OAuth Working Group M. Jones
Internet-Draft Microsoft
Intended status: Standards Track B. Campbell
Expires: April 22, 2019 Ping Identity
J. Bradley
Yubico
W. Denniss
Google
October 19, 2018

OAuth 2.0 Token Binding
draft-ietf-oauth-token-binding-08

Abstract

This specification enables OAuth 2.0 implementations to apply Token Binding to Access Tokens, Authorization Codes, Refresh Tokens, JWT Authorization Grants, and JWT Client Authentication. This cryptographically binds these tokens to a client's Token Binding key pair, possession of which is proven on the TLS connections over which the tokens are intended to be used. This use of Token Binding protects these tokens from man-in-the-middle and token export and replay attacks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This specification enables OAuth 2.0 [RFC6749] implementations to apply Token Binding (TLS Extension for Token Binding Protocol Negotiation [RFC8472], The Token Binding Protocol Version 1.0 [RFC8471] and Token Binding over HTTP [RFC8473]) to Access Tokens, Authorization Codes, Refresh Tokens, JWT Authorization Grants, and JWT Client Authentication. This cryptographically binds these tokens to a client's Token Binding key pair, possession of which is proven on the TLS connections over which the tokens are intended to be used. This use of Token Binding protects these tokens from man-in-the-middle and token export and replay attacks.

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

This specification uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Server", "Client", "Protected Resource", "Refresh Token", and "Token Endpoint" defined by OAuth 2.0 [RFC6749], the terms "Claim" and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [JWT], the term "User Agent" defined by RFC 7230 [RFC7230], and the terms "Provided", "Referred", "Token

Binding" and "Token Binding ID" defined by Token Binding over HTTP [RFC8473].

2. Token Binding for Refresh Tokens

Token Binding of refresh tokens is a straightforward first-party scenario, applying term "first-party" as used in Token Binding over HTTP [RFC8473]. It cryptographically binds the refresh token to the client's Token Binding key pair, possession of which is proven on the TLS connections between the client and the token endpoint. This case is straightforward because the refresh token is both retrieved by the client from the token endpoint and sent by the client to the token endpoint. Unlike the federation use cases described in Token Binding over HTTP [RFC8473], Section 4, and the access token case described in the next section, only a single TLS connection is involved in the refresh token case.

Token Binding a refresh token requires that the authorization server do two things. First, when refresh token is sent to the client, the authorization server needs to remember the Provided Token Binding ID and remember its association with the issued refresh token. Second, when a token request containing a refresh token is received at the token endpoint, the authorization server needs to verify that the Provided Token Binding ID for the request matches the remembered Token Binding ID associated with the refresh token. If the Token Binding IDs do not match, the authorization server should return an error in response to the request.

How the authorization server remembers the association between the refresh token and the Token Binding ID is an implementation detail that beyond the scope of this specification. Some authorization servers will choose to store the Token Binding ID (or a cryptographic hash of it, such a SHA-256 hash [SHS]) in the refresh token itself, provided it is integrity-protected, thus reducing the amount of state to be kept by the server. Other authorization servers will add the Token Binding ID value (or a hash of it) to an internal data structure also containing other information about the refresh token, such as grant type information. These choices make no difference to the client, since the refresh token is opaque to it.

2.1. Example Token Binding for Refresh Tokens

This section provides an example of what the interactions around a Token Bound refresh token might look like, along with some details of the involved processing. Token Binding of refresh tokens is most useful for native application clients so the example has protocol elements typical of a native client flow. Extra line breaks in all examples are for display purposes only.

A native application client makes the following access token request with an authorization code using a TLS connection where Token Binding has been negotiated. A PKCE "code_verifier" is included because use of PKCE is considered best practice for native application clients [BCP212]. The base64url-encoded representation of the exported keying material (EKM) from that TLS connection is "p6ZuSwfl6pIe8es5KyeV76T4swZmQp0_awd27jHfrbo", which is needed to validate the Token Binding Message.

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AIkAAgBBQGto7hHRR0Y5nkOWqc9KNfwW95dEFmSI_tCZ_Cbl
  7LWlt6Xjp3DbjiDJavGFiKP2HV_2JSE42VzmKOVVV8m7eqAAQOKiDK1Oi0z6v4X5B
  P7uc0pFestVZ42TTOdJmoHpji06Qq3jsCiCRSJx9ck2fWJYx8tLVXRZPATB3x6c24
  aY0ZEAAA

grant_type=authorization_code&code=4bwcZesc7Xacc330ltc66Wxk8EAfP9j2
  &code_verifier=2x6_ylS390-8V7jaT9wj.8qP9nKmYCf.V-rD9O4r_1
  &client_id=example-native-client-id
```

Figure 1: Initial Request with Code

A refresh token is issued in response to the prior request. Although it looks like a typical response to the client, the authorization server has bound the refresh token to the Provided Token Binding ID from the encoded Token Binding message in the "Sec-Token-Binding" header of the request. In this example, that binding is done by saving the Token Binding ID alongside other information about the refresh token in some server side persistent storage. The base64url-encoded representation of that Token Binding ID is "AgBBQGto7hHRR0Y5nkOWqc9KNfwW95dEFmSI_tCZ_Cbl7LWlt6Xjp3DbjiDJavGFiKP2HV_2JSE42VzmKOVVV8m7eqA".

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"EdRs7qMrLb167Z9fV2dcwoLTC",
 "refresh_token":"ACClZEIQTjW9arT9GOJGGd7QNwqOMmUYfsJTiv8his4",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Figure 2: Successful Response

When the access token expires, the client requests a new one with a refresh request to the token endpoint. In this example, the request is made on a new TLS connection so the EKM (base64url-encoded: "va-84Ukw4Zqfd7uWOtFrAJda96WwgbdaPDX2knoOiAE") and signature in the Token Binding Message are different than in the initial request.

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AIkAAgBBQGto7hHRR0Y5nkOWqc9KNfwW95dEFmSI_tCZ_Cbl
  7LWlt6Xjp3DbjiDJavGFiKP2HV_2JSE42VzmKOVVV8m7eqAAQCpGbaG_YRf27qOra
  L0UT4fsKKjL6PukuOT00qzamoAXxOq7m_id7O3mLpnb_sM7kwSxLi7iNHzzDgCAkP
  t3lHwAAA

refresh_token=ACClZEIQTjW9arT9GOJGGd7QNwqOMmUYfsJTiv8his4
  &grant_type=refresh_token&client_id=example-native-client-id
```

