Internet Engineering Task Force (IETF)
Request for Comments: 8473
Category: Standards Track
ISSN: 2070-1721

A. Popov
M. Nystroem
Microsoft Corp.
D. Balfanz, Ed.
N. Harper
Google Inc.
J. Hodges
Kings Mountain Systems

October 2018

Token Binding over HTTP

Abstract

This document describes a collection of mechanisms that allow HTTP servers to cryptographically bind security tokens (such as cookies and OAuth tokens) to TLS connections.

We describe both first-party and federated scenarios. In a first-party scenario, an HTTP server is able to cryptographically bind the security tokens that it issues to a client -- and that the client subsequently returns to the server -- to the TLS connection between the client and the server. Such bound security tokens are protected from misuse, since the server can generally detect if they are replayed inappropriately, e.g., over other TLS connections.

Federated Token Bindings, on the other hand, allow servers to cryptographically bind security tokens to a TLS connection that the client has with a different server than the one issuing the token.

This document is a companion document to "The Token Binding Protocol Version 1.0" (RFC 8471).

Status of This Memo

This is an Internet Standards Track document.

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

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8473.

Table of Contents

1. Introduction

The Token Binding protocol [RFC8471] defines a Token Binding ID for a TLS connection between a client and a server. The Token Binding ID of a TLS connection is constructed using the public key of a private-public key pair. The client proves possession of the corresponding private key. This Token Binding key pair is long-lived. That is, subsequent TLS connections between the same client and server have the same Token Binding ID, unless specifically reset, e.g., by the user. When issuing a security token (e.g., an HTTP cookie or an OAuth token [RFC6749]) to a client, the server can include the Token Binding ID in the token, thus cryptographically binding the token to TLS connections between that particular client and server, and inoculating the token against abuse (reuse, attempted impersonation, etc.) by attackers.

While the Token Binding protocol [RFC8471] defines a message format for establishing a Token Binding ID, it does not specify how this message is embedded in higher-level protocols. The purpose of this specification is to define how TokenBindingMessages are embedded in HTTP (both versions 1.1 [RFC7230] and 2 [RFC7540]). Note that TokenBindingMessages are only defined if the underlying transport uses TLS. This means that Token Binding over HTTP is only defined when HTTP is layered on top of TLS (commonly referred to as HTTPS [RFC2818]).

HTTP clients establish a Token Binding ID with a server by including a special HTTP header field in HTTP requests. The HTTP header field value is a base64url-encoded TokenBindingMessage.

A TokenBindingMessage allows a client to establish multiple Token Binding IDs with the server by including multiple TokenBinding structures. By default, a client will establish a Provided Token Binding ID with the server, indicating a Token Binding ID that the client will persistently use with the server. Under certain conditions, the client can also include a Referred Token Binding ID in the TokenBindingMessage, indicating a Token Binding ID that the client is using with a different server than the one that the TokenBindingMessage is sent to. This is useful in federation scenarios.

1.1. Requirements Language

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.

2. The Sec-Token-Binding HTTP Request Header Field

Once a client and server have negotiated the Token Binding protocol with HTTP/1.1 or HTTP/2 (see [RFC8471] and [RFC8472]), clients MUST include a Sec-Token-Binding header field in their HTTP requests and MUST include only one such header field per HTTP request. Also, the Sec-Token-Binding header field MUST NOT be included in HTTP responses. The ABNF of the Sec-Token-Binding header field is (per the style of [RFC7230]; see also Section 8.3 of [RFC7231]):

```
Sec-Token-Binding = EncodedTokenBindingMessage
```

The header field name is Sec-Token-Binding, and its single value, EncodedTokenBindingMessage, is a base64url encoding of a single TokenBindingMessage, as defined in [RFC8471]. The base64url encoding uses the URL and filename safe character set described in Section 5 of [RFC4648], with all trailing padding characters (i.e., "=") omitted and without the inclusion of any line breaks, whitespace, or other additional characters.

For example:

```
Sec-Token-Binding: AIkAAgBBQFzK4_bhAqLDwRQxqJWte33d7hZ0hZWHwk-miKPg4E\
                   9fcgs7gBPoz-9RfuDfN9WCw6keHEw1ZPQMGs9CxpuHm-YAQM_j\
                   aOwwej6a-cQBGU7CJpUHOvXG4VvjNq8jDsvta9Y8_bPEPj25Gg\
                   mKiPjhJEtZA6mJ_9SNifLvVBTi7fR9wSAAAA
```

(Note that the backslashes and line breaks are provided to ease readability; they are not part of the actual encoded message.)

If the server receives more than one Sec-Token-Binding header field in an HTTP request, then the server MUST reject the message with a 400 (Bad Request) HTTP status code. Additionally, the Sec-Token-Binding header field:

- SHOULD NOT be stored by origin servers on PUT requests,
- MAY be listed by a server in a Vary response header field, and
- MUST NOT be used in HTTP trailers.

The TokenBindingMessage MUST contain exactly one TokenBinding structure with a TokenBindingType value of provided_token_binding, which MUST be signed with the Token Binding private key used by the client for connections between itself and the server that the HTTP request is sent to (clients use different Token Binding key pairs for different servers; see Section 2.1 below). The Token Binding ID

established by this TokenBinding is called a "Provided Token Binding ID".

The TokenBindingMessage MAY also contain exactly one TokenBinding structure with a TokenBindingType value of referred_token_binding, as specified in Section 5.3. In addition to the latter, or rather than the latter, the TokenBindingMessage MAY contain other TokenBinding structures. This is specific to the use case in question; such use cases are outside the scope of this specification.

