.1.1. PKI Method Metadata Value

For the PKI method of mutual TLS client authentication, this specification defines and registers the following authentication method metadata value into the "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters].

tls_client_auth
: Indicates that client authentication to the authorization server will occur with mutual TLS utilizing the PKI method of associating a certificate to a client.

2.1.2. Client Registration Metadata

In order to convey the expected subject of the certificate, the following metadata parameters are introduced for the OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] in support of the PKI method of mutual TLS client authentication. A client using the "tls_client_auth" authentication method MUST use exactly one of the below metadata parameters to indicate the certificate subject value that the authorization server is to expect when authenticating the respective client.

tls_client_auth_subject_dn
: An [RFC4514] string representation of the expected subject distinguished name of the certificate, which the OAuth client will use in mutual TLS authentication.

tls_client_auth_san_dns
: A string containing the value of an expected dNSName SAN entry in the certificate, which the OAuth client will use in mutual TLS authentication.

tls_client_auth_san_uri
: A string containing the value of an expected uniformResourceIdentifier SAN entry in the certificate, which the OAuth client will use in mutual TLS authentication.

tls_client_auth_san_ip
: A string representation of an IP address in either dotted decimal notation (for IPv4) or colon-delimited hexadecimal (for IPv6, as defined in [RFC4291] section 2.2) that is expected to be present as an iPAddress SAN entry in the certificate, which the OAuth client will use in mutual TLS authentication.

tls_client_auth_san_email
: A string containing the value of an expected rfc822Name SAN entry in the certificate, which the OAuth client will use in mutual TLS authentication.

2.2. Self-Signed Certificate Mutual TLS Method

This method of mutual TLS OAuth client authentication is intended to support client authentication using self-signed certificates. As pre-requisite, the client registers its X.509 certificates (using "jwks" defined in [RFC7591]) or a trusted source for its X.509 certificates (using "jwks_uri" from [RFC7591]) with the authorization server. During authentication, TLS is utilized to validate the client's possession of the private key corresponding to the public key presented within the certificate in the respective TLS handshake.

In contrast to the PKI method, the client's certificate chain is not validated by the server in this case. The client is successfully authenticated if the certificate that it presented during the handshake matches one of the certificates configured or registered for that particular client. The Self-Signed Certificate method allows the use of mutual TLS to authenticate clients without the need to maintain a PKI. When used in conjunction with a "jwks_uri" for the client, it also allows the client to rotate its X.509 certificates without the need to change its respective authentication data directly with the authorization server.

2.2.1. Self-Signed Method Metadata Value

For the Self-Signed Certificate method of mutual TLS client authentication, this specification defines and registers the following authentication method metadata value into the "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters].

self_signed_tls_client_auth
: Indicates that client authentication to the authorization server will occur using mutual TLS with the client utilizing a self-signed certificate.

2.2.2. Client Registration Metadata

For the Self-Signed Certificate method of binding a certificate with a client using mutual TLS client authentication, the existing "jwks_uri" or "jwks" metadata parameters from [RFC7591] are used to convey the client's certificates via JSON Web Key (JWK) in a JWK Set (JWKS) [RFC7517]. The "jwks" metadata parameter is a JWK Set containing the client's public keys as an array of JWKs while the "jwks_uri" parameter is a URL that references a client's JWK Set. A certificate is represented with the "x5c" parameter of an individual JWK within the set. Note that the members of the JWK representing the public key (e.g. "n" and "e" for RSA, "x" and "y" for EC) are required parameters per [RFC7518] so will be present even though they are not utilized in this context. Also note that that sec 4.7 of [RFC7517] requires that the key in the first certificate of the "x5c" parameter match the public key represented by those other members of the JWK.

3. Mutual TLS Client Certificate-Bound Access Tokens

When mutual TLS is used by the client on the connection to the token endpoint, the authorization server is able to bind the issued access token to the client certificate. Such a binding is accomplished by associating the certificate with the token in a way that can be accessed by the protected resource, such as embedding the certificate

hash in the issued access token directly, using the syntax described in Section 3.1, or through token introspection as described in Section 3.2. Binding the access token to the client certificate in that fashion has the benefit of decoupling that binding from the client's authentication with the authorization server, which enables mutual TLS during protected resource access to serve purely as a proof-of-possession mechanism. Other methods of associating a certificate with an access token are possible, per agreement by the authorization server and the protected resource, but are beyond the scope of this specification.

The client makes protected resource requests as described in [RFC6750], however, those requests MUST be made over a mutually authenticated TLS connection using the same certificate that was used for mutual TLS at the token endpoint.

