Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: February 27, 2017

A. Popov
M. Nystroem
Microsoft Corp.
D. Balfanz, Ed.
A. Langley
Google Inc.
J. Hodges
Paypal

August 26, 2016

Token Binding over HTTP
draft-ietf-tokbind-https-06

Abstract

This document describes a collection of mechanisms that allow HTTP servers to cryptographically bind authentication tokens (such as cookies and OAuth tokens) to TLS [RFC5246] 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 it issues to a client, and which the client subsequently returns to the server, to the TLS connection between the client and 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 Internet-Draft is a companion document to The Token Binding Protocol [I-D.ietf-tokbind-protocol]

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 February 27, 2017.

Table of Contents

1. Introduction

The Token Binding Protocol [I-D.ietf-tokbind-protocol] defines a Token Binding ID for a TLS connection between a client and a server. The Token Binding ID of a TLS connection is related to a private key, that the client proves possession of to the server, and is long-lived (i.e., subsequent TLS connections between the same client and server have the same Token Binding ID). When issuing a security token (e.g. an HTTP cookie or an OAuth token) 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 (re-use, attempted impersonation, etc.) by attackers.

While the Token Binding Protocol [I-D.ietf-tokbind-protocol] 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 the HTTP protocol is layered on top of TLS (commonly referred to as HTTPS).

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.

TokenBindingMessages allow clients to establish multiple Token Binding IDs with the server, by including multiple TokenBinding structures in the TokenBindingMessage. 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. The Sec-Token-Binding Header Field

Once a client and server have negotiated the Token Binding Protocol with HTTP/1.1 or HTTP/2 (see [I-D.ietf-tokbind-protocol] and [I-D.ietf-tokbind-negotiation]), clients MUST include the Sec-Token-Binding header field in their HTTP requests. The ABNF of the Sec-Token-Binding header field is (in [RFC7230] style, see also [RFC7231] Section 8.3):

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

The header field name is "Sec-Token-Binding" and its value is a base64url encoding of the TokenBindingMessage defined in [I-D.ietf-tokbind-protocol] using the URL- and filename-safe character set described in Section 5 of [RFC4648], with all trailing pad characters '=' omitted and without the inclusion of any line breaks, whitespace, or other additional characters.

For example:

```
Sec-Token-Binding: <base64url-encoded TokenBindingMessage>
```

The TokenBindingMessage MUST contain one TokenBinding structure with TokenBindingType 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 keys 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 one TokenBinding structure with TokenBindingType of referred_token_binding, as specified in Section 4.3. In addition to the latter, or rather than the latter, the TokenBindingMessage MAY contain other TokenBinding structures. This is use case-specific, and such use cases are outside the scope of this specification.

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 the latter 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 Token Binding key pair scoping for those key pairs generated in the context of the first-party and federation use cases defined in this specification (below), and to be used for binding HTTP cookies MUST be at the granularity of "effective top-level domain (public suffix) + 1" (eTLD+1), i.e., at the same granularity at which cookies can be set (see [RFC6265]). Key pairs used to bind other application tokens, such as OAuth tokens, SHOULD adhere to the above eTLD+1 scoping requirement for those tokens being employed in first-party or federation scenarios as described below, e.g., OAuth refresh tokens or Open ID Connect "ID Tokens". See also Section 7.1, below.

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

3. First-party Use Cases

In a first-party 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 client and 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 [I-D.ietf-tokbind-protocol] Section 6 for general guidance regarding binding of security tokens and their subsequent validation.

4. Federation Use Cases

4.1. Introduction

For privacy reasons, clients use different private keys 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 identity token) 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 [I-D.ietf-tokbind-protocol], 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.

4.2. 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 (where the identity token is called "ID Token") and SAML (where 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 in the identity token with the Token Binding ID of the TLS connection between it 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 public key that the client

uses when communicating with the Token Consumer, thus _binding_ the token to client's Token-Binding keypair. The client proves possession of the private key when communicating with the Token Consumer through the Token Binding Protocol [I-D.ietf-tokbind-protocol], and reveals the corresponding public key of this keypair as part of the Token Binding ID. Comparing the public key from the token with the public key from the Token Binding ID 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 public key in the token, the Token Binding ID (between client and 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 [I-D.ietf-tokbind-protocol].

