Internet Engineering Task Force (IETF)
Request for Comments: 8705
Category: Standards Track
ISSN: 2070-1721

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

February 2020

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

## 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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8705.

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.

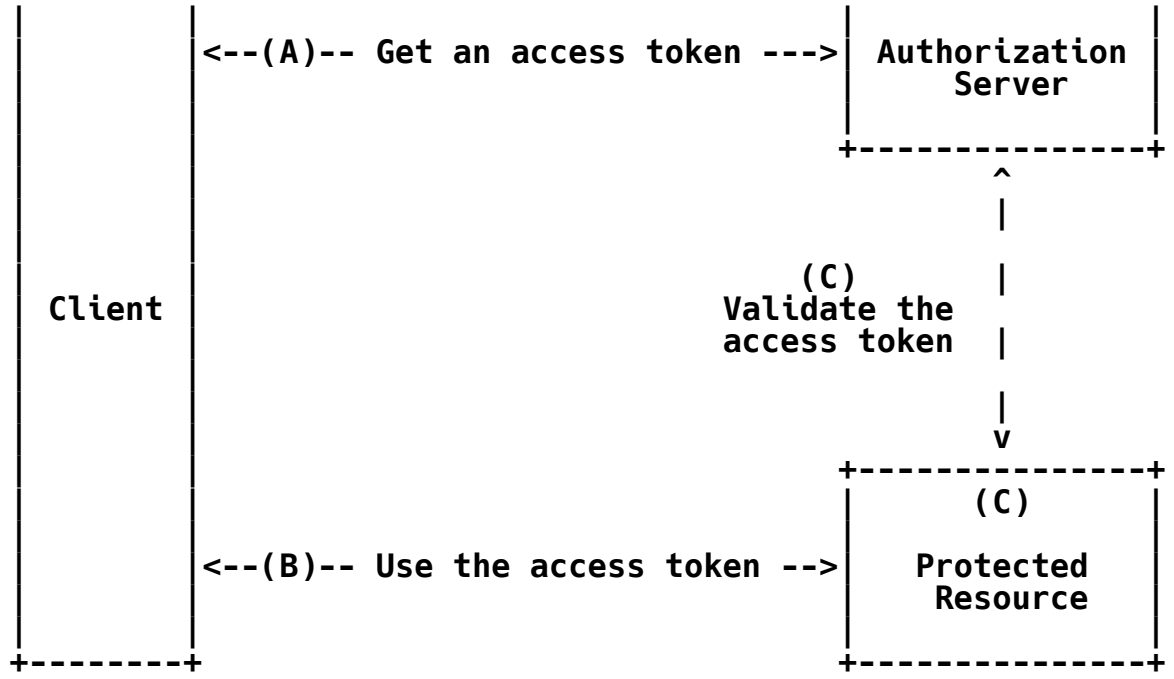


Figure 1: Abstract OAuth 2.0 Protocol Flow

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. Other cases require that the protected resource call out to the authorization server to determine the state of the token and obtain metainformation 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). Step (C) is supported with semantics to express the binding of the token to the client certificate for both local and remote processing in Sections 3.1 and 3.2, respectively. This ensures that, as described in Section 3, protected resource access in Step (B) is only possible by the legitimate client using a certificate-bound 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 defining and using 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 that 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 that are complementary but don't necessarily need to be deployed or used together.

Additional client metadata parameters are introduced by this document in support of certificate-bound access tokens and mutual-TLS client authentication. The authorization server can obtain client metadata via the Dynamic Client Registration Protocol [RFC7591], which defines mechanisms for dynamically registering OAuth 2.0 client metadata with authorization servers. Also the metadata defined by [RFC7591], and registered extensions to it, imply a general data model for clients that is useful for authorization server implementations, even when the Dynamic Client Registration Protocol isn't in play. Such implementations will typically have some sort of user interface available for managing client configuration.

## 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, in addition to the normal TLS server authentication with a certificate, 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 [RFC5246] [RFC8446], 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 Section 2.3 of OAuth 2.0 [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 re-established with mutual-TLS X.509 certificate authentication (i.e., the client Certificate and CertificateVerify 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 Section 2.2 of OAuth 2.0 [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 2.2 below. If no certificate is presented, or that which is presented doesn't match that which is expected for the given "client_id", the authorization server returns a normal OAuth 2.0 error response per Section 5.2 of [RFC6749] with the "invalid_client" error code to indicate failed client authentication.

### 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
: A string representation -- as defined in [RFC4514] -- of the expected subject distinguished name of the certificate that 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 that 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 that 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 [RFC5952]) that is expected to be present as an iPAddress SAN entry in the certificate that the OAuth client will use in mutual-TLS authentication. Per Section 8 of [RFC5952], the IP address comparison of the value in this parameter and the SAN entry in the certificate is to be done in binary format.

tls_client_auth_san_email
: A string containing the value of an expected rfc822Name SAN entry in the certificate that 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 a prerequisite, the client registers its X.509 certificates (using "jwks" defined in [RFC7591]) or a reference to 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 [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 Elliptic Curve (EC)) are required parameters per [RFC7518] so will be present even though they are not utilized in this context. Also note that Section 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.