The protected resource MUST obtain, from its TLS implementation layer, the client certificate used for mutual TLS and MUST verify that the certificate matches the certificate associated with the access token. If they do not match, the resource access attempt MUST be rejected with an error per [RFC6750] using an HTTP 401 status code and the "invalid_token" error code.

Metadata to convey server and client capabilities for mutual TLS client certificate-bound access tokens is defined in Section 3.3 and Section 3.4 respectively.

3.1. JWT Certificate Thumbprint Confirmation Method

When access tokens are represented as JSON Web Tokens (JWT)[RFC7519], the certificate hash information SHOULD be represented using the "x5t#S256" confirmation method member defined herein.

To represent the hash of a certificate in a JWT, this specification defines the new JWT Confirmation Method [RFC7800] member "x5t#S256" for the X.509 Certificate SHA-256 Thumbprint. The value of the "x5t#S256" member is a base64url-encoded [RFC4648] SHA-256 [SHS] hash (a.k.a. thumbprint, fingerprint or digest) of the DER encoding of the X.509 certificate [RFC5280]. The base64url-encoded value MUST omit all trailing pad '=' characters and MUST NOT include any line breaks, whitespace, or other additional characters.

The following is an example of a JWT payload containing an "x5t#S256" certificate thumbprint confirmation method.

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

Figure 2: Example JWT Claims Set with an X.509 Certificate Thumbprint Confirmation Method

3.2. Confirmation Method for Token Introspection

OAuth 2.0 Token Introspection [RFC7662] defines a method for a protected resource to query an authorization server about the active state of an access token as well as to determine meta-information about the token.

For a mutual TLS client certificate-bound access token, the hash of the certificate to which the token is bound is conveyed to the protected resource as meta-information in a token introspection response. The hash is conveyed using the same "cnf" with "x5t#S256" member structure as the certificate SHA-256 thumbprint confirmation method, described in Section 3.1, as a top-level member of the introspection response JSON. The protected resource compares that certificate hash to a hash of the client certificate used for mutual TLS authentication and rejects the request, if they do not match.

The following is an example of an introspection response for an active token with an "x5t#S256" certificate thumbprint confirmation method.

```
HTTP/1.1 200 OK
Content-Type: application/json

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

Figure 3: Example Introspection Response for a Certificate-Bound Access Token

3.3. Authorization Server Metadata

This document introduces the following new authorization server metadata parameter to signal the server's capability to issue certificate bound access tokens:

tls_client_certificate_bound_access_tokens
: OPTIONAL. Boolean value indicating server support for mutual TLS client certificate-bound access tokens. If omitted, the default value is "false".

3.4. Client Registration Metadata

The following new client metadata parameter is introduced to convey the client's intention to use certificate bound access tokens:

tls_client_certificate_bound_access_tokens
: OPTIONAL. Boolean value used to indicate the client's intention to use mutual TLS client certificate-bound access tokens. If omitted, the default value is "false".

4. Public Clients and Certificate-Bound Tokens

Mutual TLS OAuth client authentication and certificate-bound access tokens can be used independently of each other. Use of certificate-bound access tokens without mutual TLS OAuth client authentication, for example, is possible in support of binding access tokens to a TLS client certificate for public clients (those without authentication credentials associated with the "client_id"). The authorization server would configure the TLS stack in the same manner as for the Self-Signed Certificate method such that it does not verify that the

certificate presented by the client during the handshake is signed by a trusted CA. Individual instances of a client would create a self-signed certificate for mutual TLS with both the authorization server and resource server. The authorization server would not use the mutual TLS certificate to authenticate the client at the OAuth layer but would bind the issued access token to that certificate, for which the client has proven possession of the corresponding private key. The access token is then bound to the certificate and can only be used by the client possessing the certificate and corresponding private key and utilizing them to negotiate mutual TLS on connections to the resource server. When the authorization server issues a refresh token to such a client, it SHOULD also bind the refresh token to the respective certificate. And check the binding when the refresh token is presented to get new access tokens. The implementation details of the binding the refresh token are at the discretion of the authorization server.

5. Metadata for Mutual TLS Endpoint Aliases

The process of negotiating client certificate-based mutual TLS involves a TLS server requesting a certificate from the TLS client (the client does not provide one unsolicited). Although a server can be configured such that client certificates are optional, meaning that the connection is allowed to continue when the client does not provide a certificate, the act of a server requesting a certificate can result in undesirable behavior from some clients. This is particularly true of web browsers as TLS clients, which will typically present the end-user with an intrusive certificate selection interface when the server requests a certificate.

Authorization servers supporting both clients using mutual TLS and conventional clients MAY chose to isolate the server side mutual TLS behaviour to only clients intending to do mutual TLS, thus avoiding any undesirable effects it might have on conventional clients. The following authorization server metadata parameter is introduced to facilitate such separation:

mtls_endpoint_aliases
: OPTIONAL. A JSON object containing alternative authorization server endpoints that, when present, an OAuth client intending to do mutual TLS uses in preference to the conventional endpoints. The parameter value itself consists of one or more endpoint parameters, such as "token_endpoint", "revocation_endpoint", "introspection_endpoint", etc., conventionally defined for the top-level of authorization server metadata. An OAuth client intending to do mutual TLS (for OAuth client authentication and/or to acquire or use certificate-bound tokens) when making a request directly to the authorization server MUST use the alias URL of the

endpoint within the "mtls_endpoint_aliases", when present, in preference to the endpoint URL of the same name at top-level of metadata. When an endpoint is not present in "mtls_endpoint_aliases", then the client uses the conventional endpoint URL defined at the top-level of the authorization server metadata. Metadata parameters within "mtls_endpoint_aliases" that do not define endpoints to which an OAuth client makes a direct request have no meaning and SHOULD be ignored.

Below is an example of an authorization server metatdata document with the "mtls_endpoint_aliases" parameter, which indicates aliases for the token, revocation, and introspection endpoints that an OAuth client intending to do mutual TLS would in preference to the conventional token, revocation, and introspection endpoints. Note that the endpoints in "mtls_endpoint_aliases" use a different host than their conventional counterparts, which allows the authorization server (via SNI or actual distinct hosts) to differentiate its TLS behavior as appropriate.

