Abstract

This specification enables OAuth 2.0 implementations to apply Token Binding to Access Tokens, Authorization Codes, and Refresh Tokens. 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

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ............................................... 3
   1.1. Requirements Notation and Conventions .................. 3
   1.2. Terminology ........................................... 3
2. Token Binding for Refresh Tokens ............................ 3
   2.1. Example Token Binding for Refresh Tokens .............. 4
3. Token Binding for Access Tokens ............................ 6
   3.1. Access Tokens Issued from the Authorization Endpoint .. 7
      3.1.1. Example Access Token Issued from the Authorization
               Endpoint .......................................... 8
   3.2. Access Tokens Issued from the Token Endpoint .......... 9
      3.2.1. Example Access Token Issued from the Token Endpoint . 9
3.3. Protected Resource Token Binding Validation ............ 11
   3.3.1. Example Protected Resource Request .................. 11
3.4. Representing Token Binding in JWT Access Tokens .......... 11
3.5. Representing Token Binding in Introspection Responses ... 12
4. Token Binding for Authorization Codes .................... 13
   4.1. Native Application Clients ............................ 13
      4.1.1. Code Challenge .................................... 14
         4.1.1.1. Example Code Challenge ........................ 14
      4.1.2. Code Verifier ..................................... 14
         4.1.2.1. Example Code Verifier .......................... 15
   4.2. Web Server Clients .................................... 15
      4.2.1. Code Challenge .................................... 16
         4.2.1.1. Example Code Challenge ........................ 16
      4.2.2. Code Verifier ..................................... 17
         4.2.2.1. Example Code Verifier .......................... 18
5. Phasing in Token Binding and Preventing Downgrade Attacks .. 18
6. Token Binding Metadata ..................................... 19
   6.1. Token Binding Client Metadata ........................ 19
   6.2. Token Binding Authorization Server Metadata ........... 20
   6.3. Token Binding Protected Resource Metadata ............. 20
7. Security Considerations .................................... 20
8. IANA Considerations ........................................ 20
   8.1. OAuth Dynamic Client Registration Metadata Registration 20
      8.1.1. Registry Contents .................................. 21
   8.2. OAuth Authorization Server Metadata Registration ........ 21
      8.2.1. Registry Contents .................................. 21
   8.3. OAuth Protected Resource Metadata Registration ........ 21
      8.3.1. Registry Contents .................................. 21
1. Introduction

This specification enables OAuth 2.0 [RFC6749] implementations to apply Token Binding (TLS Extension for Token Binding Protocol Negotiation [I-D.ietf-tokbind-negotiation], The Token Binding Protocol Version 1.0 [I-D.ietf-tokbind-protocol] and Token Binding over HTTP [I-D.ietf-tokbind-https]) to Access Tokens, Authorization Codes, and Refresh Tokens. 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 RFC 2119 [RFC2119].

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 [I-D.ietf-tokbind-https].

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 [I-D.ietf-tokbind-https]. 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
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 federated scenarios described in Section 4 (Federation Use Cases) of Token Binding over HTTP [I-D.ietf-tokbind-https] 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 as 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 [I-D.ietf-oauth-native-apps]. The base64url-encoded representation of the exported keying material (EKM) from that TLS connection is "p6ZuSwfl6pIe8es5KyeV76T4sw2mQp0_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_Cbl7LWlt6Xjp3DbjiDjavGFiKP2HV_2JSE42VzmKOVVV8m7eqAAQOKiDK10i0z6v4X5BP7uc0pFestVZ42TTojMoHpji06Qq3jsCiCRSJx9ck2fWJYx8tLVXRZPATB3x6c24aY0ZEA AAA

grant_type=authorization_code&code=4bwcZesc7Xacc330tc66Wxk8EAfp9j2&code_verifier=2x6_y1S390-BV7jaT9wj.8qP9nKmYcf.V-rD904r_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":"ACClZEIQTjW9arT9GOJJGgd7QNWqOMmUYfsJTiv8his4",
"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: AIkAAgBBQGto7hHRR0Y5nkOWqcSKNfwW95dEFmSI_tCZ_Cbl 71WIt6Xjp3DbjiDjavGF1KF2HV_2JSE42VzmKOVVV8m7eqAAQpGbA_G_YRf27qOra LOUT4fsKKjL6PukuOT00qzamoAXxOq7m_id703mLpnb_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 Section 4 (Federation Use Cases) of Token Binding over HTTP [I-D.ietf-tokbind-https]. 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 client.
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 Section 5 (Implementation Considerations) of Token Binding over HTTP [I-D.ietf-tokbind-https].