The client will perform this last operation (proving possession of a private key that corresponds to a Token Binding ID between the client and the Token Consumer while delivering the token request to the Token Provider) 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 include a redirect from Token Consumer to Token Provider. It is outside the scope of this document to specify similar mechanisms for flows that do not include such redirects.

4.3. HTTP Redirects

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

```
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 4.5, when a client receives this header field, it should take the TokenBindingID of the provided TokenBinding from the referrer and create a referred TokenBinding with it to include in the TokenBindingMessage on 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. Note that the referred token binding is sent only on the request resulting from the redirect and not on any subsequent requests to the Token Provider.

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

This header field has only meaning if the HTTP status code is 301, 302, 303, 307 or 308, and MUST be ignored by the client for any other status codes. 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 already sends the Sec-Token-Binding header field to the Token Provider with each HTTP request (see above).

The TokenBindingMessage SHOULD contain a TokenBinding with TokenBindingType referred_token_binding. If included, this TokenBinding MUST be signed with the Token Binding key used by the client for connections between itself and the Token Consumer (more specifically, the web origin 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 TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the Token Provider (more specifically, the web origin 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 accomodated 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 the client's request, containing a TokenBindingMessage which contains both provided and referred TokenBindings (for TP1 and TC, respectively), TP1 responds to the client with a redirect response containing the Include-Referred-Token-Binding-ID header field and directing the client to send a request to TP2. This causes the client to follow the same pattern and send a request containing a TokenBindingMessage which contains both provided and referred TokenBindings (for TP2 and TP1, respectively) to TP2. Note that this pattern can continue to further 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 which 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.

4.4. Negotiated Key Parameters

The TLS Extension for Token Binding Protocol Negotiation [I-D.ietf-tokbind-negotiation] allows the server and client to negotiate the parameters (signature algorithm, length) of the Token Binding key. 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 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 origin that the header field is sent to.

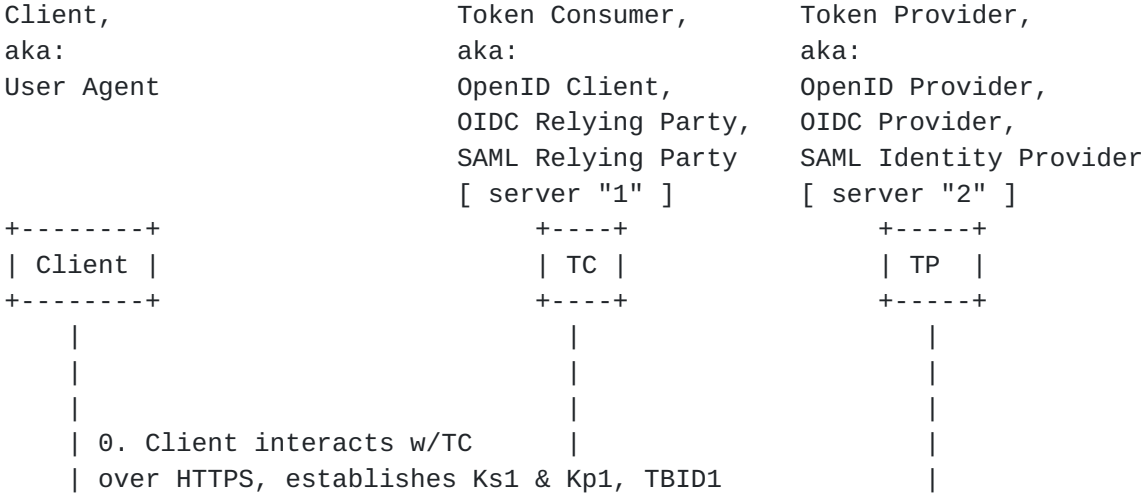
Token Providers SHOULD support all the Token Binding key parameters specified in the [I-D.ietf-tokbind-protocol]. If a token provider does not support the key parameters specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST NOT issue a bound token.

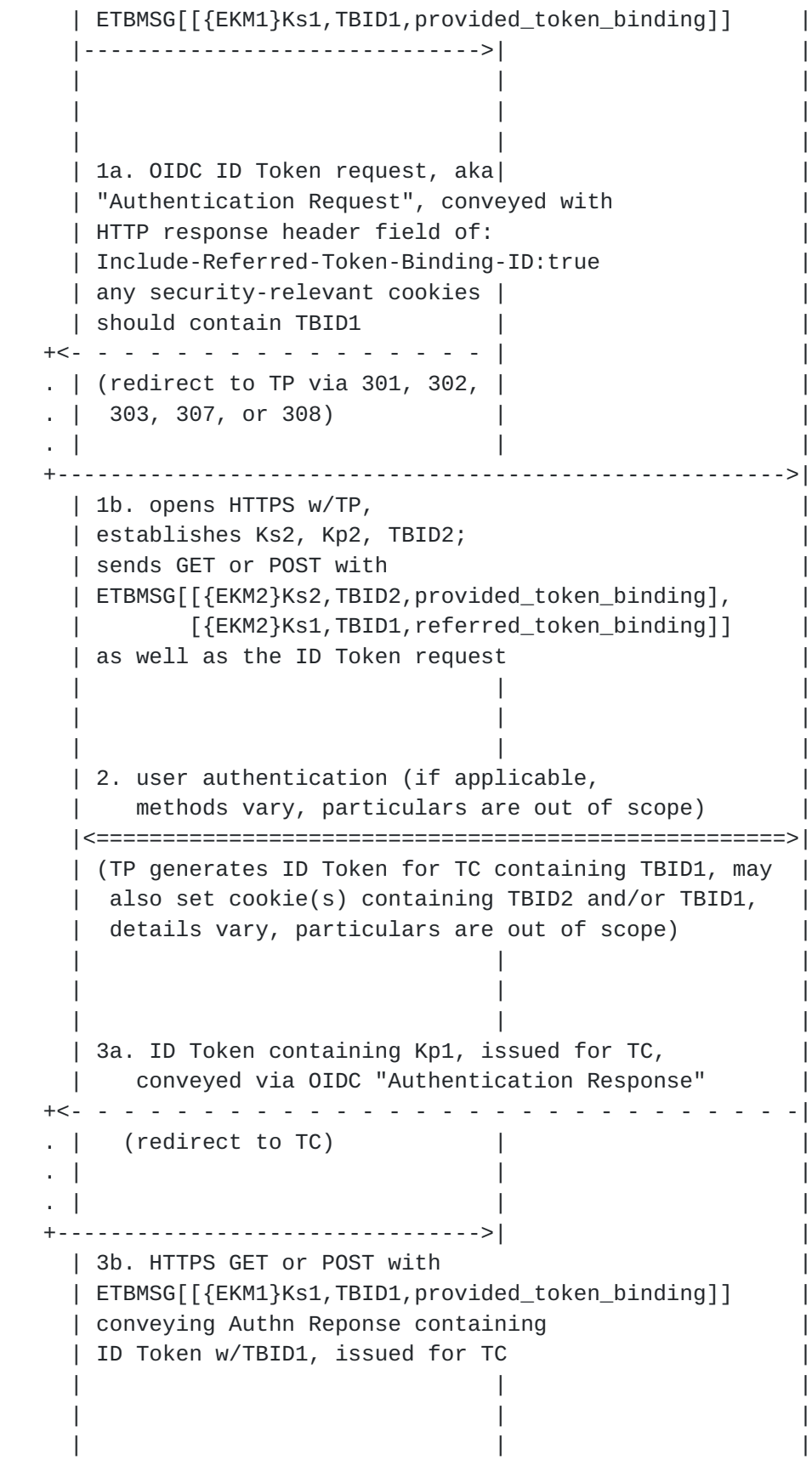
4.5. Federation Example

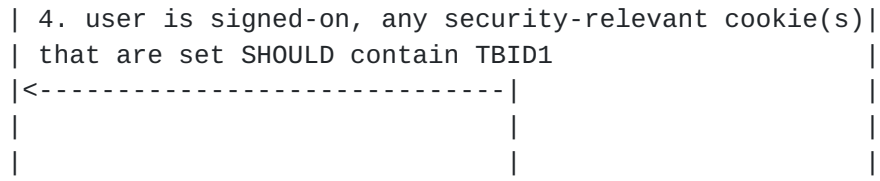
The diagram below shows a typical HTTP Redirect-based Web Browser SSO Profile (no artifact, no callbacks), featuring binding of, e.g., a TLS Token Binding ID into an OpenID Connect "ID Token".