```
{
  "issuer": "https://server.example.com",
  "authorization_endpoint": "https://server.example.com/authz",
  "token_endpoint": "https://server.example.com/token",
  "introspection_endpoint": "https://server.example.com/introspect",
  "revocation_endpoint": "https://server.example.com/revo",
  "jwks_uri": "https://server.example.com/jwks",
  "response_types_supported": ["code"],
  "response_modes_supported": ["fragment","query","form_post"],
  "grant_types_supported": ["authorization_code", "refresh_token"],
  "token_endpoint_auth_methods_supported":
                ["tls_client_auth","client_secret_basic","none"],
  "tls_client_certificate_bound_access_tokens": true
  "mtls_endpoint_aliases": {
    "token_endpoint": "https://mtls.example.com/token",
    "revocation_endpoint": "https://mtls.example.com/revo",
    "introspection_endpoint": "https://mtls.example.com/introspect"
  }
}
```

Figure 4: Example Authorization Server Metadata with Mutual TLS Endpoint Aliases

6. Implementation Considerations

6.1. Authorization Server

The authorization server needs to set up its TLS configuration appropriately for the OAuth client authentication methods it supports.

An authorization server that supports mutual TLS client authentication and other client authentication methods or public clients in parallel would make mutual TLS optional (i.e. allowing a handshake to continue after the server requests a client certificate but the client does not send one).

In order to support the Self-Signed Certificate method, the authorization server would configure the TLS stack in such a way that it does not verify whether the certificate presented by the client during the handshake is signed by a trusted CA certificate.

As described in Section 3, the authorization server binds the issued access token to the TLS client certificate, which means that it will only issue certificate-bound tokens for a certificate which the client has proven possession of the corresponding private key.

The authorization server may also consider hosting the token endpoint, and other endpoints requiring client authentication, on a separate host name or port in order to prevent unintended impact on the TLS behavior of its other endpoints, e.g. the authorization endpoint. As described in Section 5, it may further isolate any potential impact of the server requesting client certificates by offering a distinct set of endpoints on a separate host or port, which are aliases for the originals that a client intending to do mutual TLS will use in preference to the conventional endpoints.

6.2. Resource Server

OAuth divides the roles and responsibilities such that the resource server relies on the authorization server to perform client authentication and obtain resource owner (end-user) authorization. The resource server makes authorization decisions based on the access token presented by the client but does not directly authenticate the client per se. The the manner in which an access token is bound to the client certificate decouples it from the specific method that the client used to authenticate with the authorization server. Mutual TLS during protected resource access can therefore serve purely as a proof-of-possession mechanism. As such, it is not necessary for the resource server to validate the trust chain of the client's certificate in any of the methods defined in this document. The resource server would therefore configure the TLS stack in a way that

it does not verify whether the certificate presented by the client during the handshake is signed by a trusted CA certificate.

6.3. Certificate Expiration and Bound Access Tokens

As described in Section 3, an access token is bound to a specific client certificate, which means that the same certificate must be used for mutual TLS on protected resource access. It also implies that access tokens are invalidated when a client updates the certificate, which can be handled similar to expired access tokens where the client requests a new access token (typically with a refresh token) and retries the protected resource request.

6.4. Implicit Grant Unsupported

This document describes binding an access token to the client certificate presented on the TLS connection from the client to the authorization server's token endpoint, however, such binding of access tokens issued directly from the authorization endpoint via the implicit grant flow is explicitly out of scope. End users interact directly with the authorization endpoint using a web browser and the use of client certificates in user's browsers bring operational and usability issues, which make it undesirable to support certificate-bound access tokens issued in the implicit grant flow. Implementations wanting to employ certificate-bound access tokens should utilize grant types that involve the client making an access token request directly to the token endpoint (e.g. the authorization code and refresh token grant types).

6.5. TLS Termination

An authorization server or resource server MAY choose to terminate TLS connections at a load balancer, reverse proxy, or other network intermediary. How the client certificate metadata is securely communicated between the intermediary and the application server in this case is out of scope of this specification.

7. Security Considerations

7.1. Certificate-Bound Refresh Tokens

