

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

A. Popov
M. Nystroem
Microsoft Corp.
D. Balfanz, Ed.
A. Langley
Google Inc.
J. Hodges
Paypal
March 21, 2016

Token Binding over HTTP
draft-ietf-tokbind-https-03

Abstract

This document describes a collection of mechanisms that allow HTTP servers to cryptographically bind authentication tokens (such as cookies and OAuth tokens) to a TLS [RFC5246] connection.

We describe both `_first-party_` as well as `_federated_` scenarios. In a first-party scenario, an HTTP server issues a security token (such as a cookie) to a client, and expects the client to send the security token back to the server at a later time in order to authenticate. Binding the token to the TLS connection between client and server protects the security token from theft, and ensures that the security token can only be used by the client that it was issued to.

Federated token bindings, on the other hand, allow servers to cryptographically bind security tokens to a TLS [RFC5246] 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 [TBPROTO]

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 September 22, 2016.

Copyright Notice

Copyright (c) 2016 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. 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 Language	3
2. The Sec-Token-Binding Header	4
3. Federation Use Cases	4
3.1. Introduction	4
3.2. Overview	5
3.3. HTTP Redirects	6
3.4. Negotiated Key Parameters	7
3.5. Federation Example	7
4. Security Considerations	10
4.1. Security Token Replay	10
4.2. Triple Handshake Vulnerability in TLS	10
4.3. Sensitivity of the Sec-Token-Binding Header	10
4.4. Securing Federated Sign-On Protocols	11
5. Privacy Considerations	13
5.1. Scoping of Token Binding Keys	13
5.2. Life Time of Token Binding Keys	14
6. References	14
6.1. Normative References	14
6.2. Informative References	15
Authors' Addresses	15

1. Introduction

The Token Binding Protocol [TBPROTO] 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 theft by attackers.

While the Token Binding Protocol [TBPROTO] defines a message format for establishing a Token Binding ID, it doesn't 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 [RFC2616] and 2 [I-D.ietf-httpbis-http2]). 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 in HTTP requests. The HTTP header value is a 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

Once a client and server have negotiated the Token Binding Protocol with HTTP/1.1 or HTTP/2 (see The Token Binding Protocol [TBPROTO]), clients MUST include the Sec-Token-Binding header in their HTTP requests. The ABNF of the Sec-Token-Binding header is:

```
Sec-Token-Binding = "Sec-Token-Binding" ":" [CFWS] EncodedTokenBindingMessage
```

The EncodedTokenBindingMessage is a web-safe Base64-encoding of the TokenBindingMessage as defined in the TokenBindingProtocol [TBPROTO].

The TokenBindingMessage MUST contain a TokenBinding with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding 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). The Token Binding ID established by this TokenBinding is called a _Provided Token Binding ID_

In HTTP/2, the client SHOULD use Header Compression [I-D.ietf-httpbis-header-compression] to avoid the overhead of repeating the same header in subsequent HTTP requests.

3. Federation Use Cases

3.1. 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) gives the client permission to 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 [TBPROTO], and includes a TokenBinding structure in the Sec-Token-Binding HTTP header defined above. What differs between the

various mechanisms is how the Token Consumer grants the permission to 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.

3.2. 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:

- o 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.
- o The client delivers the token request to the Token Provider.
- o 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.
- o 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 [TBPROTO], 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.

- o 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 [TBPROTO].

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 permits the client to do so.

Below, we specify how Token Consumers can grant this permission. during redirect-based federation protocols.

3.3. 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-Referer-Token-Binding-ID HTTP response header in its HTTP response. The ABNF of the Include-Referer-Token-Binding-ID header is:

```
Include-Referer-Token-Binding-ID = "Include-Referer-Token-Binding-ID" ":"  
                                     [CFWS] %x74.72.75.65 ; "true", case-sensiti  
ve
```

Including this response header 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, the client will not disclose any information about the Token Binding used between the client and the Token Consumer to the Token Provider.

When a client receives this header, 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 includes one provided binding and one referred binding, the latter constructed from the binding between the client and the Token Consumer.

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

This header 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 following header to the Token Provider with each HTTP request (see above):

Sec-Token-Binding: EncodedTokenBindingMessage

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-Referer-Token-Binding-ID response header). 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 sent to).

3.4. Negotiated Key Parameters

The Token Binding Protocol [TBPROTO] allows the server and client to negotiate a signature algorithm used in the TokenBindingMessage. 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 signature algorithms. The client MUST use the signature algorithm negotiated with the Token Consumer in the referred_token_binding TokenBinding of the TokenBindingMessage, even if that signature algorithm is different from the one negotiated with the origin that the header is sent to.

Token Providers SHOULD support all the SignatureAndHashAlgorithms specified in the Token Binding Protocol [TBPROTO]. If a token provider does not support the SignatureAndHashAlgorithm specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST issue an unbound token.

3.5. 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".

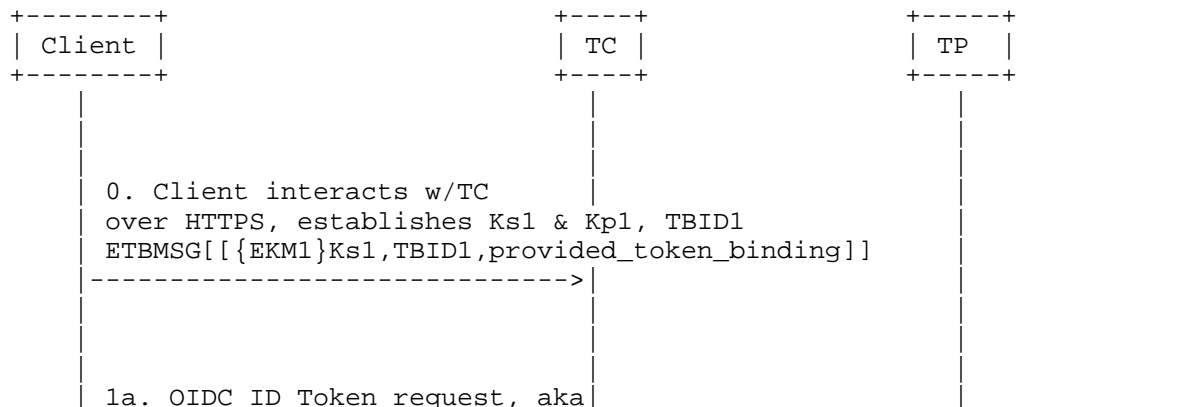
Legend:

EKM:	TLS Exported Keying Material [RFC5705]
{EKM _n }K _{sm} :	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/TBID _n " denotes a "token bound" ID Token containing TBID _n .
K _s & K _p :	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.:
TBID _n :	[{EKM1}Ks1,TBID1,provided_token_binding] Token Binding ID for client and server n's token-bound TLS association. TBID _n contains K _{pn} .

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"]




```

    | "Authentication Request", conveyed with
    | HTTP response header field of:
    | Include-Referer-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
|
|
| 4. user is signed-on, any security-relevant cookie(s)
| that are set SHOULD contain TBID1
|<----->
|

```

4. Security Considerations

4.1. 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.

4.2. Triple Handshake Vulnerability in TLS

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 TLS protocol vulnerability 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 unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

4.3. 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, 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, 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 cookie theft. 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 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, didn't successfully bind the cookie to the client.

Likewise, we need integrity protection of the Sec-Token-Binding header: A client shouldn't be tricked into sending a Sec-Token-Binding header to a server that contains Token Binding messages about key pairs that the client doesn't 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's 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 with A's key pair over C's EKM. If the attacker could now trick C to send such a header to S, it would appear to S as if C controls a certain key pair when in fact it doesn't (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 doesn't 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. One origin should not be able to set the Sec-Token-Binding header (through a DOM API or otherwise) that the User Agent uses with another origin.

4.4. 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:

- o The client has an authentication token with the Token Provider that is bound to the client's Token Binding key.
- o The client wants to establish a secure (i.e., free of man-in-the-middle) authenticated session with the Token Consumer, but hasn't done so yet (in other words, we're about to run the federated sign-on protocol).
- o 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 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 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 (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 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.

5. Privacy Considerations

5.1. Scoping of Token Binding Keys

Clients must use different Token Binding keys for different servers, so as to not allow Token Binding to become a tracking tool across different servers. When Token Binding is used over HTTPS, this key scoping should in particular happen at the granularity of "effective top-level domain (public suffix) + 1", i.e., at the same granularity at which cookies can be set.

The reason for this is that servers may use Token Binding to secure their cookies. These cookies can be attached to any sub-domain of public suffixes, 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.

5.2. Life Time of Token Binding Keys

Token Binding keys don't 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.

6. References

6.1. Normative References

- [I-D.ietf-httpbis-header-compression]
Peon, R. and H. Ruellan, "HPACK - Header Compression for HTTP/2", draft-ietf-httpbis-header-compression-12 (work in progress), February 2015.
- [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>>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<http://www.rfc-editor.org/info/rfc2616>>.
- [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>>.
- [TBPROTO] Popov, A., "The Token Binding Protocol Version 1.0", 2014.

6.2. Informative References

[I-D.ietf-httpbis-http2]

Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-17 (work in progress), February 2015.

[I-D.ietf-tls-session-hash]

Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", draft-ietf-tls-session-hash-06 (work in progress), July 2015.

[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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 28, 2018

A. Popov
M. Nystroem
Microsoft Corp.
D. Balfanz, Ed.
A. Langley
N. Harper
Google Inc.
J. Hodges
PayPal
June 26, 2018

Token Binding over HTTP
draft-ietf-tokbind-https-18

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 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 document is a companion document to The Token Binding Protocol.

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 <https://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 December 28, 2018.

Copyright Notice

Copyright (c) 2018 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 (<https://trustee.ietf.org/license-info>) in effect on the date of 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 Language	3
2. The Sec-Token-Binding HTTP Request Header Field	4
2.1. HTTPS Token Binding Key Pair Scoping	5
3. TLS Renegotiation	6
4. First-Party Use Cases	6
5. Federation Use Cases	7
5.1. Introduction	7
5.2. Overview	8
5.3. HTTP Redirects	10
5.4. Negotiated Key Parameters	12
5.5. Federation Example	12
6. Implementation Considerations	15
7. Security Considerations	15
7.1. Security Token Replay	15
7.2. Sensitivity of the Sec-Token-Binding Header	15
7.3. Securing Federated Sign-On Protocols	17
8. Privacy Considerations	19
8.1. Scoping of Token Binding Key Pairs	19
8.2. Lifetime of Token Binding Key Pairs	20
8.3. Correlation	20
9. IANA Considerations	21
10. Acknowledgements	21
11. References	21
11.1. Normative References	21
11.2. Informative References	23

Authors' Addresses	24
--------------------	----

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 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. I.e., 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 (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 [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.

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", "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 [I-D.ietf-tokbind-protocol] and [I-D.ietf-tokbind-negotiation]), 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 (in [RFC7230] style, 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 [I-D.ietf-tokbind-protocol]. The base64url encoding uses the URL- and filename-safe character set described in Section 5 of [RFC4648], with all trailing padding characters '=' 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 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 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 TokenBindingType 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 use case-specific, and 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 [I-D.ietf-tokbind-protocol]. 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, with the term "public suffix" being defined in a note 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 more narrow. 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, 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. (I.e., 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. (I.e., due to TLS message fragmentation - see Section 6.2.1 of [RFC5246].)

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 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.