Figure 3: Refresh Request

However, because the Token Binding ID is long-lived and may span multiple TLS sessions and connections, it is the same as in the initial request. That Token Binding ID is what the refresh token is bound to, so the authorization server is able to verify it and issue a new access token.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"bwcESCwC4yOCQ8iPsgcn117k7",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Figure 4: Successful Response

3. Token Binding for Access Tokens

Token Binding for access tokens cryptographically binds the access token to the client’s Token Binding key pair, possession of which is proven on the TLS connections between the client and the protected resource. Token Binding is applied to access tokens in a similar manner to that described in Token Binding over HTTP [RFC8473], Section 4 (Federation Use Cases). It also builds upon the mechanisms for Token Binding of ID Tokens defined in OpenID Connect Token Bound Authentication 1.0 [OpenID.TokenBinding].

In the OpenID Connect [OpenID.Core] use case, HTTP redirects are used to pass information between the identity provider and the relying party; this HTTP redirect makes the Token Binding ID of the relying party available to the identity provider as the Referred Token Binding ID, information about which is then added to the ID Token. No such redirect occurs between the authorization server and the protected resource in the access token case; therefore, information about the Token Binding ID for the TLS connection between the client and the protected resource needs to be explicitly communicated by the client to the authorization server to achieve Token Binding of the access token.

This information is passed to the authorization server using the Referred Token Binding ID, just as in the ID Token case. The only difference is that the client needs to explicitly communicate the Token Binding ID of the TLS connection between the client and the protected resource to the Token Binding implementation so that it is sent as the Referred Token Binding ID in the request to the authorization server. This functionality provided by Token Binding implementations is described in Implementation Considerations of Token Binding over HTTP [RFC8473], Section 6.

Note that to obtain this Token Binding ID, the client may need to establish a TLS connection between itself and the protected resource prior to making the request to the authorization server so that the Provided Token Binding ID for the TLS connection to the protected resource can be obtained. How the client retrieves this Token Binding ID from the underlying Token Binding API is implementation and operating system specific. An alternative, if supported, is for the client to generate a Token Binding key to use for the protected resource, use the Token Binding ID for that key, and then later use that key when the TLS connection to the protected resource is established.

3.1. Access Tokens Issued from the Authorization Endpoint

For access tokens returned directly from the authorization endpoint, such as with the implicit grant defined in OAuth 2.0 [RFC6749], Section 4.2, the Token Binding ID of the client’s TLS channel to the protected resource is sent with the authorization request as the Referred Token Binding ID in the "Sec-Token-Binding" header, and is used to Token Bind the access token.

Upon receiving the Referred Token Binding ID in an authorization request, the authorization server associates (Token Binds) the ID with the access token in a way that can be accessed by the protected resource. Such methods include embedding the Referred Token Binding ID (or a cryptographic hash of it) in the issued access token itself, possibly using the syntax described in Section 3.4, or through token introspection as described in Section 3.5. The method for associating the referred token binding ID with the access token is determined by the authorization server and the protected resource, and is beyond the scope for this specification.

3.1.1. Example Access Token Issued from the Authorization Endpoint

This section provides an example of what the interactions around a Token Bound access token issued from the authorization endpoint might look like, along with some details of the involved processing. Extra line breaks in all examples are for display purposes only.

The client directs the user-agent to make the following HTTP request to the authorization endpoint. It is a typical authorization request that, because Token Binding was negotiated on the underlying TLS connection and the user-agent was signaled to reveal the Referred Token Binding, also includes the "Sec-Token-Binding" header with a Token Binding Message that contains both a Provided and Referred Token Binding. The base64url-encoded EKM from the TLS connection over which the request was made is "jI5UAyjs5XCPISUGQIwgcSrOiVIWq4fhLVIFTQ4nLxc".

```
GET /as/authorization.oauth2?response_type=token
  &client_id=example-client-id&state=rM8pZxG1c3gKy6rEbsD8s
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb HTTP/1.1
Host: server.example.com
Sec-Token-Binding: ARIAAgBBQIEE8mSMtDy2dj9EEBdXaQT9W3Rq1NS-jW8ebPoF
  6FyL0jIfATVE55zlircgOTZmEg1xeIrC3DsGegwjs4bhw14AQGKDlAXFFMyQkZegC
  wlbTlqX3F9HTt-lJxFU_pi16ezka7qVRCpSF0BQLfSqlsxMbYfSSCJX1BDtrIL7PX
  j__fUAAAECAEFA1BNUnP3te5WrwlEwiejEz0OpesmC5PElWc7kZ5nlLSqQTj1ciIp
  5vQ30LLUCyM_a2BYTUPKtd5EdS-PalT4t6ABADgeizRa5NkTMuX4zOdC-R4cLNWVV
  O8lLu2Psko-UJLR_XAH4Q0H7-m0_nQR1zBN78nYMKPvHsz8L3zWKRVyXEgAA
```

Figure 5: Authorization Request

The authorization server issues an access token and delivers it to the client by redirecting the user-agent with the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb#state=rM8pZxG1c3gKy6rEbsD8s
  &expires_in=3600&token_type=Bearer
  &access_token=eyJhbGciOiJFUzI[...omitted for brevity...]8xy5W5sQ
```

Figure 6: Authorization Response

The access token is bound to the Referred Token Binding ID from the authorization request, which when represented as a JWT, as described in Section 3.4, contains the SHA-256 hash of the Token Binding ID as the value of the "tbh" (token binding hash) member of the "cnf" (confirmation) claim. The confirmation claim portion of the JWT Claims Set is shown in the following figure.