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 Section 4.2 of OAuth 2.0 [RFC6749], 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 "jI5UAyjs5XCPISUGQIwgcSrOivIvWq4fhLVIFTQ4nLxc".

GET /as/authorization.oauth2?response_type=token&client_id=example-client-id&state=rM8pZxG1c3gKy6rEbsD8s&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1
Host: server.example.com
Sec-Token-Binding: ARIAAgBBQIEE8mSMtDy2dj9EEBdXaQT9W3Rg1NS-jW8ebPoF6FylJ0jfATVE55zlicg0TzEg1xeIrC3DsGegwjs4bhwl4AQCKD1AXFFMyQkZegCw1bTlqX39H9Tt-ljXFU_pi16ezka7qVRCpsF0BQLfsQ1sMxMByfSSCJX1BDtrIL7PXj__fUAAECAEFA1BNuLiP3te5WrwlEwJefEz00pesm5FE1Wc7kZ5mLSqQTj1ciP5vQ30LLUCyM_a2BYTUFKtd5EdS-PalT4t6ABADgeizRa5NkTMuX4z0dC-R4cLNWVV081Lu2Psko-UJLR_XAH4Q0H7-m0_nQR1zBN78nYMKPvHzs8L3zVKRVyXegAA

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...

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.

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 "4jTc5e1QpocqPTZ516jsb6pRP181FKdwwPvasYjn1-E".

POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: ARIAAgBBQJFXJir2w4gbJ7grBx9uTYWiRs9V50-PW4ZijegQ0LUM-_bGnGT6izxUK-m5n3dQUIkeH7ybn6wb1C5dGyV_IAAQDDFToFrHt41Zppq7u_SEMF_E-KimAB-HewW12MvZzAQ9QKoWiJCLFiCkjgtr1RrA2-jaJvoB8o51DTGXQydWykAAAECAEFAuC1G1YU83rqTGHEauoqvNwy0fDsdXziyT_4xlFcldsMxjFkJacIBJFGuYcccvnCak_duFi3QKFENuwxql-H9ABAMcuU7IjJOUA4IyE6YoEcfz9BMPQqwM5M6hw4RZNQd58fsTCCslQE_NmNC19JXy4NkdkeZBxqv2GPr0y8QZ_bmAwAA

refresh_token=gZR_ZI8EAhLgWR-gWxBimbgZRZi_8EAhLgLWrWxXimbf
&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.
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 "7LsNP3BT1aHHdXdk6meEWjtsKiPV1b7YS6iHj-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...]/cs29j5c3
Sec-Token-Binding: AIkAAgBBQLgtRpWFPN66kxhxGrtaKrzcMtw7HV8yMk__-MdR
XJXbDMyxZCWnCASRRrmHHHL5wmpF3byYt0ChRDbsMapfh_QAQ01He3Ftj4Wa_S_fz
ZVns4saLfj6aBoMSQW6rLs19IvHze7LrGjKyCfPTKXjaebxp-TLPFZCc0JTqTY5
_OMBAAAA

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"
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-yTpG1rAwKAw"  
  }
}
```

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 an "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": "7NRBu9iDdJ1YCTOqyeYuLxXv0blEA-yTpmGIrAwKAws"
  }
}

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

4. 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.

4.1. 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.
4.1.1. 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.

4.1.1.1. 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

4.1.2. Code Verifier

Upon receipt of the authorization code, the client sends the access token request to the token endpoint. The Token Binding Protocol [I-D.ietf-tokbind-protocol] 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 [I-D.ietf-tokbind-https], 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.

4.1.2.1. 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 "pNVktFuQFy1NYn000QowWwrOkoeMkeX9H32hVuU71Bs".