A TokenBindingMessage is validated by the server as described in Section 4.2 ("Server Processing Rules") of [RFC8471]. If validation fails and a Token Binding is rejected, any associated bound tokens MUST also be rejected by the server. HTTP requests containing invalid tokens MUST be rejected. In this case, the server application MAY return HTTP status code 400 (Bad Request) or proceed with an application-specific "invalid token" response (e.g., directing the client to re-authenticate and present a different token), or terminate the connection.

In HTTP/2, the client SHOULD use header compression [RFC7541] to avoid the overhead of repeating the same header field in subsequent HTTP requests.

2.1. HTTPS Token Binding Key-Pair Scoping

HTTPS is used in conjunction with various application protocols and application contexts, in various ways. For example, general-purpose web browsing is one such HTTP-based application context. Within that context, HTTP cookies [RFC6265] are typically utilized for state management, including client authentication. A related, though distinct, example of other HTTP-based application contexts is where OAuth tokens [RFC6749] are utilized to manage authorization for third-party application access to resources. The token-scoping rules of these two examples can differ: the scoping rules for cookies are concisely specified in [RFC6265], whereas OAuth is a framework and defines various token types with various scopings, some of which are determined by the encompassing application.

The scoping of Token Binding key pairs generated by web browsers for the purpose of binding HTTP cookies MUST be no wider than the granularity of a "registered domain" (also known as "effective top-level domain + 1", or "eTLD+1"). An origin's "registered domain" is the origin's host's public suffix plus the label to its left (where the term "public suffix" is defined in the "NOTE:" paragraph in Section 5.3 of [RFC6265] as "a domain that is controlled by a public registry"). For example, for "https://www.example.com", the public suffix (eTLD) is "com", and the registered domain (eTLD+1) is

"example.com". User Agents SHOULD use an up-to-date public suffix list, such as the one maintained by Mozilla [PSL].

This means that in practice the scope of a Token Binding key pair is no larger than the scope of a cookie allowed by a web browser. If a web browser restricts cookies to a narrower scope than registered domains, the scope of Token Binding key pairs MAY also be narrower. This applies to the use of Token Binding key pairs in first-party use cases, as well as in federation use cases defined in this specification (Section 5).

Key pairs used to bind other application tokens, such as OAuth tokens or "OpenID Connect" ID Tokens [OpenID.Core], SHOULD adhere to the above eTLD+1 scoping requirement for those tokens being employed in first-party or federation scenarios. Applications other than web browsers MAY use different key-pair scoping rules. See also Section 8.1 below.

Scoping rules for other HTTP-based application contexts are outside the scope of this specification.

3. TLS Renegotiation

Token Binding over HTTP/1.1 [RFC7230] can be performed in combination with TLS renegotiation. In this case, renegotiation MUST only occur between a client's HTTP request and the server's response, the client MUST NOT send any pipelined requests, and the client MUST NOT initiate renegotiation. (That is, the client may only send a renegotiation ClientHello in response to the server's HelloRequest.) These conditions ensure that both the client and the server can clearly identify which TLS Exported Keying Material value [RFC5705] to use when generating or verifying the TokenBindingMessage. This also prevents a TokenBindingMessage from being split across TLS renegotiation boundaries due to TLS message fragmentation; see Section 6.2.1 of [RFC5246].

(Note: This document deals with TLS 1.2 and therefore refers to RFC 5246 (which has been obsoleted by RFC 8446); [TOKENBIND-TLS13] addresses Token Binding in TLS 1.3.)

4. First-Party Use Cases

In a first-party use case (also known as a "same-site" use case), an HTTP server issues a security token such as a cookie (or similar) to a client and expects the client to return the security token at a later time, e.g., in order to authenticate. Binding the security token to the TLS connection between the client and the server protects the security token from misuse, since the server can detect if the security token is replayed inappropriately, e.g., over other TLS connections.