```
{
  ...other claims omitted for brevity...
  "cnf":{
     "tbh": "vowQESa_MgbGJwIXaFm_BTN2QDPwh8PhuBm-EtUAqxc"
  }
}
```

Figure 7: Confirmation Claim

3.2. Access Tokens Issued from the Token Endpoint

For access tokens returned from the token endpoint, the Token Binding ID of the client’s TLS channel to the protected resource is sent as the Referred Token Binding ID in the "Sec-Token-Binding" header, and is used to Token Bind the access token. This applies to all the grant types from OAuth 2.0 [RFC6749] using the token endpoint, including, but not limited to the refresh and authorization code token requests, as well as some extension grants, such as JWT assertion authorization grants [RFC7523].

Upon receiving the Referred Token Binding ID in a token request, the authorization server associates (Token Binds) the ID with the access token in a way that can be accessed by the protected resource. Such methods include embedding the Referred Token Binding ID (or a cryptographic hash of it) in the issued access token itself, possibly using the syntax described in Section 3.4, or through token introspection as described in Section 3.5. The method for associating the referred token binding ID with the access token is determined by the authorization server and the protected resource, and is beyond the scope for this specification.

Note that if the request results in a new refresh token being generated, it can be Token bound using the Provided Token Binding ID, per Section 2.

3.2.1. Example Access Token Issued from the Token Endpoint

This section provides an example of what the interactions around a Token Bound access token issued from the token endpoint might look like, along with some details of the involved processing. Extra line breaks in all examples are for display purposes only.

The client makes an access token request to the token endpoint and includes the "Sec-Token-Binding" header with a Token Binding Message that contains both Provided and Referred Token Binding IDs. The Provided Token Binding ID is used to validate the token binding of the refresh token in the request (and to Token Bind a new refresh token, if one is issued), and the Referred Token Binding ID is used to Token Bind the access token that is generated. The base64url-encoded EKM from the TLS connection over which the access token request was made is "4jTc5e1QpocqPTZ5l6jsb6pRP18IFKdwwPvasYjn1-E".

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: ARIAAgBBQJFXJir2w4gbJ7grBx9uTYWIrs9V50-PW4ZijegQ
  0LUM-_bGnGT6DizxUK-m5n3dQUIkeH7ybn6wb1C5dGyV_IAAQDDFToFrHt41Zppq7
  u_SEMF_E-KimAB-HewWl2MvZzAQ9QKoWiJCLFiCkjgtr1RrA2-jaJvoB8o51DTGXQ
  ydWYkAAAECAEFAuC1GlYU83rqTGHEau1oqvNwy0fDsdXzIyT_4x1FcldsMxjFkJac
  IBJFGuYcccvnCak_duFi3QKFENuwxql-H9ABAMcU7IjJOUA4IyE6YoEcfz9BMPQqw
  M5M6hw4RZNQd58fsTCCslQE_NmNCl9JXy4NkdkEZBxqvZGPr0y8QZ_bmAwAA

refresh_token=gZR_ZI8EAhLgWR-gWxBimbgZRZi_8EAhLgWRgWxBimbf
 &grant_type=refresh_token&client_id=example-client-id
```

Figure 8: Access Token Request

The authorization server issues an access token bound to the Referred Token Binding ID and delivers it in a response the client.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"eyJhbGciOiJFUzI1NiIsImtp[...omitted...]1cs29j5c3",
 "token_type":"Bearer",
 "expires_in":3600
}
```

Figure 9: Response

The access token is bound to the Referred Token Binding ID of the access token request, which when represented as a JWT, as described in Section 3.4, contains the SHA-256 hash of the Token Binding ID as the value of the "tbh" (token binding hash) member of the "cnf" (confirmation) claim. The confirmation claim portion of the JWT Claims Set of the access token is shown in the following figure.

```
{
  ...other claims omitted for brevity...
  "cnf":{
     "tbh": "7NRBu9iDdJlYCTOqyeYuLxXv0blEA-yTpmGIrAwKAws"
  }
}
```

Figure 10: Confirmation Claim

3.3. Protected Resource Token Binding Validation

Upon receiving a token bound access token, the protected resource validates the binding by comparing the Provided Token Binding ID to the Token Binding ID for the access token. Alternatively, cryptographic hashes of these Token Binding ID values can be compared. If the values do not match, the resource access attempt MUST be rejected with an error.

3.3.1. Example Protected Resource Request

For example, a protected resource request using the access token from Section 3.2.1 would look something like the following. The base64url-encoded EKM from the TLS connection over which the request was made is "7LsNP3BT1aHHdXdk6meEWjtSkiPVLb7YS6iHp-JXmuE". The protected resource validates the binding by comparing the Provided Token Binding ID from the "Sec-Token-Binding" header to the token binding hash confirmation of the access token. Extra line breaks in the example are for display purposes only.