Legend:

EKM:	TLS Exported Keying Material [RFC5705]
{EKMn}Ksm:	EKM for server "n", signed by private key of TBID "m", where "n" must represent 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. E.g., see step 1b in 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 OIDC, it is the semantic equivalent of a SAML "authentication assertion". "ID Token w/TBIDn" denotes a "token bound" ID Token containing TBIDn.
Ks & Kp:	private (aka secret) key, and public key, respectively, of client-side Token Binding key pair
OIDC:	Open ID Connect
TB:	TokenBinding struct containing 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.




| ETBMSG[[{EKM1}Ks1,TBID1,provided_token_binding]] |
|----------->|
| 1a. OIDC ID Token request, aka|
| "Authentication Request", conveyed with |
| 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 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) |
| 3a. ID Token containing Kp1, issued for TC, |
| conveyed via OIDC "Authentication Response" |
+<- -|
. | (redirect to TC) |
. | |
+----------->|
| 3b. HTTPS GET or POST with |
| ETBMSG[[{EKM1}Ks1,TBID1,provided_token_binding]] |
| conveying Authn Reponse containing |
| ID Token w/TBID1, issued for TC |



5. 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 4.3.

Thus, generic Token Binding implementations intended to support any HTTPS-based client-side application (e.g., so-called "native applications"), should provide means for applications to have Token Binding messages, containing Token Binding IDs of various application-specified Token Binding types and for application-specified TLS connections, conveyed over an application-specified HTTPS connection, i.e., within the TokenBindingMessage conveyed by the Sec-Token-Binding header field.

However, such applications MUST only convey Token Binding IDs to other servers if the server associated with a Token Binding ID explicitly signals to do so, e.g., by returning an Include-Referred-Token-Binding-ID HTTP response header field.

NOTE: See Section 7 "Privacy Considerations", for privacy guidance regarding the use of this functionality.

6. Security Considerations

6.1. 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 Token Consumer, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by malware present in the client. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. The Token Binding private key is therefore a high-value asset and MUST be strongly protected, ideally by generating it in a hardware security module that prevents key export.

6.2. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions

The Token Binding protocol relies on the exported key material (EKM) value [RFC5705] to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known vulnerability in TLS 1.2 and older TLS versions, allowing the attacker to synchronize keying material between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated with these TLS versions, unless the Extended Master Secret [RFC7627] TLS extension has also been negotiated. In addition, TLS renegotiation MUST NOT be initiated or allowed, unless the Renegotiation Indication [RFC5746] TLS extension has been negotiated.

