OAuth Working Group B. Campbell
Internet-Draft Ping Identity
Intended status: Standards Track J. Bradley
Expires: April 21, 2019 Yubico
N. Sakimura
Nomura Research Institute
T. Lodderstedt
YES.com AG
October 18, 2018

OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

draft-ietf-oauth-mtls-12

Abstract

This document describes OAuth client authentication and certificate bound access 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

Table of Contents

1. Introduction

This document describes OAuth client authentication and certificate bound access tokens using mutual TLS [RFC5246] authentication with X.509 certificates. OAuth clients are provided mechanisms for authentication to the authorization server using mutual TLS. 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.

The OAuth 2.0 Authorization Framework [RFC6749] 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] and Revocation [RFC7009]) 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

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 TLS 1.2 [RFC5246] this requires the client to send Client Certificate and Certificate Verify messages during the TLS handshake and for the server to verify these 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 [RFC5246]).

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 of a certificate to a specific client as described in either Section 2.1 or Section 2.2 below.

2.1. PKI Mutual TLS OAuth Client Authentication Method

The PKI (public key infrastructure) method of mutual TLS OAuth client authentication uses a subject distinguished name (DN) and validated certificate chain to identify the 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 expected DN 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). If and how to check a certificate's revocation status is a deployment decision at the discretion of the authorization server. The PKI method facilitates the way X.509 certificates are traditionally being used for authentication. It also allows the client to rotate its X.509 certificates without the need to modify its respective authentication data at the authorization server by obtaining a new certificate with the same subject DN from a trusted certificate authority (CA).

2.1.1. PKI Authentication 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

The following metadata parameter is introduced for the OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] in support of the PKI method of binding a certificate to a 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.

2.2. Self-Signed Certificate Mutual TLS OAuth Client Authentication 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 an X.509 certificate or a trusted source for its X.509 certificates (such as the "jwks_uri" defined in [RFC7591] that references a JSON Web Key [RFC7517] Set containing the client's certificates and public keys) 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 subject public key info of the certificate matches the subject public key info of one of the certificates configured or registered for that particular client. The Self-Signed Certificate method allows to use 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 Certificate Authentication 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 to 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 and public keys, where the X.509 certificates are represented using the JSON Web Key (JWK) [RFC7517] "x5c" parameter. Note that Sec 4.7 of [RFC7517] requires that the key in the first certificate of the "x5c" parameter must match the public key represented by other members of the JWK (e.g. "n" and "e" for RSA, "x" and "y" for EC).

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 the client certificate used for mutual TLS authentication 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. X.509 Certificate Thumbprint Confirmation Method for JWT

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

If, in the future, certificate thumbprints need to be computed using hash functions other than SHA-256, it is suggested that additional related JWT confirmation methods members be defined for that purpose. For example, a new "x5t#S512" (X.509 Certificate Thumbprint using SHA-512) confirmation method member could be defined by registering it in the the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for JWT "cnf" member values established by [RFC7800].

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 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.

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, it needed 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.

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 2: 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. Implementation Considerations

4.1. Authorization Server

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

If the authorization server wants to support mutual TLS client authentication and other client authentication methods in parallel, it should make mutual TLS optional.

If the authorization server supports the Self-Signed Certificate method, it should 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.

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.

4.2. Resource Server

Since the resource server relies on the authorization server to perform client authentication, there is no need for the resource server to validate the trust chain of the client's certificate in any of the methods defined in this document. Mutual TLS is used only as a proof-of-possession mechanism during protected resource access. The resource server should 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.

4.3. Certificate Bound Access Tokens Without Client Authentication

Mutual TLS OAuth client authentication and mutual TLS client 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 or clients utilizing other methods of authentication to the authorization server. 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, 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.

4.4. Certificate 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.

4.5. 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).

4.6. 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.

5. Security Considerations

5.1. TLS Versions and Best Practices

TLS 1.2 [RFC5246] is cited in this document because, at the time of writing, it is the latest version that is widely deployed. However, this document is applicable with other TLS versions supporting certificate-based client authentication, including the relatively recently published TLS 1.3 [RFC8446]. Implementation security considerations for TLS, including version recommendations, can be found in Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) [BCP195].