```
GET /api/stuff HTTP/1.1
Host: resource.example.org
Authorization: Bearer eyJhbGciOiJFUzI1NiIsI[...omitted...]1cs29j5c3
Sec-Token-Binding: AIkAAgBBQLgtRpWFPN66kxhxGrtaKrzcMtHw7HV8yMk_-MdR
  XJXbDMYxZCWnCASRRrmHHHL5wmpP3bhYt0ChRDbsMapfh_QAQN1He3Ftj4Wa_S_fz
  ZVns4saLfj6aBoMSQW6rLs19IIvHze7LrGjKyCfPTKXjajebxp-TLPFZCc0JTqTY5
  _0MBAAAA
```

Figure 11: Protected Resource Request

3.4. Representing Token Binding in JWT Access Tokens

If the access token is represented as a JWT, the token binding information SHOULD be represented in the same way that it is in token bound OpenID Connect ID Tokens [OpenID.TokenBinding]. That specification defines the new JWT Confirmation Method RFC 7800 [RFC7800] member "tbh" (token binding hash) to represent the SHA-256 hash of a Token Binding ID in an ID Token. The value of the "tbh" member is the base64url encoding of the SHA-256 hash of the Token

Binding ID. All trailing pad '=' characters are omitted from the encoded value and no line breaks, whitespace, or other additional characters are included.

The following example demonstrates the JWT Claims Set of an access token containing the base64url encoding of the SHA-256 hash of a Token Binding ID as the value of the "tbh" (token binding hash) element in the "cnf" (confirmation) claim:


```
{
 "iss": "https://server.example.com",
 "aud": "https://resource.example.org",
 "sub": "brian@example.com"
 "iat": 1467324320,
 "exp": 1467324920,
 "cnf":{
   "tbh": "7NRBu9iDdJlYCTOqyeYuLxXv0blEA-yTpmGIrAwKAws"
  }
}
```


Figure 12: JWT with Token Binding Hash Confirmation Claim

3.5. Representing Token Binding in Introspection Responses

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 token bound access token, the hash of the Token Binding ID 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 same structure as the token binding hash confirmation method, described in Section 3.4, as a top-level member of the introspection response JSON. The protected resource compares that token binding hash to a hash of the provided Token Binding ID and rejects the request, if they do not match.

The following is an example of an introspection response for an active token bound access token with a "tbh" token binding hash confirmation method.

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

{
  "active": true,
  "iss": "https://server.example.com",
  "aud": "https://resource.example.org",
  "sub": "brian@example.com"
  "iat": 1467324320,
  "exp": 1467324920,
  "cnf":{
    "tbh": "7NRBu9iDdJlYCTOqyeYuLxXv0blEA-yTpmGIrAwKAws"
  }
}
```

Figure 13: Example Introspection Response for a Token Bound Access Token

4. Token Binding Metadata

4.1. Token Binding Client Metadata

Clients supporting Token Binding that also support the OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] use these metadata values to declare their support for Token Binding of access tokens and refresh tokens:

client_access_token_token_binding_supported
: OPTIONAL. Boolean value specifying whether the client supports Token Binding of access tokens. If omitted, the default value is "false".

client_refresh_token_token_binding_supported
: OPTIONAL. Boolean value specifying whether the client supports Token Binding of refresh tokens. If omitted, the default value is "false". Authorization servers MUST NOT Token Bind refresh tokens issued to a client that does not support Token Binding of refresh tokens, but MAY reject requests completely from such clients if token binding is required by authorization server policy by returning an OAuth error response.

4.2. Token Binding Authorization Server Metadata

Authorization servers supporting Token Binding that also support OAuth 2.0 Authorization Server Metadata [RFC8414] use these metadata values to declare their support for Token Binding of access tokens and refresh tokens:

as_access_token_token_binding_supported
OPTIONAL. Boolean value specifying whether the authorization server supports Token Binding of access tokens. If omitted, the default value is "false".

as_refresh_token_token_binding_supported
OPTIONAL. Boolean value specifying whether the authorization server supports Token Binding of refresh tokens. If omitted, the default value is "false".

5. Token Binding for Authorization Codes

There are two variations for Token Binding of an authorization code. One is appropriate for native application clients and the other for web server clients. The nature of where the various components reside for the different client types demands different methods of Token Binding the authorization code so that it is bound to a Token Binding key on the end user’s device. This ensures that a lost or stolen authorization code cannot be successfully utilized from a different device. For native application clients, the code is bound to a Token Binding key pair that the native client itself possesses. For web server clients, the code is bound to a Token Binding key pair on the end user’s browser. Both variations utilize the extensible framework of Proof Key for Code Exchange (PKCE) [RFC7636], which enables the client to show possession of a certain key when exchanging the authorization code for tokens. The following subsections individually describe each of the two PKCE methods respectively.

5.1. Native Application Clients

This section describes a PKCE method suitable for native application clients that cryptographically binds the authorization code to a Token Binding key pair on the client, which the client proves possession of on the TLS connection during the access token request containing the authorization code. The authorization code is bound to the Token Binding ID that the native application client uses to resolve the authorization code at the token endpoint. This binding ensures that the client that made the authorization request is the same client that is presenting the authorization code.

5.1.1. Code Challenge

As defined in Proof Key for Code Exchange [RFC7636], the client sends the code challenge as part of the OAuth 2.0 authorization request with the two additional parameters: "code_challenge" and "code_challenge_method".

For this Token Binding method of PKCE, "TB-S256" is used as the value of the "code_challenge_method" parameter.

The value of the "code_challenge" parameter is the base64url encoding (per Section 5 of [RFC4648] with all trailing padding ('=') characters omitted and without the inclusion of any line breaks or whitespace) of the SHA-256 hash of the Provided Token Binding ID that the client will use when calling the authorization server's token endpoint. Note that, prior to making the authorization request, the client may need to establish a TLS connection between itself and the authorization server's token endpoint in order to establish the appropriate Token Binding ID.

When the authorization server issues the authorization code in the authorization response, it associates the code challenge and method values with the authorization code so they can be verified later when the authorization code is presented in the access token request.