The OAuth 2.0 Authorization Framework [RFC6749] requires that an authorization server bind refresh tokens to the client to which they where issued and that confidential clients (those having established authentication credentials with the authorization server) authenticate to the AS when presenting a refresh token. As a result, refresh tokens are indirectly certificate-bound when issued to clients utilizing the "tls_client_auth" or

"self_signed_tls_client_auth" methods of client authentication. Section 4 describes certificate-bound refresh tokens issued to public clients (those without authentication credentials associated with the "client_id").

7.2. Certificate Thumbprint Binding

The binding between the certificate and access token specified in Section 3.1 uses a cryptographic hash of the certificate. It relies on the hash function having sufficient preimage and second-preimage resistance so as to make it computationally infeasible to find or create another certificate that produces to the same hash output value. The SHA-256 hash function was used because it meets the aforementioned requirement while being widely available. If, in the future, certificate thumbprints need to be computed using hash function(s) other than SHA-256, it is suggested that additional related JWT confirmation methods members be defined for that purpose and registered in the the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for JWT "cnf" member values.

7.3. TLS Versions and Best Practices

In the abstract this document is applicable with any TLS version supporting certificate-based client authentication. Both TLS 1.3 [RFC8446] and TLS 1.2 [RFC5246] are cited herein because, at the time of writing, 1.3 is the newest version while 1.2 is the most widely deployed. General implementation and security considerations for TLS, including version recommendations, can be found in [BCP195].

7.4. X.509 Certificate Spoofing

If the PKI method of client authentication is used, an attacker could try to impersonate a client using a certificate with the same subject (DN or SAN) but issued by a different CA, which the authorization server trusts. To cope with that threat, the authorization server SHOULD only accept as trust anchors a limited number of CAs whose certificate issuance policy meets its security requirements. There is an assumption then that the client and server agree on the set of trust anchors that the server uses to create and validate the certificate chain. Without this assumption the use of a subject to identify the client certificate would open the server up to certificate spoofing attacks.

7.5. X.509 Certificate Parsing and Validation Complexity

Parsing and validation of X.509 certificates and certificate chains is complex and implementation mistakes have previously exposed

security vulnerabilities. Complexities of validation include (but are not limited to) [CX5P] [DCW] [RFC5280]:

- checking of Basic Constraints, basic and extended Key Usage constraints, validity periods, and critical extensions;
- handling of null-terminator bytes and non-canonical string representations in subject names;
- handling of wildcard patterns in subject names;
- recursive verification of certificate chains and checking certificate revocation.

For these reasons, implementors SHOULD use an established and well-tested X.509 library (such as one used by an established TLS library) for validation of X.509 certificate chains and SHOULD NOT attempt to write their own X.509 certificate validation procedures.

8. Privacy Considerations

In TLS versions prior to 1.3, the client's certificate is sent unencrypted in the initial handshake and can potentially be used by third parties to monitor, track, and correlate client activity. This is likely of little concern for clients that act on behalf of a significant number of end-users because individual user activity will not be discernible amidst the client activity as a whole. However, clients that act on behalf of a single end-user, such as a native application on a mobile device, should use TLS version 1.3 whenever possible or consider the potential privacy implications of using mutual TLS on earlier versions.

9. IANA Considerations

9.1. JWT Confirmation Methods Registration

This specification requests registration of the following value in the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for JWT "cnf" member values established by [RFC7800].

- Confirmation Method Value: "x5t#S256"
- Confirmation Method Description: X.509 Certificate SHA-256 Thumbprint
- Change Controller: IESG
- Specification Document(s): Section 3.1 of [[this specification]]

9.2. Authorization Server Metadata Registration

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

- Metadata Name: "tls_client_certificate_bound_access_tokens"
- Metadata Description: Indicates authorization server support for mutual TLS client certificate-bound access tokens.
- Change Controller: IESG
- Specification Document(s): Section 3.3 of [[this specification]]

- Metadata Name: "mtls_endpoint_aliases"
- Metadata Description: JSON object containing alternative authorization server endpoints, which a client intending to do mutual TLS will use in preference to the conventional endpoints.
- Change Controller: IESG
- Specification Document(s): Section 5 of [[this specification]]

9.3. Token Endpoint Authentication Method Registration

This specification requests registration of the following value in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] established by [RFC7591].

- Token Endpoint Authentication Method Name: "tls_client_auth"
- Change Controller: IESG
- Specification Document(s): Section 2.1.1 of [[this specification]]