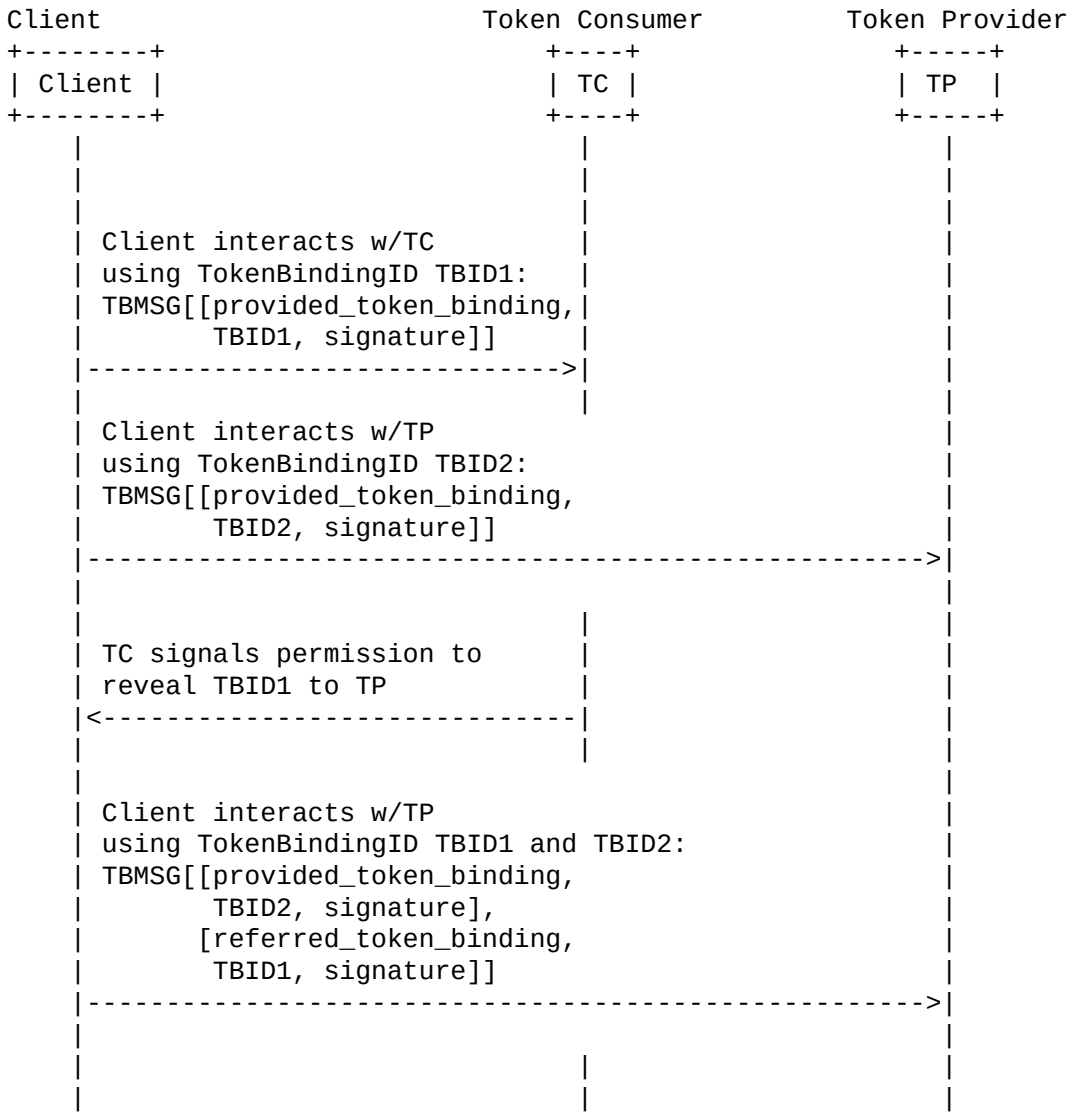
See Section 5 of [RFC8471] for general guidance regarding the binding of security tokens and their subsequent validation.

5. Federation Use Cases

5.1. Introduction

For privacy reasons, clients use different Token Binding key pairs to establish Provided Token Binding IDs with different servers. As a result, a server cannot bind a security token (such as an OAuth token or an OpenID Connect ID Token [OpenID.Core]) to a TLS connection that the client has with a different server. This is, however, a common requirement in federation scenarios: for example, an Identity Provider may wish to issue an identity token to a client and cryptographically bind that token to the TLS connection between the client and a Relying Party.

In this section, we describe mechanisms to achieve this. The common idea among these mechanisms is that a server (called the "Token Consumer" in this document) signals to the client that it should reveal the Provided Token Binding ID that is used between the client and itself to another server (called the "Token Provider" in this document). Also common across the mechanisms is how the Token Binding ID is revealed to the Token Provider: the client uses the Token Binding protocol [RFC8471] and includes a TokenBinding structure in the Sec-Token-Binding HTTP header field defined above. What differs between the various mechanisms is how the Token Consumer signals to the client that it should reveal the Token Binding ID to the Token Provider. Below, we specify one such mechanism, which is suitable for redirect-based interactions between Token Consumers and Token Providers.



5.2. Overview

In a federated sign-on protocol, an Identity Provider issues an identity token to a client, which sends the identity token to a Relying Party to authenticate itself. Examples of this include OpenID Connect (in which the identity token is called an "ID Token") and the Security Assertion Markup Language (SAML) [OASIS.saml-core-2.0-os] (in which the identity token is a SAML assertion).

To better protect the security of the identity token, the Identity Provider may wish to bind the identity token to the TLS connection between the client and the Relying Party, thus ensuring that only said client can use the identity token. The Relying Party will compare the Token Binding ID (or a cryptographic hash of it) in the identity token with the Token Binding ID (or a hash thereof) of the TLS connection between this Relying Party and the client.

This is an example of a federation scenario, which more generally can be described as follows:

- A Token Consumer causes the client to issue a token request to the Token Provider. The goal is for the client to obtain a token and then use it with the Token Consumer.

- The client delivers the token request to the Token Provider.

- The Token Provider issues the token. The token is issued for the specific Token Consumer who requested it (thus preventing malicious Token Consumers from using tokens with other Token Consumers). The token is, however, typically a bearer token, meaning that any client can use it with the Token Consumer -- not just the client to which it was issued.

- Therefore, in the previous step, the Token Provider may want to include in the token the Token Binding ID (or a cryptographic hash of it) that the client uses when communicating with the Token Consumer, thus binding the token to the client's Token Binding key pair. The client proves possession of the private key when communicating with the Token Consumer through the Token Binding protocol [RFC8471] and uses the corresponding public key of this key pair as a component of the Token Binding ID. Comparing the Token Binding ID from the token to the Token Binding ID established with the client allows the Token Consumer to verify that the token was sent to it by the legitimate client.

- To allow the Token Provider to include the Token Binding ID in the token, the Token Binding ID between the client and the Token Consumer must therefore be communicated to the Token Provider along with the token request. Communicating a Token Binding ID involves proving possession of a private key and is described in the Token Binding protocol [RFC8471].

The client will perform this last operation only if the Token Consumer requests the client to do so.

Below, we specify how Token Consumers can signal this request in redirect-based federation protocols. Note that this assumes that the federated sign-on flow starts at the Token Consumer or, at the very least, includes a redirect from the Token Consumer to the Token Provider. It is outside the scope of this document to specify similar mechanisms for flows that do not include such redirects.