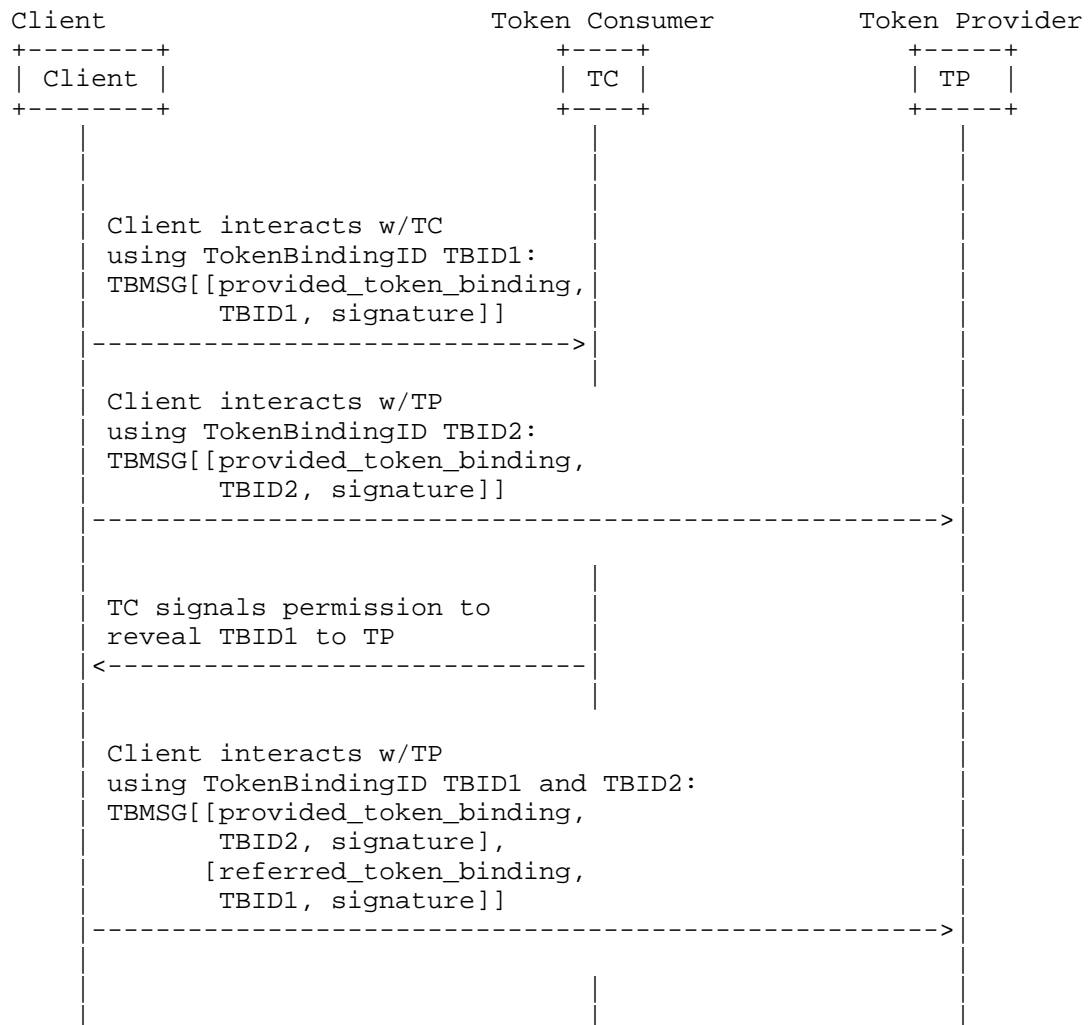
See Section 5 of [I-D.ietf-tokbind-protocol] for general guidance regarding binding of security tokens and their subsequent validation.

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 [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.



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 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:

- o 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.
- o The client delivers the token request to the Token Provider.
- o 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.
- o 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 [I-D.ietf-tokbind-protocol], 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.
- o To allow the Token Provider to include the Token Binding ID 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 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 (in [RFC7230] style, 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 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.

The referred token binding is sent only on 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 a 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 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 a redirection code (300-399), 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 (as described in Section 2 above).

The TokenBindingMessage included in the redirect request to the Token Provider SHOULD contain a TokenBinding with TokenBindingType `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 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 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 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.

5.4. 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 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 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 [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.

5.5. 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]
{EKM _n }K _{sm} :	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 <code>provided_token_binding</code> , then <code>m = n</code> , else if TB's type is <code>referred_token_binding</code> , then <code>m != n</code> . E.g., see step 1b in diagram below.
ETBMSG:	"Sec-Token-Binding" HTTP header field conveying an <code>EncodedTokenBindingMessage</code> , in turn conveying <code>TokenBinding (TB)struct(s)</code> , e.g.: <code>ETBMSG[[TB]]</code> or <code>ETBMSG[[TB1],[TB2]]</code>
ID Token:	the ID Token in OpenID Connect, it is the semantic equivalent of a SAML "authentication assertion". "ID Token w/TBID _n " denotes a "token bound" ID Token containing TBID _n .
K _s & K _p :	private (aka secret) key, and public key, respectively, of client-side Token Binding key pair
OIDC:	OpenID Connect
TB:	<code>TokenBinding struct</code> containing signed EKM, TBID, and TB type, e.g.:
TBID _n :	<code>[{EKM1}Ks1,TBID1,provided_token_binding]</code> Token Binding ID for client and server n's token-bound TLS association. TBID _n contains K _{pn} .