6.3. 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's private Token Binding key must be kept secret by the client. If the private key is not secret, then another actor in the system could create a valid Token Binding header field, impersonating 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, he 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 a Sec-Token-Binding header field to a server that contains Token Binding messages about key pairs 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 to send 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 (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 keys 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, A tricked 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 A'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 origin should not be able to set the Sec-Token-Binding header field (through a DOM API or otherwise) that the User Agent uses with another origin. Employing the "Sec-" header field prefix helps to meet this requirement by denoting the header field name to be a "forbidden header name", see [fetch-spec].

6.4. Securing Federated Sign-On Protocols

As explained above, in a federated sign-in scenario a client will prove possession of two different key pairs to a Token Provider: One key pair is the "provided" Token Binding key pair (which the client normally uses with the Token Provider), and the other is the "referred" Token Binding key pair (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 key.

Both proofs (that of the provided Token Binding key and that of the referred Token Binding 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 key.

- The client wants to establish a secure (i.e., free of men-in-the-middle) authenticated session with the Token Consumer, but has not

done so yet (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 client and Token Consumer or between Client and 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 key. Therefore, it can either choose to replace the Token Binding key 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 it can choose to 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 exported keying material (EKM) for the connection between client and man-in-the-middle) will not match the EKM between man-in-the-middle and the Token Provider. If it chooses the former (and creates its own signature, with its own Token Binding key, over the EKM for the connection between man-in-the-middle and Token Provider), then the Token Binding message will match the connection between man-in-the-middle and Token Provider, but the Token Binding key in the message will not match the Token Binding key 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 key possession of the provided Token Binding key is required in the protocol (as we do above).

Next, consider the presence of a man-in-the-middle between client and 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 Issuer to issue a token bound to _its_ Token Binding key, not to the Token Binding key of the

legitimate client. To thwart this goal of the man-in-the-middle, the client's referred Token Binding key must be communicated to the Token Producer in a manner that can not be affected by the man-in-the-middle (who, as we recall, can modify redirect URLs and Javascript at the client). Including the referred Token Binding message in the Sec-Token-Binding header field (as opposed to, say, including the referred Token Binding key in an application-level message as part of the redirect URL) is one way to assure that the man-in-the-middle between client and Token Consumer cannot affect the communication of the referred Token Binding key 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, as well as a proof of possession of the referred Token Binding key.

7. Privacy Considerations

7.1. Scoping of Token Binding Keys