5.3. HTTP Redirects

When a Token Consumer redirects the client to a Token Provider as a means to deliver the token request, it SHOULD include an Include-Referred-Token-Binding-ID HTTP response header field in its HTTP response. The ABNF of the Include-Referred-Token-Binding-ID header is (per the style of [RFC7230]; see also Section 8.3 of [RFC7231]):

```
Include-Referred-Token-Binding-ID = "true"
```

Where the header field name is "Include-Referred-Token-Binding-ID" and the field value of "true" is case insensitive. For example:

```
Include-Referred-Token-Binding-ID: true
```

Including this response header field signals to the client that it should reveal, to the Token Provider, the Token Binding ID used between itself and the Token Consumer. In the absence of this response header field, the client will not disclose any information about the Token Binding used between the client and the Token Consumer to the Token Provider.

As illustrated in Section 5.5, when a client receives this header field, it should take the TokenBindingID [RFC8471] of the provided TokenBinding from the referrer and create a referred TokenBinding with it to include in the TokenBindingMessage in the redirect request. In other words, the Token Binding message in the redirect request to the Token Provider now includes one provided binding and one referred binding, the latter constructed from the binding between the client and the Token Consumer.

When a client receives the Include-Referred-Token-Binding-ID header, it includes the referred Token Binding even if both the Token Provider and the Token Consumer fall under the same eTLD+1 and the provided and Referred Token Binding IDs are the same.

The referred Token Binding is sent only in the initial request resulting from the HTTP response that included the Include-Referred-Token-Binding-ID header. Should the response to that initial request be a further redirect, the original referred

Token Binding is no longer included in subsequent requests. (A new referred Token Binding may be included if the redirecting endpoint itself responded with an Include-Referred-Token-Binding-ID response header.)

If the Include-Referred-Token-Binding-ID header field is received in response to a request that did not include the Sec-Token-Binding header field, the client MUST ignore the Include-Referred-Token-Binding-ID header field.

This header field only has meaning if the HTTP status code is a redirection code (300-399) and MUST be ignored by the client for any other status codes. As described in Section 2, if the client supports the Token Binding protocol and has negotiated the Token Binding protocol with both the Token Consumer and the Token Provider, it sends the Sec-Token-Binding header field to the Token Provider with each HTTP request.

The TokenBindingMessage included in the redirect request to the Token Provider SHOULD contain a TokenBinding with a TokenBindingType value of referred_token_binding. If included, this TokenBinding MUST be signed with the Token Binding private key used by the client for connections between itself and the Token Consumer (more specifically, the server that issued the Include-Referred-Token-Binding-ID response header field). The Token Binding ID established by this TokenBinding is called a "Referred Token Binding ID".

As described above, the TokenBindingMessage MUST additionally contain a Provided Token Binding ID, i.e., a TokenBinding structure with a TokenBindingType value of provided_token_binding, which MUST be signed with the Token Binding private key used by the client for connections between itself and the Token Provider (more specifically, the server that the token request is being sent to).

If, for some deployment-specific reason, the initial Token Provider ("TP1") needs to redirect the client to another Token Provider ("TP2") rather than directly back to the Token Consumer, it can be accommodated using the header fields defined in this specification in the following fashion ("the redirect-chain approach"):

> Initially, the client is redirected to TP1 by the Token Consumer ("TC"), as described above. Upon receiving a client's request that contains a TokenBindingMessage that in turn contains both provided and referred TokenBindings (for TP1 and TC, respectively), TP1 responds to the client with a redirect response that (1) contains the Include-Referred-Token-Binding-ID header field and (2) directs the client to send a request to TP2. This causes the client to follow the same pattern and send a request

containing a TokenBindingMessage that contains both provided and referred TokenBindings (for TP2 and TP1, respectively) to TP2. Note that this pattern can continue to additional Token Providers. In this case, TP2 issues a security token, bound to the client's TokenBinding with TP1, and sends a redirect response to the client pointing to TP1. TP1 in turn constructs a security token for the Token Consumer, bound to the TC's referred TokenBinding that had been conveyed earlier, and sends a redirect response pointing to the TC, containing the bound security token, to the client.

The above is intended as only a non-normative example. Details are specific to deployment contexts. Other approaches are possible but are outside the scope of this specification.

5.4. Negotiated Key Parameters

The TLS extension for Token Binding protocol negotiation [RFC8472] allows the server and client to negotiate the parameters (signature algorithm, length) of the Token Binding key pair. It is possible that the Token Binding ID used between the client and the Token Consumer, and the Token Binding ID used between the client and the Token Provider, use different key parameters. The client MUST use the key parameters negotiated with the Token Consumer in the referred_token_binding TokenBinding of the TokenBindingMessage, even if those key parameters are different from the ones negotiated with the server that the header field is sent to.