5.1.1.1. Example Code Challenge

For example, a native application client sends an authorization request by sending the user's browser to the authorization endpoint. The resulting HTTP request looks something like the following (with extra line breaks for display purposes only).

```
GET /as/authorization.oauth2?response_type=code
  &client_id=example-native-client-id&state=oUC2jyYtzRCrMyWrVnGj
  &code_challenge=rBlgOyMY4teiuJMDgOwkrpsAjPyI07D2WsEM-dnq6eE
  &code_challenge_method=TB-S256 HTTP/1.1
Host: server.example.com
```

Figure 14: Authorization Request with PKCE Challenge

5.1.2. Code Verifier

Upon receipt of the authorization code, the client sends the access token request to the token endpoint. The Token Binding Protocol [RFC8471] is negotiated on the TLS connection between the client and the authorization server and the "Sec-Token-Binding" header, as defined in Token Binding over HTTP [RFC8473], is included in the access token request. The authorization server extracts the Provided Token Binding ID from the header value, hashes it with SHA-256, and compares it to the "code_challenge" value previously associated with the authorization code. If the values match, the token endpoint continues processing as normal (as defined by OAuth 2.0 [RFC6749]). If the values do not match, an error response indicating "invalid_grant" MUST be returned.

The "Sec-Token-Binding" header contains sufficient information for verification of the authorization code and its association to the original authorization request. However, PKCE [RFC7636] requires that a "code_verifier" parameter be sent with the access token request, so the static value "provided_tb" is used to meet that requirement and indicate that the Provided Token Binding ID is used for the verification.

5.1.2.1. Example Code Verifier

An example access token request, correlating to the authorization request in the previous example, to the token endpoint over a TLS connection for which Token Binding has been negotiated would look like the following (with extra line breaks for display purposes only). The base64url-encoded EKM from the TLS connection over which the request was made is "pNVKtPuQFvylNYn000QowWrQKoeMkeX9H32hVuU71Bs".

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AIkAAgBBQEOO9GRFP-LM0hoWw6-2i318BsuuUum5AL8bt1sz
  lr1EFfp5DMXMNW3O8WjcIXr2DKJnI4xnuGsE6GywQd9RbD0AQJDb3xyo9PBxj8M6Y
  jLt-6OaxgDkyoBoTkyrnNbLc8tJQ0JtXomKzBbj5qPtHDduXc6xz_lzvNpxSPxi42
  8m7wkAAA

grant_type=authorization_code&code=mJAReTWKX7zI3oHUNd4o3PeNqNqxKGp6
  &code_verifier=provided_tb&client_id=example-native-client-id
```

Figure 15: Token Request with PKCE Verifier

5.2. Web Server Clients

This section describes a PKCE method suitable for web server clients, which cryptographically binds the authorization code to a Token Binding key pair on the browser. The authorization code is bound to the Token Binding ID that the browser uses to deliver the authorization code to a web server client, which is sent to the authorization server as the Referred Token Binding ID during the authorization request. The web server client conveys the Token Binding ID to the authorization server when making the access token request containing the authorization code. This binding ensures that the authorization code cannot successfully be played or replayed to the web server client from a different browser than the one that made the authorization request.

5.2.1. Code Challenge

As defined in Proof Key for Code Exchange [RFC7636], the client sends the code challenge as part of the OAuth 2.0 Authorization Request with the two additional parameters: "code_challenge" and "code_challenge_method".

The client must send the authorization request through the browser such that the Token Binding ID established between the browser and itself is revealed to the authorization server’s authorization endpoint as the Referred Token Binding ID. Typically, this is done with an HTTP redirection response and the "Include-Referred-Token-Binding-ID" header, as defined in Token Binding over HTTP [RFC8473], Section 5.3.

For this Token Binding method of PKCE, "referred_tb" is used for the value of the "code_challenge_method" parameter.

The value of the "code_challenge" parameter is "referred_tb". The static value for the required PKCE parameter indicates that the authorization code is to be bound to the Referred Token Binding ID from the Token Binding Message sent in the "Sec-Token-Binding" header of the authorization request.

When the authorization server issues the authorization code in the authorization response, it associates the Token Binding ID (or hash thereof) and code challenge method with the authorization code so they can be verified later when the authorization code is presented in the access token request.

5.2.1.1. Example Code Challenge

For example, the web server client sends the authorization request by redirecting the browser to the authorization endpoint. That HTTP redirection response looks like the following (with extra line breaks for display purposes only).

```
HTTP/1.1 302 Found
Location: https://server.example.com?response_type=code
  &client_id=example-web-client-id&state=P4FUFqYzs1ij3ffsYCP34d3
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb
  &code_challenge=referred_tb&code_challenge_method=referred_tb
Include-Referred-Token-Binding-ID: true
```

Figure 16: Redirect the Browser

The redirect includes the "Include-Referred-Token-Binding-ID" response header field that signals to the user-agent that it should reveal, to the authorization server, the Token Binding ID used on the connection to the web server client. The resulting HTTP request to the authorization server looks something like the following (with extra line breaks for display purposes only). The base64url-encoded EKM from the TLS connection over which the request was made is "7gOdRzMhPeO-1YwZGmnVHyReN5vd2CxcsRBN69Ue4cI".