```
POST /as/token.oauth2 HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Sec-Token-Binding: AIkAAgBBQEOO9GRFP-LM0hoWw6-2i318BsuuUum5AL8bt1sz
  lr1EFfp5DMXMNW308WjcIXr2DKJnI4xnuGsE6GywQd9RbD0AQJDb3xyo9PBxj8M6Y
  jlt-60axgDkycBoTkyrnNbc8tJQ0JtXomKzBb9j5gPtHduXc6xz_lzvNpxSPxi42
  8m7wkAAA

grant_type=authorization_code&code=mJAReTWXX7zI3oHUNd4o3PeNqNqxBKg6
&code_verifier=provided_tb&client_id=example-native-client-id
```

Figure 15: Token Request with PKCE Verifier

4.2. 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.

4.2.1. 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 Section 5.3 of Token Binding over HTTP [I-D.ietf-tokbind-https].

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.

4.2.1.1. 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=P4FUFqYzsi1j3ffsYCP34d3
 &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-1YwZGmnVHyReN5vd2CxcxRBN69e4cI".

GET /as/authorization.oauth2?response_type=code
 &client_id=example-web-client-id&state=dryo8YFpWacbhUpjhBf4Nvt51
 &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-XOPf5ePlf7ikATiAFEGOS503IPmRfkyyzmdWw
HCxl10njxC3D0EmVfB0qIRQzZIfkF7tWby2ZfyaE6XpwTsAQhFZ8vMSgDX_F
_d_b2d1HyH1MmkIz8iMVBY_reW9BOUafJz51B7PG9z11j58LoGQmQoI9NXYktKZ
RXxrYAAAECAGFEdUFTnQAdkn1uDdbQnVeKE6c0Qs38L92gv-KO-q1YadLoDIKe2h53
hSiKwIP981Rj_unedkNkAMyg9e2mY4G7WwBaweUOwawSNz1e6gKohwN4SAZ5eNyx
45Mh8VI4woL1BipLoqrJRoK6dxFkWgHRMuBR0cLGuJ5PiOoxybQH_Tom3gAA

Figure 17: Authorization Request

4.2.2. 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.
4.2.2.1. 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_dgL_iijvqDnymGnS6AyCnToed4UoisCD_fIkY_7p3nZDZAdmPXtpmOBqesiAQLeWg6Z9Tq7zCDBib6Gl1Zk1i3Mh32KNfLefLJc1vR1x817OMfPLZHP2Woxh6rEtmgBcAABuEBbTz7muN1Ln8uoAAA

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 b3JnLmV4YW1wbGUxY2xpZW50Om1ldG5OGNoaWWhZ28=
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_dgL_iijv qpWgnS6AyCnToed4UoisCD_fIkY_7p3nZDZAdmPXtpmOBqesi

Figure 19: Exchange Authorization Code

5. Phasing in Token Binding and Preventing Downgrade Attacks

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 Token Binding failures at any step as fatal errors. 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 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.

If all the participants in an authorization interaction support Token Binding and yet one or more of them does not use it, this is likely evidence of a downgrade attack. In this case, the authorization SHOULD be aborted with an error. For instance, if the protected resource knows that the authorization server and the user agent both support Token Binding and yet the access token received does not contain Token Binding information, this is almost certainly a sign of an attack.

The authorization server, client, and protected resource can determine whether the others support Token Binding using the metadata values defined in the next section. They can determine whether the user agent supports Token Binding by whether it negotiated Token Binding for the TLS connection.

6. Token Binding Metadata

6.1. 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".
6.2. Token Binding Authorization Server Metadata

Authorization servers supporting Token Binding that also support OAuth 2.0 Authorization Server Metadata [OAuth.AuthorizationMetadata] 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".

6.3. Token Binding Protected Resource Metadata

Protected resources supporting Token Binding that also support the OAuth 2.0 Protected Resource Metadata [OAuth.ResourceMetadata] use this metadata value to declare their support for Token Binding of access tokens:

resource_access_token_token_binding_supported
OPTIONAL. Boolean value specifying whether the protected resource supports Token Binding of access tokens. If omitted, the default value is "false".

7. Security Considerations

If a refresh request is received by the authorization server containing a Referred Token Binding ID and the refresh token in the request is not itself token bound, then it is not clear that token binding the access token adds significant value. This situation should be considered an open issue for discussion by the working group.

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

- Client Metadata Name: "client_access_token_token_binding_supported"
- Client Metadata Description: Boolean value specifying whether the client supports Token Binding of access tokens
- Change Controller: IESG
- Specification Document(s): Section 6.1 of [[ this specification ]]

- Client Metadata Name: "client_refresh_token_token_binding_supported"
- Client Metadata Description: Boolean value specifying whether the client supports Token Binding of refresh tokens