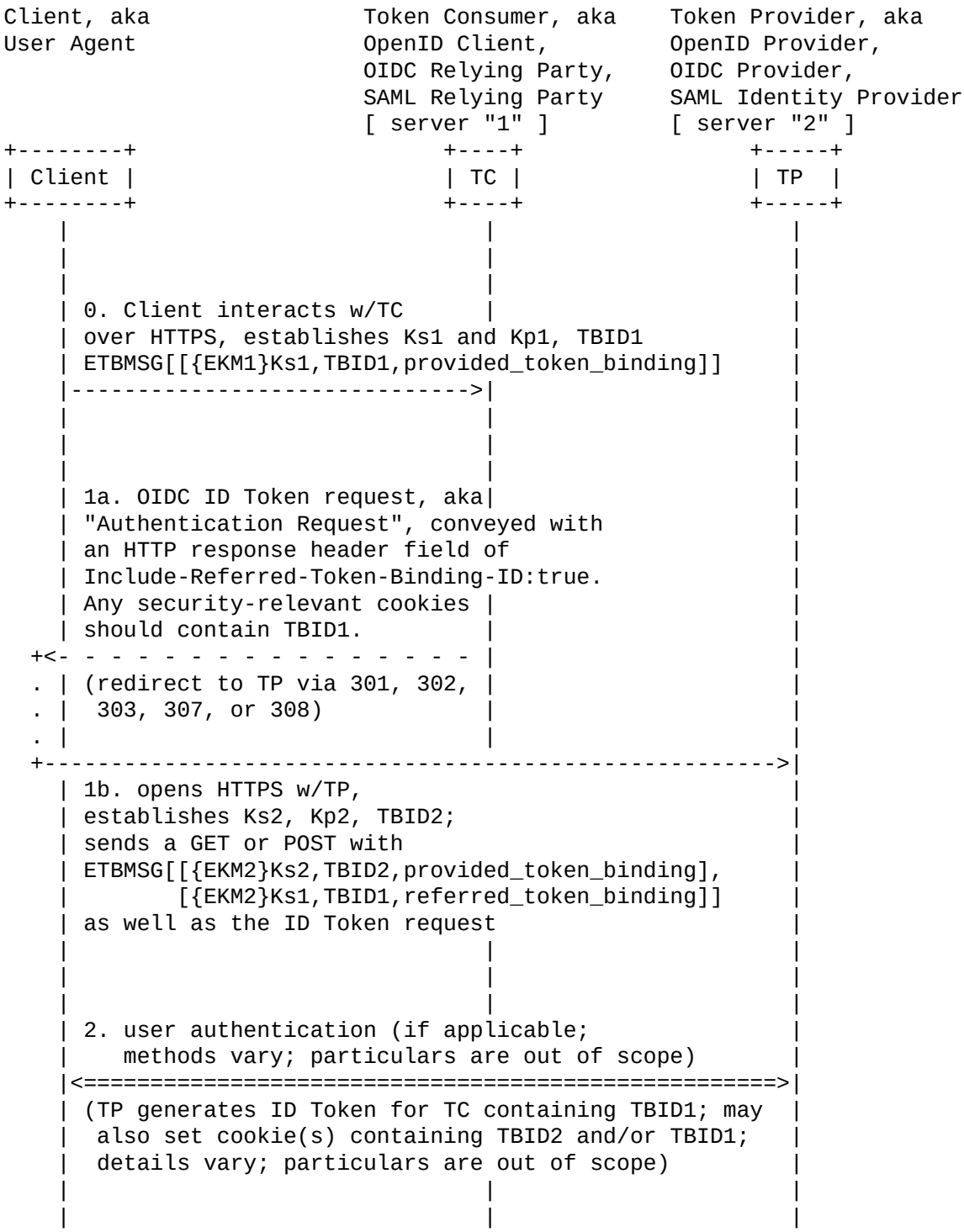
Token Providers SHOULD support all the Token Binding key parameters specified in [RFC8471]. If a Token Provider does not support the key parameters specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST NOT issue a bound token.

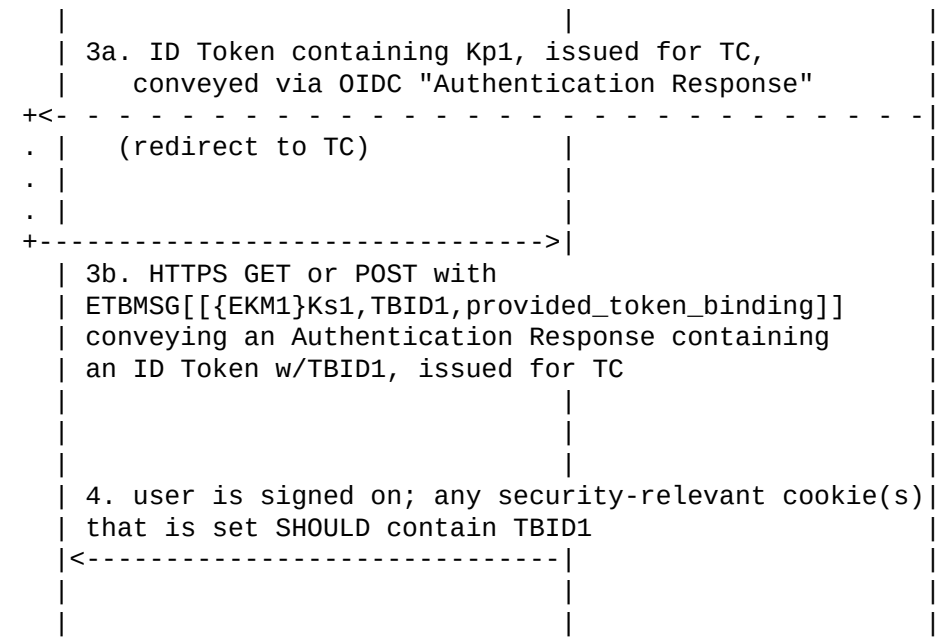
5.5. Federation Example

The diagram below shows a typical HTTP redirect-based web browser single sign-on (SSO) profile (Section 4.1 of [OASIS.saml-prof-2.0-os]) (no artifact, no callbacks), featuring the binding of, for example, a TLS Token Binding ID into an OpenID Connect ID Token.