```
GET /as/authorization.oauth2?response_type=code
  &client_id=example-web-client-id&state=dryo8YFpWacbUPjhBf4Nvt51
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb
  &code_challenge=referred_tb
  &code_challenge_method=referred_tb HTTP/1.1
Host: server.example.com
Sec-Token-Binding: ARIAAgBBQB-XOPf5ePlf7ikATiAFEGOS503lPmRfkyymzdWw
  HCxl0njjxC3D0E_OVfBNqrIQxzIfkF7tWby2ZfyaE6XpwTsAQBYqhFX78vMOgDX_F
  d_b2dlHyHlMmkIz8iMVBY_reM98OUaJFz5IB7PG9nZ11j58LoG5QhmQoI9NXYktKZ
  RXxrYAAAECAEFAdUFTnfQADkn1uDbQnvJEk6oQs38L92gv-KO-qlYadLoDIKe2h53
  hSiKwIP98iRj_unedkNkAMyg9e2mY4Gp7WwBAeDUOwaSXNz1e6gKohwN4SAZ5eNyx
  45Mh8VI4woL1BipLoqrJRoK6dxFkWgHRMuBROcLGUj5PiOoxybQH_Tom3gAA
```

Figure 17: Authorization Request

5.2.2. Code Verifier

The web server client receives the authorization code from the browser and extracts the Provided Token Binding ID from the "Sec-Token-Binding" header of the request. The client sends the base64url-encoded (per Section 5 of [RFC4648] with all trailing padding ('=') characters omitted and without the inclusion of any line breaks or whitespace) Provided Token Binding ID as the value of the "code_verifier" parameter in the access token request to the authorization server's token endpoint. The authorization server compares the value of the "code_verifier" parameter to the Token Binding ID value previously associated with the authorization code. If the values match, the token endpoint continues processing as normal (as defined by OAuth 2.0 [RFC6749]). If the values do not match, an error response indicating "invalid_grant" MUST be returned.

5.2.2.1. Example Code Verifier

Continuing the example from the previous section, the authorization server sends the code to the web server client by redirecting the browser to the client's "redirect_uri", which results in the browser making a request like the following (with extra line breaks for display purposes only) to the web server client over a TLS channel for which Token Binding has been established. The base64url-encoded EKM from the TLS connection over which the request was made is "EzW60vyINbsb_tajt8ij3tV6cwy2KH-i8BdEMYXcNn0".

```
GET /cb?state=dryo8YFpWacbUPjhBf4Nvt51&code=jwD3oOa5cQvvLc81bwc4CMw
Host: client.example.org
Sec-Token-Binding: AIkAAgBBQHVBU530AA5J9bg20J7yRJOqELN_C_doL_ijvqpW
  GnS6AyCntoed4UoisCD_fIkY_7p3nZDZADMoPXtpmOBqe1sAQEwgC9Zpg7QFCDBib
  6GlZki3MhH32KNfLefLJc1vR1xE8l7OMfPLZHP2Woxh6rEtmgBcAABubEbTz7muNl
  Ln8uoAAA
```

Figure 18: Authorization Response to Web Server Client

The web server client takes the Provided Token Binding ID from the above request from the browser and sends it, base64url encoded, to the authorization server in the "code_verifier" parameter of the authorization code grant type request. Extra line breaks in the example request are for display purposes only.

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic b3JnLmV4YW1wbGUuY2xpZW50OmlldGY5OGNoaWNhZ28=

grant_type=authorization_code&code=jwD3oOa5cQvvLc81bwc4CMw
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Eorg%2Fcb
  &client_id=example-web-client-id
  &code_verifier=AgBBQHVBU530AA5J9bg20J7yRJOqELN_C_doL_ijv
  qpWGnS6AyCntoed4UoisCD_fIkY_7p3nZDZADMoPXtpmOBqe1s
```

Figure 19: Exchange Authorization Code

6. Token Binding JWT Authorization Grants and Client Authentication

The JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants [RFC7523] defines the use of bearer JWTs as a means for requesting an OAuth 2.0 access token as well as for client authentication. This section describes extensions to that specification enabling the application of Token Binding to JWT client authentication and JWT authorization grants.

6.1. JWT Format and Processing Requirements

In addition the requirements set forth in Section 3 of RFC 7523 [RFC7523], the following criteria must also be met for token bound JWTs used as authorization grants or for client authentication.

- The JWT MUST contain a "cnf" (confirmation) claim with a "tbh" (token binding hash) member identifying the Token Binding ID of the Provided Token Binding used by the client on the TLS connection to the authorization server. The authorization server MUST reject any JWT that has a token binding hash confirmation that does not match the corresponding hash of the Provided Token Binding ID from the "Sec-Token-Binding" header of the request.

6.2. Token Bound JWTs for Client Authentication

To use a token bound JWT for client authentication, the client uses the parameter values and encodings from Section 2.2 of RFC 7523 [RFC7523] with one exception: the value of the "client_assertion_type" is "urn:ietf:params:oauth:client-assertion-type:jwt-token-bound".

The "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] contains values, each of which specify a method of authenticating a client to the authorization server. The values are used to indicated supported and utilized client authentication methods in authorization server metadata, such as [OpenID.Discovery] and [RFC8414], and in OAuth 2.0 Dynamic Client Registration Protocol [RFC7591]. The values "private_key_jwt" and "client_secret_jwt" are designated by OpenID Connect [OpenID.Core] as authentication method values for bearer JWT client authentication using asymmetric and symmetric JWS [RFC7515] algorithms respectively. For Token Bound JWT for client authentication, this specification defines and registers the following authentication method values.

private_key_token_bound_jwt
: Indicates that client authentication to the authorization server will occur with a Token Bound JWT, which is signed with a client's private key.

client_secret_token_bound_jwt
: Indicates that client authentication to the authorization server will occur with a Token Bound JWT, which is integrity protected with a MAC using the octets of the UTF-8 representation of the client secret as the shared key.

Note that just as with the "private_key_jwt" and "client_secret_jwt" authentication methods, the "token_endpoint_auth_signing_alg" client registration parameter may be used to indicate the JWS algorithm used for signing the client authentication JWT for the authentication methods defined above.