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 K_{s1} & K_{p1}, TBID₁
`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 Response containing
    | ID Token w/TBID1, issued for TC
    |
    | 4. user is signed-on, any security-relevant cookie(s)
    |     that are set SHOULD contain TBID1
    | <-----

```

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. For example, a server can return an Include-Referred-Token-Binding-ID HTTP response header field to an application, which then signals to the Token Binding implementation that it intends to convey the Token Binding ID used with this server to another server. Other signaling mechanisms are possible, and are specific to the application layer protocol, but 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 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 by 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 Security Token Replay security consideration laid out in the The Token Binding Protocol [I-D.ietf-tokbind-protocol] specification.

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's Token Binding private 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 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 (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, 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 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 DOM API 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 to be a "forbidden header name", see [fetch-spec].

7.3. Securing Federated Sign-On Protocols

As explained above, in a federated sign-in 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:

- o 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.
- o The client wants to establish a secure (i.e., free of man-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).
- o 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 private key. Therefore, it can either choose to 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 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 a signature on the EKM between 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 Token Provider), then the Token Binding message will match the connection between man-in-the-middle and 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 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 Provider into issuing a token bound to its Token Binding ID, 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 we recall, can modify redirect URLs and Javascript at the client). Including the referred Token Binding 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 client and 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, as well as 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 federated sign-on protocols. These include the following:

- o 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.)
- o The Token Consumer should verify that the tokens were issued for it, and not 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 [I-D.ietf-tokbind-protocol]).

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 (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 role of Relying Party and Identity Provider 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-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 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 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 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 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, Token Binding implementations must take care to only reveal Token Binding IDs to other endpoints if the application associated with a Token Binding ID signals to do so, 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 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

10. 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.

11. References

11.1. Normative References

[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-14 (work in progress), May 2018.

- [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-19 (work in progress), May 2018.
- [PSL] Mozilla, "Public Suffix List, <https://publicsuffix.org/>", <<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>>.

11.2. Informative References

- [fetch-spec]
WhatWG, "Fetch", Living Standard ,
<<https://fetch.spec.whatwg.org/>>.
- [I-D.ietf-tokbind-tls13]
Harper, N., "Token Binding for Transport Layer Security (TLS) Version 1.3 Connections", draft-ietf-tokbind-tls13-01 (work in progress), May 2018.
- [OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol 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>>.
- [OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", August 2015, <http://openid.net/specs/openid-connect-core-1_0.html>.
- [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, <<https://www.rfc-editor.org/info/rfc5746>>.
- [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>>.

[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, <<https://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

Nick Harper
Google Inc.
USA

Email: nharper@google.com

Jeff Hodges
PayPal
USA

Email: Jeff.Hodges@paypal.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: July 11, 2016

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.
January 8, 2016

Transport Layer Security (TLS) Extension for Token Binding Protocol
Negotiation
draft-ietf-tokbind-negotiation-02

Abstract

This document specifies a Transport Layer Security (TLS) [RFC5246] extension for the negotiation of Token Binding protocol [I-D.ietf-tokbind-protocol] version and key parameters.

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 July 11, 2016.

Copyright Notice

Copyright (c) 2016 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. 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	2
1.1. Requirements Language	2
2. Token Binding Negotiation Client Hello Extension	2
3. Token Binding Negotiation Server Hello Extension	3
4. Negotiating Token Binding Protocol Version and Key Parameters	4
5. IANA Considerations	5
6. Security Considerations	6
6.1. Downgrade Attacks	6
6.2. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions	6
7. Acknowledgements	6
8. References	7
8.1. Normative References	7
8.2. Informative References	7
Authors' Addresses	8

1. Introduction

In order to use the Token Binding protocol [I-D.ietf-tokbind-protocol], the client and server need to agree on the Token Binding protocol version and the parameters (signature algorithm, length) of the Token Binding key. This document specifies a new TLS extension to accomplish this negotiation without introducing additional network round-trips.

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. Token Binding Negotiation Client Hello Extension

The client uses the "token_binding" TLS extension to indicate the highest supported Token Binding protocol version and key parameters.

```
enum {
    token_binding(TBD), (65535)
} ExtensionType;
```

The "extension_data" field of this extension contains a "TokenBindingParameters" value.

```

struct {
    uint8 major;
    uint8 minor;
} ProtocolVersion;

enum {
    rsa2048_pkcs1.5(0), rsa2048_pss(1), ecdsap256(2), (255)
} TokenBindingKeyParameters;