Legend:

EKM:	TLS Exported Keying Material [RFC5705]
{EKMn}Ksm:	EKM for server "n", signed by the private key of TBID "m", where "n" must represent the server receiving the ETBMSG. If a conveyed TB's type is provided_token_binding, then m = n, else if TB's type is referred_token_binding, then m != n. For example, see step 1b in the diagram below.
ETBMSG:	"Sec-Token-Binding" HTTP header field conveying an EncodedTokenBindingMessage, in turn conveying TokenBinding (TB)struct(s), e.g., ETBMSG[[TB]] or ETBMSG[[TB1],[TB2]]
ID Token:	the ID Token in OpenID Connect. It is the semantic equivalent of a SAML "authentication assertion". "ID Token w/TBIDn" denotes a "token bound" ID Token containing TBIDn.
Ks and Kp:	private (aka secret) key and public key, respectively, of the client-side Token Binding key pair
OIDC:	OpenID Connect
TB:	TokenBinding struct containing a signed EKM, TBID, and TB type, e.g., [{EKM1}Ks1,TBID1,provided_token_binding]
TBIDn:	Token Binding ID for client and server n's token-bound TLS association. TBIDn contains Kpn.


Client, aka
User Agent
Token Consumer, aka
OpenID Client,
OIDC Relying Party,
SAML Relying Party
[server "1"]
Token Provider, aka
OpenID Provider,
OIDC Provider,
SAML Identity Provider
[server "2"]
Client
TC
TP
0. Client interacts w/TC
over HTTPS, establishes Ks1 and Kp1, TBID1
ETBMSG[[{EKM1}Ks1,TBID1,provided_token_binding]]
1a. OIDC ID Token request, aka
"Authentication Request", conveyed with
an HTTP response header field of
Include-Referred-Token-Binding-ID:true.
Any security-relevant cookies
should contain TBID1.
(redirect to TP via 301, 302,
303, 307, or 308)
1b. opens HTTPS w/TP,
establishes Ks2, Kp2, TBID2;
sends a GET or POST with
ETBMSG[[{EKM2}Ks2,TBID2,provided_token_binding],
[{EKM2}Ks1,TBID1,referred_token_binding]]
as well as the ID Token request
2. user authentication (if applicable;
methods vary; particulars are out of scope)
(TP generates ID Token for TC containing TBID1; may
also set cookie(s) containing TBID2 and/or TBID1;
details vary; particulars are out of scope)



6. Implementation Considerations

HTTPS-based applications may have multi-party use cases other than, or in addition to, the HTTP redirect-based signaling and conveyance of referred Token Bindings, as presented above in Section 5.3.

Thus, Token Binding implementations should provide APIs for such applications to generate Token Binding messages containing Token Binding IDs of various application-specified Token Binding types, to be conveyed by the Sec-Token-Binding header field.

However, Token Binding implementations MUST only convey Token Binding IDs to servers if signaled to do so by an application. Signaling mechanisms other than the Include-Referred-Token-Binding-ID HTTP response header field are possible, but these mechanisms are outside the scope of this specification.

NOTE: See Section 8 ("Privacy Considerations") for privacy guidance regarding the use of this functionality.

7. Security Considerations

7.1. Security Token Replay

The goal of the federated Token Binding mechanisms is to prevent attackers from exporting and replaying tokens used in protocols between the client and the Token Consumer, thereby impersonating legitimate users and gaining access to protected resources. Although bound tokens can still be replayed by any malware present in clients (which may be undetectable to a server), in order to export bound tokens to other machines and successfully replay them, attackers also need to export the corresponding Token Binding private keys. Token Binding private keys are therefore high-value assets and SHOULD be strongly protected, ideally by generating them in a hardware security module that prevents key export.

This consideration is a special case of the scenario described in Section 7.1 ("Security Token Replay") of [RFC8471].

7.2. Sensitivity of the Sec-Token-Binding Header

The purpose of the Token Binding protocol is to convince the server that the client that initiated the TLS connection controls a certain key pair. For the server to correctly draw this conclusion after processing the Sec-Token-Binding header field, certain secrecy and integrity requirements must be met.

For example, the client must keep its Token Binding private key secret. If the private key is not secret, then another actor in the system could create a valid Token Binding header field and thereby impersonate the client. This can render the main purpose of the protocol -- to bind bearer tokens to certain clients -- moot. Consider, for example, an attacker who obtained (perhaps through a network intrusion) an authentication cookie that a client uses with a certain server. Consider further that the server bound that cookie to the client's Token Binding ID precisely to thwart misuse of the cookie. If the attacker were to come into possession of the client's private key, they could then establish a TLS connection with the server and craft a Sec-Token-Binding header field that matches the binding present in the cookie, thus successfully authenticating as the client and gaining access to the client's data at the server. The Token Binding protocol, in this case, did not successfully bind the cookie to the client.

Likewise, we need integrity protection of the Sec-Token-Binding header field. A client should not be tricked into sending to a server a Sec-Token-Binding header field that contains Token Bindings signed with any Token Binding keys that the client does not control.

Consider an attacker A that somehow has knowledge of the Exported Keying Material (EKM) for a TLS connection between a client C and a server S. (While that is somewhat unlikely, it is also not entirely out of the question, since the client might not treat the EKM as a secret -- after all, a pre-image-resistant hash function has been applied to the TLS master secret, making it impossible for someone knowing the EKM to recover the TLS master secret. Such considerations might lead some clients to not treat the EKM as a secret.) Such an attacker A could craft a Sec-Token-Binding header field with A's key pair over C's EKM. If the attacker could now trick C into sending such a header field to S, it would appear to S as if C controls a certain key pair, when in fact it does not (the attacker A controls the key pair).