- Token Endpoint Authentication Method Name: "self_signed_tls_client_auth"
- Change Controller: IESG
- Specification Document(s): Section 2.2.1 of [[this specification]]

9.4. Token Introspection Response Registration

Proof-of-Possession Key Semantics for JSON Web Tokens [RFC7800] defined the "cnf" (confirmation) claim, which enables confirmation key information to be carried in a JWT. However, the same proof-of-possession semantics are also useful for introspected access tokens whereby the protected resource obtains the confirmation key data as meta-information of a token introspection response and uses that information in verifying proof-of-possession. Therefore this specification defines and registers proof-of-possession semantics for OAuth 2.0 Token Introspection [RFC7662] using the "cnf" structure. When included as a top-level member of an OAuth token introspection

response, "cnf" has the same semantics and format as the claim of the same name defined in [RFC7800]. While this specification only explicitly uses the "x5t#S256" confirmation method member (see Section 3.2), it needs to define and register the higher level "cnf" structure as an introspection response member in order to define and use the more specific certificate thumbprint confirmation method.

As such, this specification requests registration of the following value in the IANA "OAuth Token Introspection Response" registry [IANA.OAuth.Parameters] established by [RFC7662].

- Claim Name: "cnf"
- Claim Description: Confirmation
- Change Controller: IESG
- Specification Document(s): [RFC7800] and [[this specification]]

9.5. Dynamic Client Registration Metadata Registration

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

- Client Metadata Name: "tls_client_certificate_bound_access_tokens"
- Client Metadata Description: Indicates the client's intention to use mutual TLS client certificate-bound access tokens.
- Change Controller: IESG
- Specification Document(s): Section 3.4 of [[this specification]]

- Client Metadata Name: "tls_client_auth_subject_dn"
- Client Metadata Description: String value specifying the expected subject DN of the client certificate.
- Change Controller: IESG
- Specification Document(s): Section 2.1.2 of [[this specification]]

- Client Metadata Name: "tls_client_auth_san_dns"
- Client Metadata Description: String value specifying the expected dNSName SAN entry in the client certificate.
- Change Controller: IESG
- Specification Document(s): Section 2.1.2 of [[this specification]]

- Client Metadata Name: "tls_client_auth_san_uri"
- Client Metadata Description: String value specifying the expected uniformResourceIdentifier SAN entry in the client certificate.
- Change Controller: IESG
- Specification Document(s): Section 2.1.2 of [[this specification]]

- Client Metadata Name: "tls_client_auth_san_ip"
- Client Metadata Description: String value specifying the expected iPAddress SAN entry in the client certificate.
- Change Controller: IESG
- Specification Document(s): Section 2.1.2 of [[this specification]]

- Client Metadata Name: "tls_client_auth_san_email"
- Client Metadata Description: String value specifying the expected rfc822Name SAN entry in the client certificate.
- Change Controller: IESG
- Specification Document(s): Section 2.1.2 of [[this specification]]

10. References

10.1. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/bcp195>.

[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/info/rfc2119>.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, DOI 10.17487/RFC4514, June 2006, <https://www.rfc-editor.org/info/rfc4514>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

10.2. Informative References

[CX5P] Wong, D., "Common x509 certificate validation/creation pitfalls", September 2016, <https://www.cryptologie.net/article/374/common-x509-certificate-validationcreation-pitfalls>.

[DCW] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software", <http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf>.

[I-D.ietf-oauth-token-binding] Jones, M., Campbell, B., Bradley, J., and W. Denniss, "OAuth 2.0 Token Binding", draft-ietf-oauth-token-binding-06 (work in progress), March 2018.

[IANA.JWT.Claims] IANA, "JSON Web Token Claims", <http://www.iana.org/assignments/jwt>.

[IANA.OAuth.Parameters] IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

[OpenID.CIBA] Fernandez, G., Walter, F., Nennker, A., Tonge, D., and B. Campbell, "OpenID Connect Client Initiated Backchannel Authentication Flow - Core 1.0", January 2019, <https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4517] Legg, S., Ed., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, DOI 10.17487/RFC4517, June 2006, <https://www.rfc-editor.org/info/rfc4517>.

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

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

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

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

[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/info/rfc7591>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[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/info/rfc8174>.

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

Appendix A. Example "cnf" Claim, Certificate and JWK

For reference, an "x5t#S256" value and the X.509 Certificate from which it was calculated are provided in the following example figures. A JWK representation of the certificate's public key along with the "x5c" member is also provided.