struct {
    ProtocolVersion token_binding_version;
    TokenBindingKeyParameters key_parameters_list<1..2^8-1>
} TokenBindingParameters;

```

"token_binding_version" indicates the version of the Token Binding protocol the client wishes to use during this connection. This SHOULD be the latest (highest valued) version supported by the client. [I-D.ietf-tokbind-protocol] describes version {1, 0} of the protocol. Prototype implementations of Token Binding drafts can indicate support of a specific draft version, e.g. {0, 1} or {0, 2}.

"key_parameters_list" contains the list of identifiers of the Token Binding key parameters supported by the client, in descending order of preference.

3. Token Binding Negotiation Server Hello Extension

The server uses the "token_binding" TLS extension to indicate support for the Token Binding protocol and to select the protocol version and key parameters.

The server that supports Token Binding and receives a client hello message containing the "token_binding" extension, will include the "token_binding" extension in the server hello if all of the following conditions are satisfied:

1. The server supports the Token Binding protocol version offered by the client or a lower version.
2. The server finds acceptable Token Binding key parameters on the client's list.
3. The server is also negotiating Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions. This requirement only applies when TLS 1.2 or an older TLS version is used (see security considerations section below for more details).

The server will ignore any key parameters that it does not recognize. The "extension_data" field of the "token_binding" extension is structured the same as described above for the client "extension_data".

"token_binding_version" contains the lower of the Token Binding protocol version offered by the client in the "token_binding" extension and the highest version supported by the server.

"key_parameters_list" contains exactly one Token Binding key parameters identifier selected by the server from the client's list.

4. Negotiating Token Binding Protocol Version and Key Parameters

It is expected that a server will have a list of Token Binding key parameters identifiers that it supports, in preference order. The server MUST only select an identifier that the client offered. The server SHOULD select the most highly preferred key parameters identifier it supports which is also advertised by the client. In the event that the server supports none of the key parameters that the client advertises, then the server MUST NOT include "token_binding" extension in the server hello.

The client receiving the "token_binding" extension MUST terminate the handshake with a fatal "unsupported_extension" alert if any of the following conditions are true:

1. The client did not include the "token_binding" extension in the client hello.
2. "token_binding_version" is higher than the Token Binding protocol version advertised by the client.
3. "key_parameters_list" includes more than one Token Binding key parameters identifier.
4. "key_parameters_list" includes an identifier that was not advertised by the client.
5. TLS 1.2 or an older TLS version is used, but Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions are not negotiated (see security considerations section below for more details).

If the "token_binding" extension is included in the server hello and the client supports the Token Binding protocol version selected by the server, it means that the version and key parameters have been negotiated between the client and the server and SHALL be definitive

for the TLS connection. In this case, the client MUST use the negotiated key parameters in the "provided_token_binding" as described in [I-D.ietf-tokbind-protocol].

If the client does not support the Token Binding protocol version selected by the server, then the connection proceeds without Token Binding.

Please note that the Token Binding protocol version and key parameters are negotiated for each TLS connection, which means that the client and server include their "token_binding" extensions both in the full TLS handshake that establishes a new TLS session and in the subsequent abbreviated TLS handshakes that resume the TLS session.

5. IANA Considerations

This document defines a new TLS extension "token_binding", which needs to be added to the IANA "Transport Layer Security (TLS) Extensions" registry.

This document establishes a registry for identifiers of Token Binding key parameters entitled "Token Binding Key Parameters" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies a set of Token Binding key parameters (0-255).
- o Description: The description of the Token Binding key parameters.
- o Specification: A reference to a specification that defines the Token Binding key parameters.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified set of Token Binding key parameters.

An initial set of registrations for this registry follows:

Value: 0

Description: rsa2048_pkcs1.5

Specification: this document

Value: 1

Description: rsa2048_pss

Specification: this document

Value: 2

Description: ecdsap256

Specification: this document

6. Security Considerations

6.1. Downgrade Attacks

The Token Binding protocol version and key parameters are negotiated via "token_binding" extension within the TLS handshake. TLS prevents active attackers from modifying the messages of the TLS handshake, therefore it is not possible for the attacker to remove or modify the "token_binding" extension. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via "token_binding" extension.

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

The Token Binding protocol relies on the TLS Exporters [RFC5705] to associate a TLS connection with a Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize exported 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] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

7. Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam, Bill Cox, Nick Harper and others.

8. References

8.1. Normative References

- [I-D.ietf-tokbind-protocol]
Popov, A., Nystrom, M., Balfanz, D., and A. Langley, "The Token Binding Protocol Version 1.0", draft-ietf-tokbind-protocol-03 (work in progress), October 2015.
- [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>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [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>>.
- [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>>.
- [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>>.

8.2. Informative References

- [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 (editor)
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: November 24, 2018

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.
May 23, 2018

Transport Layer Security (TLS) Extension for Token Binding Protocol
Negotiation
draft-ietf-tokbind-negotiation-14

Abstract

This document specifies a Transport Layer Security (TLS) extension for the negotiation of Token Binding protocol version and key parameters.

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 <https://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 November 24, 2018.

Copyright Notice

Copyright (c) 2018 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 (<https://trustee.ietf.org/license-info>) in effect on the date of 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	2
1.1. Requirements Language	2
2. Token Binding Negotiation Client Hello Extension	2
3. Token Binding Negotiation Server Hello Extension	4
4. Negotiating Token Binding Protocol Version and Key Parameters	4
5. IANA Considerations	6
6. Security Considerations	6
6.1. Downgrade Attacks	6
6.2. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions	6
7. Acknowledgements	7
8. References	7
8.1. Normative References	7
8.2. Informative References	8
Authors' Addresses	8

1. Introduction

In order to use the Token Binding protocol [I-D.ietf-tokbind-protocol], the client and server need to agree on the Token Binding protocol version and the parameters (signature algorithm, length) of the Token Binding key. This document specifies a new TLS [RFC5246] extension to accomplish this negotiation without introducing additional network round-trips in TLS 1.2 and earlier versions. The negotiation of the Token Binding protocol and key parameters in combination with TLS 1.3 and later versions is beyond the scope of this document.

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. Token Binding Negotiation Client Hello Extension

The client uses the "token_binding" TLS extension to indicate the highest supported Token Binding protocol version and key parameters.

```
enum {  
    token_binding(24), (65535)  
} ExtensionType;
```

The "extension_data" field of this extension contains a "TokenBindingParameters" value.

```
struct {  
    uint8 major;  
    uint8 minor;  
} TB_ProtocolVersion;
```

```
enum {  
    rsa2048_pkcs1.5(0), rsa2048_pss(1), ecdsap256(2), (255)  
} TokenBindingKeyParameters;
```