In order for a resource server to use certificate-bound access tokens, it must have advance knowledge that mutual TLS is to be used for some or all resource accesses. In particular, the access token itself cannot be used as input to the decision of whether or not to request mutual TLS because (from the TLS perspective) it is "Application Data", only exchanged after the TLS handshake has been completed, and the initial CertificateRequest occurs during the handshake, before the Application Data is available. Although subsequent opportunities for a TLS client to present a certificate may be available, e.g., via TLS 1.2 renegotiation [RFC5246] or TLS 1.3 post-handshake authentication [RFC8446], this document makes no provision for their usage. It is expected to be common that a mutual-TLS-using resource server will require mutual TLS for all resources hosted thereupon or will serve mutual-TLS-protected and regular resources on separate hostname and port combinations, though other workflows are possible. How resource server policy is synchronized with the authorization server (AS) is out of scope for this document.

Within the scope of a mutual-TLS-protected resource-access flow, 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 Sections 3.3 and 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 [X690] 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. The new JWT content introduced by this specification is the "cnf" confirmation method claim at the bottom of the example that has the "x5t#S256" confirmation method member containing the value that is the hash of the client certificate to which the access token is bound.

```
{
  "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 metainformation 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 metainformation 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. The new introspection response content introduced by this specification is the "cnf" confirmation method at the bottom of the example that has the "x5t#S256" confirmation method member containing the value that is the hash of the client certificate to which the access token is bound.

```
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 [RFC8414] 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".

Note that if a client that has indicated the intention to use mutual-TLS client certificate-bound tokens makes a request to the token endpoint over a non-mutual-TLS connection, it is at the authorization server's discretion as to whether to return an error or issue an unbound token.

## 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 the 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 of 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 behavior 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 the 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 metadata 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 use 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 TLS "server_name" extension [RFC6066] 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 alone, 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 that 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 manner in which an access token is bound to the client certificate and how a protected resource verifies the proof-of-possession decouples that 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 similarly 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 that 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 (AS) bind refresh tokens to the client to which they were issued and that confidential clients (those having established authentication credentials with the AS) authenticate to the AS when presenting a refresh token. As a result, refresh tokens are indirectly certificate-bound by way of the client ID and the associated requirement for (certificate-based) authentication to the AS 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 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, for additional related JWT confirmation methods, members be defined for that purpose and registered in the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for JWT "cnf" member values.

Community knowledge about the strength of various algorithms and feasible attacks can change suddenly, and experience shows that a document about security is a point-in-time statement. Readers are advised to seek out any errata or updates that apply to this document.

## 7.3. TLS Versions and Best Practices

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].

TLS certificate validation (for both client and server certificates) requires a local database of trusted certificate authorities (CAs). Decisions about what CAs to trust and how to make such a determination of trust are out of scope for this document.

## 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 that 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 out of band 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 embedded NUL bytes in ASN.1 counted-length strings and non-canonical or non-normalized 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

Per this specification, the following value has been registered 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 RFC 8705

### 9.2. Authorization Server Metadata Registration

Per this specification, the following values have been registered 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 RFC 8705

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 RFC 8705

### 9.3. Token Endpoint Authentication Method Registration

Per this specification, the following values have been registered 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 RFC 8705

Token Endpoint Authentication Method Name: "self_signed_tls_client_auth"
Change Controller: IESG
Specification Document(s): Section 2.2.1 of RFC 8705

### 9.4. Token Introspection Response Registration

"Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" [RFC7800] defined the "cnf" (confirmation) claim that 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 metainformation 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, the following values have been registered 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 RFC 8705

### 9.5.  Dynamic Client Registration Metadata Registration

Per this specification, the following client metadata definitions have been registered 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 RFC 8705

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 RFC 8705

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 RFC 8705

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 RFC 8705

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 RFC 8705

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 RFC 8705

## 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, May 2015, <https://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>.

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

[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>.

[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>.

[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>.

[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 (NIST), "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[X690] ITU-T, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015.

## 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", DOI 10.1145/2382196.2382204, October 2012, <http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf>.

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

[IANA.OAuth.Parameters] IANA, "OAuth Parameters", <https://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", 16 January 2019, <https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html>.

[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>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[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>.

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

[TOKEN] Jones, M., Campbell, B., Bradley, J., and W. Denniss, "OAuth 2.0 Token Binding", Work in Progress, Internet-Draft, draft-ietf-oauth-token-binding-08, 19 October 2018, <https://tools.ietf.org/html/draft-ietf-oauth-token-binding-08>.

## 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 examples, Figures 5 and 6, respectively. A JWK representation of the certificate's public key along with the "x5c" member is also provided in Figure 7.

```
"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 [TOKEN] 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.

## Acknowledgements

Scott "not Tomlinson" Tomilson and Matt Peterson were involved in design and development work on a mutual-TLS OAuth client authentication implementation that 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, Benjamin Kaduk, and Roman Danyliw 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, Roman Danyliw, 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, Vincent Roca, Filip Skokan, Dave Tonge, and Hannes Tschofenig.

## 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