5.2. 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 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 DN to identify the client certificate would open the server up to certificate spoofing attacks.

5.3. 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) [X509Pitfalls] [DangerousCode] [RFC5280]:

- o checking of Basic Constraints, basic and extended Key Usage constraints, validity periods, and critical extensions;

- o 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.

6. IANA Considerations

6.1. 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]]

6.2. OAuth 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]]

6.3. 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]]

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

6.4. OAuth Token Introspection Response Registration

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

- o Claim Name: "cnf"
- o Claim Description: Confirmation
- o Change Controller: IESG
- o Specification Document(s): Section 3.2 of [[this specification]]

6.5. OAuth 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]:

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

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

7. References

7.1. 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>.

[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>.

7.2. Informative References

[DangerousCode]
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>.

[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>.

[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>.

[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>.

[X509Pitfalls] Wong, D., "Common x509 certificate validation/creation pitfalls", September 2016, <https://www.cryptologie.net/article/374/common-x509-certificate-validationcreation-pitfalls>.

Appendix A. Example Certificate, JSON Web Key, and Confirmation Method

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 3: x5t#S256 Confirmation Claim

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

Figure 4: 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 5: 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: Sergey Beryozkin, Sophie Bremer, Vladimir Dzhuvinov, Samuel Erdtman, Leif Johansson, Michael Jones, Phil Hunt, Benjamin Kaduk, Takahiko Kawasaki, Sean Leonard, Kepeng Li, Neil Madden, James Manger, Jim Manico, Nov Matake, Sascha Preibisch, 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-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

- o Incorporate clarifications and editorial improvements from Justin Richer's WGLC review
- o 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)
- o Add a new security considerations section on X.509 parsing and validation per WGLC feedback from Neil Madden and Benjamin Kaduk
- o 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
- o Note that revocation checking is at the discretion of the AS per WGLC feedback
- o Editorial updates and clarifications
- o Update draft-ietf-oauth-discovery reference to -10 and draft-ietf-oauth-token-binding to -06
- o Add folks involved in WGLC feedback to the acknowledgements list

draft-ietf-oauth-mtls-07

- o Update to use the boilerplate from RFC 8174

draft-ietf-oauth-mtls-06

- o 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/
- o 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/
- o Add/incorporate text to the Security Considerations on Certificate Spoofing as suggested https://mailarchive.ietf.org/arch/msg/oauth/V26070X-6OtbVSeUz_7W2k94vCo
- o Changed the title to be more descriptive
- o Move the Security Considerations section to before the IANA Considerations
- o Elaborated on certificate bound access tokens a bit more in the Abstract
- o Update draft-ietf-oauth-discovery reference to -08

draft-ietf-oauth-mtls-05

- o Editorial fixes

draft-ietf-oauth-mtls-04

- o 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".
- o Removed the "tls_client_auth_root_dn" client metadata field as discussed in https://mailarchive.ietf.org/arch/msg/oauth/swDV2y0be6o8czGKQi1eJV-g8qc
- o Update draft-ietf-oauth-discovery reference to -07
- o 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
- o 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
- o Editorial fixes and clarifications

draft-ietf-oauth-mtls-03

- o Introduced metadata and client registration parameter to publish and request support for mutual TLS sender constrained access tokens
- o Added description of two methods of binding the cert and client, PKI and Public Key.
- o 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
- o 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
- o Added new section to security considerations on cert spoofing
- o 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

- o Fixed editorial issue https://mailarchive.ietf.org/arch/msg/oauth/U46UMEh8XIOQnvXY9pHFq1MKPns
- o 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

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

draft-ietf-oauth-mtls-00

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

draft-campbell-oauth-mtls-01

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

draft-campbell-oauth-mtls-00

o 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).
o Fixed "token_endpoint_auth_methods_supported" to "token_endpoint_auth_method" for client metadata.
o Add "tls_client_auth_subject_dn" and "tls_client_auth_issuer_dn" client metadata parameters and mention using "jwks_uri" or "jwks".
o Say that the authentication method is determined by client policy regardless of whether the client was dynamically registered or statically configured.
o Expand acknowledgements to those that participated in discussions around draft-campbell-oauth-tls-client-auth-00
o Add Nat Sakimura and Torsten Lodderstedt to the author list.

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

o 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