```
struct {  
    TB_ProtocolVersion token_binding_version;  
    TokenBindingKeyParameters key_parameters_list<1..2^8-1>  
} TokenBindingParameters;
```

"token_binding_version" indicates the version of the Token Binding protocol the client wishes to use during this connection. If the client supports multiple Token Binding protocol versions, it SHOULD indicate the latest supported version (the one with the highest TB_ProtocolVersion.major and TB_ProtocolVersion.minor) in TokenBindingParameters.token_binding_version. E.g. if the client supports versions {1, 0} and {0, 13} of the Token Binding protocol, it SHOULD indicate version {1, 0}. Please note that the server MAY select any lower protocol version, see Section 3 "Token Binding Negotiation Server Hello Extension" for more details. If the client does not support the Token Binding protocol version selected by the server, then the connection proceeds without Token Binding. [I-D.ietf-tokbind-protocol] describes version {1, 0} of the protocol.

Please note that the representation of the Token Binding protocol version using two octets ("major" and "minor") is for human convenience only and carries no protocol significance.

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH: Prototype implementations of Token Binding drafts can indicate support of a specific draft version, e.g. {0, 1} or {0, 2}.

"key_parameters_list" contains the list of identifiers of the Token Binding key parameters supported by the client, in descending order of preference. [I-D.ietf-tokbind-protocol] establishes an IANA registry for Token Binding key parameter identifiers.

3. Token Binding Negotiation Server Hello Extension

The server uses the "token_binding" TLS extension to indicate support for the Token Binding protocol and to select the protocol version and key parameters.

The server that supports Token Binding and receives a client hello message containing the "token_binding" extension will include the "token_binding" extension in the server hello if all of the following conditions are satisfied:

1. The server supports the Token Binding protocol version offered by the client or a lower version.
2. The server finds acceptable Token Binding key parameters on the client's list.
3. The server is also negotiating the Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions. This requirement applies when TLS 1.2 or an older TLS version is used (see Section 6 "Security Considerations" below for more details).

The server will ignore any key parameters that it does not recognize. The "extension_data" field of the "token_binding" extension is structured the same as described above for the client "extension_data".

"token_binding_version" contains the lower of:

- o the Token Binding protocol version offered by the client in the "token_binding" extension and
- o the highest Token Binding protocol version supported by the server.

"key_parameters_list" contains exactly one Token Binding key parameters identifier selected by the server from the client's list.

4. Negotiating Token Binding Protocol Version and Key Parameters

It is expected that a server will have a list of Token Binding key parameters identifiers that it supports, in preference order. The server MUST only select an identifier that the client offered. The server SHOULD select the most highly preferred key parameters identifier it supports which is also advertised by the client. In the event that the server supports none of the key parameters that

the client advertises, then the server MUST NOT include the "token_binding" extension in the server hello.

The client receiving the "token_binding" extension MUST terminate the handshake with a fatal "unsupported_extension" alert if any of the following conditions are true:

1. The client did not include the "token_binding" extension in the client hello.
2. "token_binding_version" is higher than the Token Binding protocol version advertised by the client.
3. "key_parameters_list" includes more than one Token Binding key parameters identifier.
4. "key_parameters_list" includes an identifier that was not advertised by the client.
5. TLS 1.2 or an older TLS version is used, but the Extended Master Secret [RFC7627] and TLS Renegotiation Indication [RFC5746] extensions are not negotiated (see Section 6 "Security Considerations" below for more details).

If the "token_binding" extension is included in the server hello and the client supports the Token Binding protocol version selected by the server, it means that the version and key parameters have been negotiated between the client and the server and SHALL be definitive for the TLS connection. TLS 1.2 and earlier versions support renegotiation, allowing the client and server to renegotiate the Token Binding protocol version and key parameters on the same connection. The client MUST use the negotiated key parameters in the "provided_token_binding" as described in [I-D.ietf-tokbind-protocol].

If the client does not support the Token Binding protocol version selected by the server, then the connection proceeds without Token Binding. There is no requirement for the client to support any Token Binding versions other than the one advertised in the client's "token_binding" extension.

Client and server applications can choose to handle failure to negotiate Token Binding in a variety of ways, e.g.: continue using the connection as usual, shorten the lifetime of tokens issued during this connection, require stronger authentication, terminate the connection, etc.

The Token Binding protocol version and key parameters are negotiated for each TLS connection, which means that the client and server

include their "token_binding" extensions both in the full TLS handshake that establishes a new TLS session and in the subsequent abbreviated TLS handshakes that resume the TLS session.

5. IANA Considerations

This document updates the TLS "ExtensionType Values" registry. IANA has provided the following temporary registration for the "token_binding" TLS extension:

Value: 24

Extension name: token_binding

Reference: this document

Recommended: Yes

IANA is requested to make this registration permanent, keeping the value of 24, which has been used by the prototype implementations of the Token Binding protocol.

This document uses "Token Binding Key Parameters" registry originally created in [I-D.ietf-tokbind-protocol]. This document creates no new registrations in this registry.

6. Security Considerations

6.1. Downgrade Attacks

The Token Binding protocol version and key parameters are negotiated via the "token_binding" extension within the TLS handshake. TLS detects handshake message modification by active attackers, therefore it is not possible for an attacker to remove or modify the "token_binding" extension without breaking the TLS handshake. The signature algorithm and key length used in the Token Binding of type "provided_token_binding" MUST match the parameters negotiated via the "token_binding" extension.

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