```
"cnf":{"x5t#S256":"A4DtL2JmUMhAsvJj5tKyn64SqzmuXbMrJa0n761y5v0"}
```

Figure 5: x5t#S256 Confirmation Claim

```
-----BEGIN CERTIFICATE-----
MIIBBjCBrAIBAjAKBggqhkjOPQQDAjAPMQ0wCwYDVQQDDARtdGxzMB4XDTE4MTAx
ODEyMzcwOVoXDTIyMDUwMjEyMzcwOVowDzENMAsGA1UEAwwEbXRsczBZMBMGByqG
SM49AgEGCCqGSM49AwEHA0IABNcnyxwqV6hY8QnhxxzFQ03C7HKW9OylMbnQZjjJ
/Au08/coZwxS7LfA4vOLS9WuneIXhbGGWvsDSb0tH6IxLm8wCgYIKoZIzj0EAwID
SQAwRgIhAP0RC1E+vwJD/D1AGHGzuri+hlV/PpQEKTWUVeORWz83AiEA5x2eXZOV
bUlJSGQgjwD5vaUaKlLR50Q2DmFfQj1L+SY=
-----END CERTIFICATE-----
```

Figure 6: PEM Encoded Self-Signed Certificate

```
{
 "kty":"EC",
 "x":"1yfLHCpXqFjxCeHHHMVDTcLscpb07KUxudBmOMn8C7Q",
 "y":"8_coZwxS7LfA4vOLS9WuneIXhbGGWvsDSb0tH6IxLm8",
 "crv":"P-256",
 "x5c":[
  "MIIBBjCBrAIBAjAKBggqhkjOPQQDAjAPMQ0wCwYDVQQDDARtdGxzMB4XDTE4MTA
   xODEyMzcwOVoXDTIyMDUwMjEyMzcwOVowDzENMAsGA1UEAwwEbXRsczBZMBMGBy
   qGSM49AgEGCCqGSM49AwEHA0IABNcnyxwqV6hY8QnhxxzFQ03C7HKW9OylMbnQZ
   jjJ/Au08/coZwxS7LfA4vOLS9WuneIXhbGGWvsDSb0tH6IxLm8wCgYIKoZIzj0E
   AwIDSQAwRgIhAP0RC1E+vwJD/D1AGHGzuri+hlV/PpQEKTWUVeORWz83AiEA5x2
   eXZOVbUlJSGQgjwD5vaUaKlLR50Q2DmFfQj1L+SY="
  ]
 }
```

Figure 7: JSON Web Key

Appendix B. Relationship to Token Binding

OAuth 2.0 Token Binding [I-D.ietf-oauth-token-binding] enables the application of Token Binding to the various artifacts and tokens employed throughout OAuth. That includes binding of an access token to a Token Binding key, which bears some similarities in motivation and design to the mutual TLS client certificate-bound access tokens defined in this document. Both documents define what is often called a proof-of-possession security mechanism for access tokens, whereby a client must demonstrate possession of cryptographic keying material when accessing a protected resource. The details differ somewhat between the two documents but both have the authorization server bind the access token that it issues to an asymmetric key pair held by the client. The client then proves possession of the private key from that pair with respect to the TLS connection over which the protected resource is accessed.

Token Binding uses bare keys that are generated on the client, which avoids many of the difficulties of creating, distributing, and managing certificates used in this specification. However, at the time of writing, Token Binding is fairly new and there is relatively little support for it in available application development platforms and tooling. Until better support for the underlying core Token Binding specifications exists, practical implementations of OAuth 2.0 Token Binding are infeasible. Mutual TLS, on the other hand, has been around for some time and enjoys widespread support in web servers and development platforms. As a consequence, OAuth 2.0 Mutual TLS Client Authentication and Certificate-Bound Access Tokens can be built and deployed now using existing platforms and tools. In the future, the two specifications are likely to be deployed in parallel for solving similar problems in different environments. Authorization servers may even support both specifications simultaneously using different proof-of-possession mechanisms for tokens issued to different clients.

Appendix C. Acknowledgements

Scott "not Tomlinson" Tomilson and Matt Peterson were involved in design and development work on a mutual TLS OAuth client authentication implementation, which predates this document. Experience and learning from that work informed some of the content of this document.

This specification was developed within the OAuth Working Group under the chairmanship of Hannes Tschofenig and Rifaat Shekh-Yusef with Eric Rescorla and Benjamin Kaduk serving as Security Area Directors. Additionally, the following individuals contributed ideas, feedback, and wording that helped shape this specification: Vittorio Bertocci,

Sergey Beryozkin, Ralph Bragg, Sophie Bremer, Vladimir Dzhuvinov, Samuel Erdtman, Evan Gilman, Leif Johansson, Michael Jones, Phil Hunt, Benjamin Kaduk, Takahiko Kawasaki, Sean Leonard, Kepeng Li, Neil Madden, James Manger, Jim Manico, Nov Matake, Sascha Preibisch, Eric Rescorla, Justin Richer, Filip Skokan, Dave Tonge, and Hannes Tschofenig.

Appendix D. Document(s) History

[[to be removed by the RFC Editor before publication as an RFC]]

draft-ietf-oauth-mtls-14

- Editorial clarifications around there being only a single subject registered/configured per client for the tls_client_auth method.