6.3. Token Bound JWTs for as Authorization Grants

To use a token bound JWT for an authorization grant, the client uses the parameter values and encodings from Section 2.1 of RFC 7523 [RFC7523] with one exception: the value of the "grant_type" is "urn:ietf:params:oauth:grant-type:jwt-token-bound".

7. Security Considerations

7.1. Phasing in Token Binding

Many OAuth implementations will be deployed in situations in which not all participants support Token Binding. Any of combination of the client, the authorization server, the protected resource, and the user agent may not yet support Token Binding, in which case it will not work end-to-end.

It is a context-dependent deployment choice whether to allow interactions to proceed in which Token Binding is not supported or whether to treat the omission of Token Binding at any step as a fatal error. Particularly in dynamic deployment environments in which End Users have choices of clients, authorization servers, protected resources, and/or user agents, it is recommended that, for some reasonable period of time during which Token Binding technology is being adopted, authorizations using one or more components that do not implement Token Binding be allowed to successfully proceed. This enables different components to be upgraded to supporting Token Binding at different times, providing a smooth transition path for phasing in Token Binding. However, when Token Binding has been performed, any Token Binding key mismatches MUST be treated as fatal errors.

In more controlled deployment environments where the participants in an authorization interaction are known or expected to support Token Binding and yet one or more of them does not use it, the authorization SHOULD be aborted with an error. For instance, an authorization server should reject a token request that does not include the "Sec-Token-Binding" header, if the request is from a client known to support Token Binding (via configuration or the "client_access_token_token_binding_supported" metadata parameter).

7.2. Binding of Refresh Tokens

Section 6 of RFC 6749 [RFC6749] requires that a refresh token be bound to the client to which it was issued and that, if the client type is confidential or the client was issued client credentials (or assigned other authentication requirements), the client must authenticate with the authorization server when presenting the refresh token. As a result, for non-public clients, refresh tokens are indirectly bound to the client's credentials and cannot be used without the associated client authentication. Non-public clients then are afforded protections (equivalent to the strength of their authentication credentials) against unauthorized replay of refresh tokens and it is reasonable to not Token Bind refresh tokens for such clients while still Toking Binding the issued access tokens. Refresh tokens issued to public clients, however, do not have the benefit of such protections and authorization servers MAY elect to disallow public clients from registering or establishing configuration that would allow Token Bound access tokens but unbound refresh tokens.

Some web-based confidential clients implemented as distributed nodes may be perfectly capable of implementing access token binding (if the access token remains on the node it was bound to, the token binding keys would be locally available for that node to prove possession), but may struggle with refresh token binding due to an inability to share token binding key material between nodes. As confidential clients already have credentials which are required to use the refresh token, and those credentials should only ever be sent over TLS server-to-server between the client and the Token Endpoint, there is still value in token binding access tokens without token binding refresh tokens. Authorization servers SHOULD consider supporting access token binding without refresh token binding for confidential web clients as there are still security benefits to do so.

Clients MUST declare through dynamic (Section 4.1) or static registration information what types of token bound tokens they support to enable the server to bind tokens accordingly, taking into account any phase-in policies. Authorization servers MAY reject requests from any client who does not support token binding (by returning an OAuth error response) per their own security policies.

8. IANA Considerations

8.1. OAuth Dynamic Client Registration Metadata Registration

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

8.1.1. Registry Contents

- o Client Metadata Name: "client_access_token_token_binding_supported"
- o Client Metadata Description: Boolean value specifying whether the client supports Token Binding of access tokens
- o Change Controller: IESG
- o Specification Document(s): Section 4.1 of [[this specification]]

- o Client Metadata Name: "client_refresh_token_token_binding_supported"
- o Client Metadata Description: Boolean value specifying whether the client supports Token Binding of refresh tokens
- o Change Controller: IESG

- o Specification Document(s): Section 4.1 of [[this specification]]

8.2. OAuth Authorization Server Metadata Registration

This specification registers the following metadata definitions in the IANA "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414]:

8.2.1. Registry Contents

- o Metadata Name: "as_access_token_token_binding_supported"
- o Metadata Description: Boolean value specifying whether the authorization server supports Token Binding of access tokens
- o Change Controller: IESG
- o Specification Document(s): Section 4.2 of [[this specification]]

- o Metadata Name: "as_refresh_token_token_binding_supported"
- o Metadata Description: Boolean value specifying whether the authorization server supports Token Binding of refresh tokens
- o Change Controller: IESG
- o Specification Document(s): Section 4.2 of [[this specification]]

8.3. PKCE Code Challenge Method Registration

This specification requests registration of the following Code Challenge Method Parameter Names in the IANA "PKCE Code Challenge Methods" registry [IANA.OAuth.Parameters] established by [RFC7636].

8.3.1. Registry Contents

- o Code Challenge Method Parameter Name: TB-S256
- o Change controller: IESG
- o Specification document(s): Section 5.1.1 of [[this specification]]

- o Code Challenge Method Parameter Name: referred_tb
- o Change controller: IESG
- o Specification document(s): Section 5.2.1 of [[this specification]]

9. Token Endpoint Authentication Method Registration

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

9.1. Registry Contents

- Token Endpoint Authentication Method Name: "client_secret_token_bound_jwt"
- Change Controller: IESG
- Specification Document(s): Section 6 of [[this specification]]

- Token Endpoint Authentication Method Name: "private_key_token_bound_jwt"
- Change Controller: IESG
- Specification Document(s): Section 6 of [[this specification]]

10. Sub-Namespace Registrations

This specification requests registration of the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established in An IETF URN Sub-Namespace for OAuth [RFC6755].