If A has a pre-existing relationship with S (e.g., perhaps has an account on S), it now appears to the server S as if A is connecting to it, even though it is really C. (If the server S does not simply use Token Binding IDs to identify clients but also uses bound authentication cookies, then A would also have to trick C into sending one of A's cookies to S, which it can do through a variety of means -- inserting cookies through JavaScript APIs, setting cookies through related-domain attacks, etc.) In other words, in this scenario, A can trick C into logging into A's account on S. This could lead to a loss of privacy for C, since A presumably has some other way to also access the account and can thus indirectly observe C's behavior (for example, if S has a feature that lets account holders see their activity history on S).

Therefore, we need to protect the integrity of the Sec-Token-Binding header field. One eTLD+1 should not be able to set the Sec-Token-Binding header field (through a Document Object Model (DOM) API [W3C.REC-DOM-Level-3-Core-20040407] or otherwise) that the User Agent uses with another eTLD+1. Employing the "Sec-" header field prefix helps to meet this requirement by denoting the header field name as a "forbidden header name"; see [fetch-spec].

7.3. Securing Federated Sign-On Protocols

As explained above, in a federated sign-on scenario, a client will prove possession of two different Token Binding private keys to a Token Provider: one private key corresponds to the "provided" Token Binding ID (which the client normally uses with the Token Provider), and the other is the Token Binding private key corresponding to the "referred" Token Binding ID (which the client normally uses with the Token Consumer). The Token Provider is expected to issue a token that is bound to the Referred Token Binding ID.

Both proofs (that of the provided Token Binding private key and that of the referred Token Binding private key) are necessary. To show this, consider the following scenario:

- The client has an authentication token with the Token Provider that is bound to the client's Token Binding ID used with that Token Provider.
- The client wants to establish a secure (i.e., free of men-in-the-middle) authenticated session with the Token Consumer but has not yet done so (in other words, we are about to run the federated sign-on protocol).
- A man-in-the-middle is allowed to intercept the connection between the client and the Token Consumer or between the client and the Token Provider (or both).

The goal is to detect the presence of the man-in-the-middle in these scenarios.

First, consider a man-in-the-middle between the client and the Token Provider. Recall that we assume that the client possesses a bound authentication token (e.g., cookie) for the Token Provider. The man-in-the-middle can intercept and modify any message sent by the client to the Token Provider and any message sent by the Token Provider to the client. (This means, among other things, that the man-in-the-middle controls the JavaScript running at the client in the origin of the Token Provider.) It is not, however, in possession of the client's Token Binding private key. Therefore, it can choose to either (1) replace the Token Binding ID in requests from the client to the Token Provider and create a Sec-Token-Binding header field that matches the TLS connection between the man-in-the-middle and the Token Provider or (2) leave the Sec-Token-Binding header field unchanged. If it chooses the latter, the signature in the Token Binding message (created by the original client on the EKM for the connection between the client and the man-in-the-middle) will not match a signature on the EKM between the man-in-the-middle and the Token Provider. If it chooses the former (and creates its own signature, using its own Token Binding private key, over the EKM for the connection between itself, the man-in-the-middle, and the Token Provider), then the Token Binding message will match the connection between the man-in-the-middle and the Token Provider, but the Token Binding ID in the message will not match the Token Binding ID that the client's authentication token is bound to. Either way, the man-in-the-middle is detected by the Token Provider, but only if the proof of possession of the provided Token Binding private key is required in the protocol (as is done above).

Next, consider the presence of a man-in-the-middle between the client and the Token Consumer. That man-in-the-middle can intercept and modify any message sent by the client to the Token Consumer and any message sent by the Token Consumer to the client. The Token Consumer is the party that redirects the client to the Token Provider. In this case, the man-in-the-middle controls the redirect URL and can tamper with any redirect URL issued by the Token Consumer (as well as with any JavaScript running in the origin of the Token Consumer). The goal of the man-in-the-middle is to trick the Token Provider into issuing a token bound to its Token Binding ID and not to the Token Binding ID of the legitimate client. To thwart this goal of the man-in-the-middle, the client's Referred Token Binding ID must be communicated to the Token Provider in a manner that cannot be affected by the man-in-the-middle (who, as mentioned above, can modify redirect URLs and JavaScript at the client). Including the referred TokenBinding structure in the Sec-Token-Binding header field (as opposed to, say, including the Referred Token Binding ID in an application-level message as part of the redirect URL) is one way to assure that the man-in-the-middle between the client and the Token Consumer cannot affect the communication of the Referred Token Binding ID to the Token Provider.

Therefore, the Sec-Token-Binding header field in the federated sign-on use case contains both a proof of possession of the provided Token Binding key and a proof of possession of the referred Token Binding key.

Note that the presence of Token Binding does not relieve the Token Provider and Token Consumer from performing various checks to ensure the security of clients during the use of federated sign-on protocols. These include the following:

- The Token Provider should not issue tokens to Token Consumers that have been shown to act maliciously. To aid in this, the federation protocol should identify the Token Consumer to the Token Provider (e.g., through OAuth client IDs or similar mechanisms), and the Token Provider should ensure that tokens are indeed issued to the Token Consumer identified in the token request (e.g., by verifying that the redirect URI is associated with the OAuth client ID).

- The Token Consumer should verify that the tokens were issued for it and not for some other Token Consumer. To aid in this, the federation protocol should include an audience parameter in the token response or apply equivalent mechanisms (the implicit OAuth flow requires Token Consumers to identify themselves when they exchange OAuth authorization codes for OAuth refresh tokens, leaving it up to the Token Provider to verify that the OAuth authorization was delivered to the correct Token Consumer).