- Add a brief explanation about how, with tls_client_auth and self_signed_tls_client_auth, refresh tokens are certificate-bound indirectly via the client authentication.
- Add mention of refresh tokens in the abstract.

draft-ietf-oauth-mtls-13

- Add an abstract protocol flow and diagram to serve as an overview of OAuth in general and baseline to describe the various ways in which the mechanisms defined herein are intended to be used.
- A little bit less of that German influence.
- Rework the TLS references a bit and, in the Terminology section, clean up the description of what messages are sent and verified in the handshake to do 'mutual TLS'.
- Move the explanation about "cnf" introspection registration into the IANA Considerations.
- Add CIBA as an informational reference and additional example of an OAuth extension that defines an endpoint that utilizes client authentication.
- Shorten a few of the section titles.
- Add new client metadata values to allow for the use of a SAN in the PKI MTLS client authentication method.
- Add privacy considerations attempting to discuss the implications of the client cert being sent in the clear in TLS 1.2.
- Changed the 'Certificate Bound Access Tokens Without Client Authentication' section to 'Public Clients and Certificate-Bound Tokens' and moved it up to be a top level section while adding discussion of binding refresh tokens for public clients.
- Reword/restructure the main PKI method section somewhat to (hopefully) improve readability.
- Reword/restructure the Self-Signed method section a bit to (hopefully) make it more comprehensible.

- Reword the AS and RS Implementation Considerations somewhat to (hopefully) improve readability.
- Clarify that the protected resource obtains the client certificate used for mutual TLS from its TLS implementation layer.
- Add Security Considerations section about the certificate thumbprint binding that includes the hash algorithm agility recommendation.
- Add an "mtls_endpoint_aliases" AS metadata parameter that is a JSON object containing alternative authorization server endpoints, which a client intending to do mutual TLS will use in preference to the conventional endpoints.
- Minor editorial updates.

draft-ietf-oauth-mtls-12

- Add an example certificate, JWK, and confirmation method claim.
- Minor editorial updates based on implementer feedback.
- Additional Acknowledgements.

draft-ietf-oauth-mtls-11

- Editorial updates.
- Mention/reference TLS 1.3 RFC8446 in the TLS Versions and Best Practices section.

draft-ietf-oauth-mtls-10

- Update draft-ietf-oauth-discovery reference to RFC8414

draft-ietf-oauth-mtls-09

- Change "single certificates" to "self-signed certificates" in the Abstract

draft-ietf-oauth-mtls-08

- Incorporate clarifications and editorial improvements from Justin Richer's WGLC review
- Drop the use of the "sender constrained" terminology per WGLC feedback from Neil Madden (including changing the metadata parameters from mutual_tls_sender_constrained_access_tokens to tls_client_certificate_bound_access_tokens)
- Add a new security considerations section on X.509 parsing and validation per WGLC feedback from Neil Madden and Benjamin Kaduk
- Note that a server can terminate TLS at a load balancer, reverse proxy, etc. but how the client certificate metadata is securely communicated to the backend is out of scope per WGLC feedback

- Note that revocation checking is at the discretion of the AS per WGLC feedback
- Editorial updates and clarifications
- Update draft-ietf-oauth-discovery reference to -10 and draft-ietf-oauth-token-binding to -06
- Add folks involved in WGLC feedback to the acknowledgements list

draft-ietf-oauth-mtls-07

- Update to use the boilerplate from RFC 8174

draft-ietf-oauth-mtls-06

- Add an appendix section describing the relationship of this document to OAuth Token Binding as requested during the the Singapore meeting https://datatracker.ietf.org/doc/minutes-100-oauth/
- Add an explicit note that the implicit flow is not supported for obtaining certificate bound access tokens as discussed at the Singapore meeting https://datatracker.ietf.org/doc/minutes-100-oauth/
- Add/incorporate text to the Security Considerations on Certificate Spoofing as suggested https://mailarchive.ietf.org/arch/msg/oauth/V26070X-6OtbVSeUz_7W2k94vCo