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 ([I-D.ietf-tokbind-protocol] section 4.1).

In the case of HTTP cookies, servers may use Token Binding to secure their cookies. These cookies can be attached to any sub-domain of effective top-level domains, and clients therefore should use the same Token Binding key 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.

7.2. Life Time of Token Binding Keys

Token Binding keys do not have an expiration time. This means that they can potentially be used by a server to track a user across 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 keys (similar to the affordances provided to delete 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 should also discard Token Binding keys from such modes after the session is over. Generally speaking, users should be given the same level of control

over life time of Token Binding keys as they have over cookies or other potential tracking mechanisms.

7.3. 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 either or both web browser-based or native application-based). 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, also having Token Binding IDs for these connections does not provide additional correlation information. If not, then, by providing the other Token Binding IDs, additional information is 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, applications must take care to only reveal Token Binding IDs to other endpoints if the server associated with a Token Binding ID explicitly signals to do so, see Section 5 "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.

8. IANA Considerations

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

Header field name:	Sec-Token-Binding
Applicable protocol:	HTTP
Status:	standard
Author/Change controller:	IETF
Specification document(s):	this one

Header field name:	Include-Referred-Token-Binding-ID
Applicable protocol:	HTTP
Status:	standard
Author/Change controller:	IETF
Specification document(s):	this one

[[TODO: possibly add further considerations wrt the behavior of the above header fields, per <https://tools.ietf.org/html/rfc7231#section-8.3>]]

9. Acknowledgements

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

10. References

10.1. Normative References

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

[I-D.ietf-tokbind-negotiation] Popov, A., Nystrom, M., Balfanz, D., and A. Langley, "Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation", draft-ietf-tokbind-negotiation-03 (work in progress), July 2016.

[I-D.ietf-tokbind-protocol] Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "The Token Binding Protocol Version 1.0", draft-ietf-tokbind-protocol-08 (work in progress), July 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5705>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7541>.

10.2. Informative References

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <http://www.rfc-editor.org/info/rfc5746>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

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

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <http://www.rfc-editor.org/info/rfc7627>.

[TRIPLE-HS] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., and P. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. IEEE Symposium on Security and Privacy", 2014.

Authors' Addresses

Andrei Popov
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz (editor)
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com

Jeff Hodges
Paypal
USA

Email: Jeff.Hodges@paypal.com