8. Privacy Considerations

8.1. Scoping of Token Binding Key Pairs

Clients use different Token Binding key pairs for different servers, so as to not allow Token Binding to become a tracking tool across different servers. However, the scoping of the Token Binding key pairs to servers varies according to the scoping rules of the application protocol (Section 4.1 of [RFC8471]).

In the case of HTTP cookies, servers may use Token Binding to secure their cookies. These cookies can be attached to any subdomain of effective top-level domains (eTLDs), and clients therefore should use the same Token Binding key pair across such subdomains. This will ensure that any server capable of receiving the cookie will see the same Token Binding ID from the client and thus be able to verify the Token Binding of the cookie. See Section 2.1 above.

If the client application is not a web browser, it may have additional knowledge about the relationship between different servers. For example, the client application might be aware of the fact that two servers play the roles of Relying Party and Identity Provider, respectively, in a federated sign-on protocol and that they therefore share the identity of the user. In such cases, it is permissible to use different Token Binding key-pair scoping rules, such as using the same Token Binding key pair for both the Relying Party and the Identity Provider. Absent such special knowledge, conservative key-pair scoping rules should be used, assuring that clients use different Token Binding key pairs with different servers.

8.2. Lifetime of Token Binding Key Pairs

Token Binding key pairs do not have an expiration time. This means that they can potentially be used by a server to track a user for an extended period of time (similar to a long-lived cookie). HTTPS clients such as web User Agents SHOULD therefore provide a user interface for discarding Token Binding key pairs (similar to the controls provided for deleting cookies).

If a User Agent provides modes such as private browsing mode in which the user is promised that browsing state such as cookies are discarded after the session is over, the User Agent MUST also discard Token Binding key pairs from such modes after the session is over. Generally speaking, users should be given the same level of control over the lifetime of Token Binding key pairs as they have over cookies or other potential tracking mechanisms.

8.3. Correlation

An application's various communicating endpoints that receive Token Binding IDs for TLS connections other than their own obtain information about the application's other TLS connections. (In this context, "an application" is a combination of client-side and server-side components, communicating over HTTPS, where the client side may be web-browser-based, native-application-based, or both.) These other Token Binding IDs can serve as correlation handles for the endpoints of the other connections. If the receiving endpoints are otherwise aware of these other connections, then no additional information is being exposed. For instance, if in a redirect-based federation protocol the Identity Provider and Relying Party already possess URLs for one another, then also having Token Binding IDs for these connections does not provide additional correlation information. If not, by providing the other Token Binding IDs, additional information is then exposed that can be used to correlate the other endpoints. In such cases, a privacy analysis of enabled correlations and their potential privacy impacts should be performed as part of the application design decisions of how, and whether, to utilize Token Binding.

Also, Token Binding implementations must take care to only reveal Token Binding IDs to other endpoints if signaled to do so by the application associated with a Token Binding ID; see Section 6 ("Implementation Considerations").

Finally, care should be taken to ensure that unrelated applications do not obtain information about each other's Token Bindings. For instance, a Token Binding implementation shared between multiple applications on a given system should prevent unrelated applications from obtaining each other's Token Binding information. This may be accomplished by using techniques such as application isolation and key segregation, depending upon system capabilities.

9. IANA Considerations

Below is the Internet Assigned Numbers Authority (IANA) "Permanent Message Header Field Names" registration information per [RFC3864].

Header Field name:	Sec-Token-Binding
Protocol:	HTTP
Status:	standard
Reference:	This document

Header Field name:	Include-Referred-Token-Binding-ID
Protocol:	HTTP
Status:	standard
Reference:	This document

10. References

10.1. Normative References

[PSL] Mozilla, "Public Suffix List", <https://publicsuffix.org/>.

[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>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

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

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

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/info/rfc7541>.

[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>.

[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>.

10.2. Informative References

[fetch-spec]
WhatWG, "Fetch", Living Standard, <https://fetch.spec.whatwg.org/>.

[OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.

[OASIS.saml-prof-2.0-os]
Hughes, J., Ed., Cantor, S., Ed., Hodges, J., Ed., Hirsch, F., Ed., Mishra, P., Ed., Philpott, R., Ed., and E. Maler, Ed., "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard OASIS.saml-profiles-2.0-os, March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>.

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[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>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[TOKENBIND-TLS13]
Harper, N., "Token Binding for Transport Layer Security (TLS) Version 1.3 Connections", Work in Progress, draft-ietf-tokbind-tls13-01, May 2018.

[W3C.REC-DOM-Level-3-Core-20040407]
Le Hors, A., Ed., Le Hegaret, P., Ed., Wood, L., Ed., Nicol, G., Ed., Robie, J., Ed., Champion, M., Ed., and S. Byrne, Ed., "Document Object Model (DOM) Level 3 Core Specification", World Wide Web Consortium Recommendation REC-DOM-Level-3-Core-20040407, April 2004, <https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407>.

Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam, Anthony Nadalin, Michael B. Jones, Bill Cox, Brian Campbell, and others.

This document was produced under the chairmanship of John Bradley and Leif Johansson. The area directors included Eric Rescorla, Kathleen Moriarty, and Stephen Farrell.

Authors' Addresses

Andrei Popov
Microsoft Corp.
United States of America

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
United States of America

Email: mnystrom@microsoft.com

Dirk Balfanz (editor)
Google Inc.
United States of America

Email: balfanz@google.com

Nick Harper
Google Inc.
United States of America

Email: nharper@google.com

Jeff Hodges
Kings Mountain Systems
United States of America

Email: Jeff.Hodges@KingsMountain.com