10.1. Registry Contents

- URN: urn:ietf:params:oauth:grant-type:jwt-token-bound
- Common Name: Token Bound JWT Grant Type for OAuth 2.0
- Change controller: IESG
- Specification Document: Section 6 of [[this specification]]

- URN: urn:ietf:params:oauth:client-assertion-type:jwt-token-bound
- Common Name: Token Bound JWT for OAuth 2.0 Client Authentication
- Change controller: IESG
- Specification Document: Section 6 of [[this specification]]

11. References

11.1. Normative References

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

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://tools.ietf.org/html/rfc7519>.

[OpenID.TokenBinding]
Jones, M., Bradley, J., and B. Campbell, "OpenID Connect Token Bound Authentication 1.0", October 2017, <http://openid.net/specs/openid-connect-token-bound-authentication-1_0-03.html>.

[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>.

[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>.

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

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

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/info/rfc7636>.

[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>.

[RFC8471] Popov, A., Ed., Nystroem, M., Balfanz, D., and J. Hodges, "The Token Binding Protocol Version 1.0", RFC 8471, DOI 10.17487/RFC8471, October 2018, <https://www.rfc-editor.org/info/rfc8471>.

[RFC8472] Popov, A., Ed., Nystroem, M., and D. Balfanz, "Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation", RFC 8472, DOI 10.17487/RFC8472, October 2018, <https://www.rfc-editor.org/info/rfc8472>.

[RFC8473] Popov, A., Nystroem, M., Balfanz, D., Ed., Harper, N., and J. Hodges, "Token Binding over HTTP", RFC 8473, DOI 10.17487/RFC8473, October 2018, <https://www.rfc-editor.org/info/rfc8473>.

[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>.

11.2. Informative References

[BCP212] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", August 2015, <http://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", August 2015, <http://openid.net/specs/openid-connect-discovery-1_0.html>.

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

Appendix A. Acknowledgements

This specification was developed within the OAuth Working Group under the chairmanship of Hannes Tschofenig and Rifaat Shekh-Yusef with Kathleen Moriarty, Eric Rescorla, and Benjamin Kaduk serving as Security Area Directors. Additionally, the following individuals contributed ideas, feedback, and wording that helped shape this specification: Dirk Balfanz, Andrei Popov, Justin Richer, and Nat Sakimura.

Appendix B. Document History

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

-08

- o Update reference to -03 of openid-connect-token-bound-authentication.
- o Update the references to the core token binding specs, which are now RFCs 8471, 8472, and 8473.
- o Update reference to AS metadata, which is now RFC 8414.
- o Add chairs and ADs to the Acknowledgements.

-07

- o Explicitly state that the base64url encoding of the tbh value doesn't include any trailing pad characters, line breaks, whitespace, etc.
- o Update to latest references for tokbind drafts and draft-ietf-oauth-discovery.
- o Update reference to Implementation Considerations in draft-ietf-tokbind-https, which is section 6 rather than 5.
- o Try to tweak text that references specific sections in other documents so that the HTML generated by the ietf tools doesn't link to the current document (based on old suggestion from Barry https://www.ietf.org/mail-archive/web/jose/current/msg04571.html).

-06

- Use the boilerplate from RFC 8174.

- Update reference for draft-ietf-tokbind-https to -12 and draft-ietf-oauth-discovery to -09.

- Minor editorial fixes.

-05

- State that authorization servers should not token bind refresh tokens issued to a client that doesn’t support bound refresh tokens, which can be indicated by the "client_refresh_token_token_binding_supported" client metadata parameter.

- Add Token Binding for JWT Authorization Grants and JWT Client Authentication.

- Adjust the language around aborting authorizations in Phasing in Token Binding to be somewhat more general and not only about downgrades.

- Remove reference to, and usage of, ’OAuth 2.0 Protected Resource Metadata’, which is no longer a going concern.

- Moved "Token Binding Metadata" section before "Token Binding for Authorization Codes" to be closer to the "Token Binding for Access Tokens" and "Token Binding for Refresh Tokens", to which it is more closely related.

- Update references for draft-ietf-tokbind- negotiation(-10), protocol(-16), and https(-10), as well as draft-ietf-oauth-discovery(-07), and BCP212/RFC8252 OAuth 2.0 for Native Apps.

-04

- Define how to convey token binding information of an access token via RFC 7662 OAuth 2.0 Token Introspection (note that the Introspection Response Registration request for cnf/Confirmation is in https://tools.ietf.org/html/draft-ietf-oauth-mtls-02#section-4.3 which will likely be published and registered prior to this document).

- Minor editorial fixes.

- Added an open issue about needing to allow for web server clients to opt-out of having refresh tokens bound while still allowing for binding of access tokens (following from mention of the problem on slide 16 of the presentation from Chicago https://www.ietf.org/proceedings/98/slides/slides-98-oauth-sessb-token-binding-00.pdf).

-03

- Fix a few mistakes in and around the examples that were noticed preparing the slides for IETF 98 Chicago.

-02

- Added a section on Token Binding for authorization codes with one variation for native clients and one for web server clients.
- Updated language to reflect that the binding is to the token binding key pair and that proof-of-possession of that key is done on the TLS connection.
- Added a bunch of examples.
- Added a few Open Issues so they are tracked in the document.
- Updated the Token Binding and OAuth Metadata references.
- Added William Denniss as an author.

-01

- Changed Token Binding for access tokens to use the Referred Token Binding ID, now that the Implementation Considerations in the Token Binding HTTPS specification make it clear that implementations will enable using the Referred Token Binding ID.
- Defined Protected Resource Metadata value.
- Changed to use the more specific term "protected resource" instead of "resource server".

-00

- Created the initial working group version from draft-jones-oauth-token-binding-00.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Yubico

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

William Denniss
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
USA

Email: wdenniss@google.com
URI: http://wdenniss.com/