- Changed the title to be more descriptive
- Move the Security Considerations section to before the IANA Considerations
- Elaborated on certificate-bound access tokens a bit more in the Abstract
- Update draft-ietf-oauth-discovery reference to -08

draft-ietf-oauth-mtls-05

- Editorial fixes

draft-ietf-oauth-mtls-04

- Change the name of the 'Public Key method' to the more accurate 'Self-Signed Certificate method' and also change the associated authentication method metadata value to "self_signed_tls_client_auth".
- Removed the "tls_client_auth_root_dn" client metadata field as discussed in https://mailarchive.ietf.org/arch/msg/oauth/swDV2y0be6o8czGKQi1eJV-g8qc
- Update draft-ietf-oauth-discovery reference to -07
- Clarify that MTLS client authentication isn't exclusive to the token endpoint and can be used with other endpoints, e.g. RFC 7009 revocation and 7662 introspection, that utilize client

authentication as discussed in https://mailarchive.ietf.org/arch/msg/oauth/bZ6mft0G7D3ccebhOxnEYUv4puI

- Reorganize the document somewhat in an attempt to more clearly make a distinction between mTLS client authentication and certificate-bound access tokens as well as a more clear delineation between the two (PKI/Public key) methods for client authentication
- Editorial fixes and clarifications

draft-ietf-oauth-mtls-03

- Introduced metadata and client registration parameter to publish and request support for mutual TLS sender constrained access tokens
- Added description of two methods of binding the cert and client, PKI and Public Key.
- Indicated that the "tls_client_auth" authentication method is for the PKI method and introduced "pub_key_tls_client_auth" for the Public Key method
- Added implementation considerations, mainly regarding TLS stack configuration and trust chain validation, as well as how to to do binding of access tokens to a TLS client certificate for public clients, and considerations around certificate-bound access tokens
- Added new section to security considerations on cert spoofing
- Add text suggesting that a new cnf member be defined in the future, if hash function(s) other than SHA-256 need to be used for certificate thumbprints

draft-ietf-oauth-mtls-02

- Fixed editorial issue https://mailarchive.ietf.org/arch/msg/oauth/U46UMEh8XIOQnvXY9pHFq1MKPns
- Changed the title (hopefully "Mutual TLS Profile for OAuth 2.0" is better than "Mutual TLS Profiles for OAuth Clients").

draft-ietf-oauth-mtls-01

- Added more explicit details of using RFC 7662 token introspection with mutual TLS sender constrained access tokens.
- Added an IANA OAuth Token Introspection Response Registration request for "cnf".
- Specify that tls_client_auth_subject_dn and tls_client_auth_root_dn are RFC 4514 String Representation of Distinguished Names.
- Changed tls_client_auth_issuer_dn to tls_client_auth_root_dn.
- Changed the text in the Section 3 to not be specific about using a hash of the cert.

- Changed the abbreviated title to 'OAuth Mutual TLS' (previously was the acronym MTLSPOC).

draft-ietf-oauth-mtls-00

- Created the initial working group version from draft-campbell-oauth-mtls

draft-campbell-oauth-mtls-01

- Fix some typos.
- Add to the acknowledgements list.

draft-campbell-oauth-mtls-00

- Add a Mutual TLS sender constrained protected resource access method and a x5t#S256 cnf method for JWT access tokens (concepts taken in part from draft-sakimura-oauth-jpop-04).
- Fixed "token_endpoint_auth_methods_supported" to "token_endpoint_auth_method" for client metadata.
- Add "tls_client_auth_subject_dn" and "tls_client_auth_issuer_dn" client metadata parameters and mention using "jwks_uri" or "jwks".
- Say that the authentication method is determined by client policy regardless of whether the client was dynamically registered or statically configured.
- Expand acknowledgements to those that participated in discussions around draft-campbell-oauth-tls-client-auth-00
- Add Nat Sakimura and Torsten Lodderstedt to the author list.

draft-campbell-oauth-tls-client-auth-00

- Initial draft.

Authors' Addresses

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Yubico

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Nat Sakimura
Nomura Research Institute

Email: n-sakimura@nri.co.jp
URI: https://nat.sakimura.org/

Torsten Lodderstedt
YES.com AG

Email: torsten@lodderstedt.net