- Change Controller: IESG
- Specification Document(s): Section 6.1 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 established by [OAuth.AuthorizationMetadata]:

8.2.1. Registry Contents

- Metadata Name: "as_access_token_token_binding_supported"
- Metadata Description: Boolean value specifying whether the authorization server supports Token Binding of access tokens
- Change Controller: IESG
- Specification Document(s): Section 6.2 of [[ this specification ]]

- Metadata Name: "as_refresh_token_token_binding_supported"
- Metadata Description: Boolean value specifying whether the authorization server supports Token Binding of refresh tokens
- Change Controller: IESG
- Specification Document(s): Section 6.2 of [[ this specification ]]

8.3. OAuth Protected Resource Metadata Registration

This specification registers the following client metadata definition in the IANA "OAuth Protected Resource Metadata" registry established by [OAuth.ResourceMetadata]:

8.3.1. Registry Contents

- Resource Metadata Name: "resource_access_token_token_binding_supported"
- Resource Metadata Description: Boolean value specifying whether the protected resource supports Token Binding of access tokens
8.4. 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].

8.4.1. Registry Contents

- Code Challenge Method Parameter Name: TB-S256
  - Change controller: IESG
  - Specification document(s): Section 4.1.1 of [[ this specification ]]

- Code Challenge Method Parameter Name: referred_tb
  - Change controller: IESG
  - Specification document(s): Section 4.2.1 of [[ this specification ]]

9. References

9.1. Normative References

[I-D.ietf-tokbind-https]

[I-D.ietf-tokbind-negotiation]

[I-D.ietf-tokbind-protocol]

[IANA.OAuth.Parameters]


9.2. Informative References


Appendix A. Acknowledgements

The authors would like to thank the following people for their contributions to the specification: Dirk Balfanz, Andrei Popov, Justin Richer, and Nat Sakimura.

Appendix B. Open Issues

- What should we do in the case that a refresh request for a token bound access token is received when the refresh token used in the request is not token bound?
Currently the only way to request a token bound access token is via the referred token binding. By definition the referred token binding also comes with the provided token binding and the provided token binding is what is used to bind the refresh token. However, web server clients will typically be distributed/clustered and very likely will not want to, or be capable of, dealing with token bound refresh tokens. Such clients will have credentials established with the AS for authenticating to the token endpoint and refresh tokens are already bound to the client. So token binding the refresh tokens doesn’t add much, if anything, in this case. But accessing private token binding keys in a distributed system will be cumbersome or even impossible. Tracking and properly utilizing the association of a token binding key with each individual refresh token would also be exceptionally cumbersome (whereas client credentials are for the client and decoupled from individual refresh tokens) but without some such mechanism the token binding key cannot be changed without implicitly invalidating all the bound refresh tokens the web server client has stored for that AS. It seems necessary to provide some mechanism for a client to opt-out of having refresh tokens token bound while still allowing for token binding of access tokens.

Should the scope of this document include standardization or guidance on token binding of JWT Client Authentication and/or Authorization Grants from RFC 7523?

The Metadata (Section 6) and what can and cannot be reliably inferred from it (Section 5) need additional evaluation and work. OAuth 2.0 Protected Resource Metadata [OAuth.ResourceMetadata] is no longer a going concern, but is currently referenced herein. Boolean values do not adequately convey Token Binding support, as different components may support different key parameters types. And successful negotiation likely doesn’t provide the application layer info about all the supported key parameters types but rather just the one that was negotiated.

Appendix C. Document History

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

-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/

John Bradley
Ping Identity
Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Brian Campbell
Ping Identity
Email: brian.d.campbell@gmail.com
URI: https://twitter.com/__b_c

William Denniss
Google
1600 Amphitheatre Pkwy
Mountain View, CA  94043
USA
Email: wdenniss@google.com
URI: http://wdenniss.com/