The Token Binding protocol relies on the TLS Exporters [RFC5705] to associate a TLS connection with a Token Binding. The triple handshake attack [TRIPLE-HS] is a known vulnerability in TLS 1.2 and older TLS versions, allowing an 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] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

7. 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, Benjamin Kaduk, Alexey Melnikov 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.

8. References

8.1. Normative References

- [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-18 (work in progress), May 2018.
- [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>>.
- [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>>.
- [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, <<https://www.rfc-editor.org/info/rfc5746>>.
- [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, <<https://www.rfc-editor.org/info/rfc7627>>.

[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>>.

8.2. Informative References

[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 (editor)
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: July 11, 2016

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.
January 8, 2016

The Token Binding Protocol Version 1.0
draft-ietf-tokbind-protocol-04

Abstract

This document specifies Version 1.0 of the Token Binding protocol. The Token Binding protocol allows client/server applications to create long-lived, uniquely identifiable TLS [RFC5246] bindings spanning multiple TLS sessions and connections. Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks. To protect privacy, the TLS Token Binding identifiers are only transmitted encrypted and can be reset by the user at any time.

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 July 11, 2016.

Copyright Notice

Copyright (c) 2016 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. 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	2
1.1. Requirements Language	3
2. Token Binding Protocol Overview	3
3. Token Binding Protocol Message	4
4. Establishing a TLS Token Binding	7
5. TLS Token Binding ID Format	7
6. Security Token Validation	8
7. IANA Considerations	8
8. Security Considerations	10
8.1. Security Token Replay	10
8.2. Downgrade Attacks	10
8.3. Privacy Considerations	10
8.4. Token Binding Key Sharing Between Applications	11
8.5. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions	11
9. Acknowledgements	11
10. References	11
10.1. Normative References	11
10.2. Informative References	13
Authors' Addresses	13

1. Introduction

Servers generate various security tokens (e.g. HTTP cookies, OAuth tokens) for applications to access protected resources. Any party in possession of such token gains access to the protected resource. Attackers export bearer tokens from the user's machine, present them to the servers, and impersonate authenticated users. The idea of Token Binding is to prevent such attacks by cryptographically binding security tokens to the TLS layer.

A TLS Token Binding is established by the user agent generating a private-public key pair (possibly within a secure hardware module, such as TPM) per target server, and proving possession of the private key on every TLS connection to the target server. The proof of possession involves signing the exported keying material [RFC5705] for the TLS connection with the private key. The corresponding public key is included in the TLS Token Binding identifier structure (described in the "TLS Token Binding ID Format" section of this

document). TLS Token Bindings are long-lived, i.e. they encompass multiple TLS connections and TLS sessions between a given client and server. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies.

When issuing a security token to a client that supports TLS Token Binding, a server includes the client's TLS Token Binding ID in the token. Later on, when a client presents a security token containing a TLS Token Binding ID, the server makes sure the ID in the token matches the ID of the TLS Token Binding established with the client. In the case of a mismatch, the server discards the token.

In order to successfully export and replay a bound security token, the attacker needs to also be able to export the client's private key, which is hard to do in the case of the key generated in a secure hardware module.

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. Token Binding Protocol Overview

The client and server use the Token Binding Negotiation TLS Extension [I-D.ietf-tokbind-negotiation] to negotiate the Token Binding protocol version and the parameters (signature algorithm, length) of the Token Binding key. This negotiation does not require additional round-trips.

The Token Binding protocol consists of one message sent by the client to the server, proving possession of one or more client-generated asymmetric keys. This message is only sent if the client and server agree on the use of the Token Binding protocol and the key parameters. The Token Binding message is sent with the application protocol data in TLS `application_data` records.

A server receiving the Token Binding message verifies that the key parameters in the message match the Token Binding parameters negotiated via [I-D.ietf-tokbind-negotiation], and then validates the signatures contained in the Token Binding message. If either of these checks fails, the server terminates the connection, otherwise the TLS Token Binding is successfully established with the ID contained in the Token Binding message.

When a server supporting the Token Binding protocol receives a bound token, the server compares the TLS Token Binding ID in the security token with the TLS Token Binding ID established with the client. If the bound token came from a TLS connection without a Token Binding, or if the IDs don't match, the token is discarded.

This document defines the format of the Token Binding protocol message, the process of establishing a TLS Token Binding, the format of the Token Binding ID, and the process of validating a security token. Token Binding Negotiation TLS Extension [I-D.ietf-tokbind-negotiation] describes the negotiation of the Token Binding protocol and key parameters. Token Binding over HTTP [I-D.ietf-tokbind-https] explains how the Token Binding message is encapsulated within HTTP/1.1 [RFC7230] or HTTP/2 [RFC7540] messages. [I-D.ietf-tokbind-https] also describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

3. Token Binding Protocol Message

The Token Binding message is sent by the client and proves possession of one or more private keys held by the client. This message **MUST** be sent if the client and server successfully negotiated the use of the Token Binding protocol via [I-D.ietf-tokbind-negotiation], and **MUST NOT** be sent otherwise. This message **MUST** be sent in the client's first application protocol message. This message **MAY** also be sent in subsequent application protocol messages, proving possession of other keys by the same client, to facilitate token binding between more than two communicating parties. Token Binding over HTTP [I-D.ietf-tokbind-https] specifies the encapsulation of the Token Binding message in the application protocol messages, and the scenarios involving more than two communicating parties. The Token Binding message format is defined using TLS specification language:

```

enum {
    rsa2048_pkcs1.5(0), rsa2048_pss(1), ecdsap256(2), (255)
} TokenBindingKeyParameters;

struct {
    opaque modulus<1..2^16-1>;
    opaque publicexponent<1..2^8-1>;
} RSAPublicKey;

struct {
    opaque point <1..2^8-1>;
} ECPoint;

enum {
    provided_token_binding(0), referred_token_binding(1), (255)
} TokenBindingType;

struct {
    TokenBindingKeyParameters key_parameters;
    select (key_parameters) {
        case rsa2048_pkcs1.5:
        case rsa2048_pss:
            RSAPublicKey rsapubkey;
        case ecdsap256:
            ECPoint point;
    }
} TokenBindingID;

enum {
    (255)                                     // No initial ExtensionType registrations
} ExtensionType;

struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    TokenBindingType tokenbinding_type;
    TokenBindingID tokenbindingid;
    opaque signature<0..2^16-1>; // Signature over the exported keying material value
    Extension extensions<0..2^16-1>;
} TokenBinding;

struct {
    TokenBinding tokenbindings<0..2^16-1>;
} TokenBindingMessage;

```

The Token Binding message consists of a series of TokenBinding structures containing the type of the token binding, the TokenBindingID, a signature over the exported keying material (EKM) value, optionally followed by Extension structures.

This document defines two token binding types: `provided_token_binding` used to establish a Token Binding when connecting to a server, and `referred_token_binding` used when requesting tokens to be presented to a different server. Token Binding over HTTP [I-D.ietf-tokbind-https] describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

When an `rsa2048_pkcs1.5` or `rsa2048_pss` key is used, `TokenBinding.signature` contains the signature generated using, respectively, the RSASSA-PKCS1-v1_5 or RSASSA-PSS signature scheme defined in [RFC3447]. `RSAPublicKey.modulus` and `RSAPublicKey.publicexponent` contain the length-prefixed modulus and exponent of the RSA public key represented in big-endian format.

When an `ecdsap256` key is used, `TokenBinding.signature` contains a pair of integers, R followed by S, as defined in [ANSI.X9-62.2005]. R and S are encoded in big-endian format. `ECPoint.point` contains the X coordinate followed by the Y coordinate. The X and Y coordinates are unsigned integers encoded in big-endian format. Future specifications may define Token Binding keys using other elliptic curves with their corresponding signature and point formats.

The EKM is obtained using the Keying Material Exporters for TLS defined in [RFC5705], by supplying the following input values:

- o Label: The ASCII string "EXPORTER-Token-Binding" with no terminating NUL.
- o Context value: NULL (no application context supplied).
- o Length: 32 bytes.

An implementation MUST ignore any unknown extensions. Initially, no extension types are defined. One of the possible uses of extensions envisioned at the time of this writing is attestation: cryptographic proof that allows the server to verify that the Token Binding key is hardware-bound. The definitions of such Token Binding protocol extensions are outside the scope of this specification.

At least one TokenBinding MUST be included in the Token Binding message. The signature algorithm and key length used in the TokenBinding MUST match the parameters negotiated via [I-D.ietf-tokbind-negotiation]. The client SHOULD generate and store

Token Binding keys in a secure manner that prevents key export. In order to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

4. Establishing a TLS Token Binding

The triple handshake vulnerability in TLS 1.2 and older TLS versions affects the security of the Token Binding protocol, as described in the "Security Considerations" section below. Therefore, the server MUST NOT negotiate the use of the Token Binding protocol with these TLS versions, unless the server also negotiates Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions.

The server MUST terminate the connection if the use of the Token Binding protocol was not negotiated, but the client sends the Token Binding message. If the Token Binding type is "provided_token_binding", the server MUST verify that the signature algorithm (including elliptic curve in the case of ECDSA) and key length in the Token Binding message match those negotiated via [I-D.ietf-tokbind-negotiation]. In the case of a mismatch, the server MUST terminate the connection. As described in [I-D.ietf-tokbind-https], Token Bindings of type "referred_token_binding" may have different key parameters than those negotiated via [I-D.ietf-tokbind-negotiation].

If the Token Binding message does not contain at least one TokenBinding structure, or the signature contained in a TokenBinding structure is invalid, the server MUST terminate the connection. Otherwise, the TLS Token Binding is successfully established and its ID can be provided to the application for security token validation.

5. TLS Token Binding ID Format

The ID of the TLS Token Binding established as a result of Token Binding message processing is a binary representation of the following structure:

```
struct {  
    TokenBindingKeyParameters key_parameters;  
    select (key_parameters) {  
        case rsa2048_pkcs1.5:  
        case rsa2048_pss:  
            RSAPublicKey rsapubkey;  
        case ecdsap256:  
            ECPoint point;  
    }  
} TokenBindingID;
```

TokenBindingID contains the key parameters negotiated via [I-D.ietf-tokbind-negotiation]. TLS Token Binding ID can be obtained from the TokenBinding structure described in the "Token Binding Protocol Message" section of this document by discarding the token binding type, signature and extensions. TLS Token Binding ID will be available at the application layer and used by the server to generate and verify bound tokens.

6. Security Token Validation

Security tokens can be bound to the TLS layer either by embedding the Token Binding ID in the token, or by maintaining a database mapping tokens to Token Binding IDs. The specific method of generating bound security tokens is application-defined and beyond the scope of this document.

Upon receipt of a security token, the server attempts to retrieve TLS Token Binding ID information from the token and from the TLS connection with the client. Application-provided policy determines whether to honor non-bound (bearer) tokens. If the token is bound and a TLS Token Binding has not been established for the client connection, the server MUST discard the token. If the TLS Token Binding ID for the token does not match the TLS Token Binding ID established for the client connection, the server MUST discard the token.

7. IANA Considerations

This document establishes a registry for Token Binding type identifiers entitled "Token Binding Types" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies the Token Binding type (0-255).
- o Description: The description of the Token Binding type.
- o Specification: A reference to a specification that defines the Token Binding type.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding type.

An initial set of registrations for this registry follows:

Value: 0

Description: provided_token_binding

Specification: this document

Value: 1

Description: referred_token_binding

Specification: this document

This document establishes a registry for Token Binding extensions entitled "Token Binding Extensions" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies the Token Binding extension (0-255).
- o Description: The description of the Token Binding extension.
- o Specification: A reference to a specification that defines the Token Binding extension.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable

implementations using the identified Token Binding extension. This document creates no initial registrations in the "Token Binding Extensions" registry.

This document uses "Token Binding Key Parameters" registry originally created in [I-D.ietf-tokbind-negotiation]. This document creates no new registrations in this registry.

8. Security Considerations

8.1. Security Token Replay

The goal of the Token Binding protocol is to prevent attackers from exporting and replaying security tokens, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by the malware present in the User Agent. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. 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.

8.2. Downgrade Attacks

The Token Binding protocol is only used when negotiated via [I-D.ietf-tokbind-negotiation] within the TLS handshake. TLS prevents active attackers from modifying the messages of the TLS handshake, therefore it is not possible for the attacker to remove or modify the Token Binding Negotiation TLS Extension used to negotiate the Token Binding protocol and key parameters. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via [I-D.ietf-tokbind-negotiation].

8.3. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. Some applications offer a special privacy mode where they don't store or use tokens supplied by the server, e.g. "in private" browsing. When operating in this special privacy mode, applications SHOULD use newly generated Token Binding keys and delete them when exiting this mode, or else SHOULD NOT negotiate Token Binding at all.

In order to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

A server can use tokens and Token Binding IDs to track clients. Client applications that automatically limit the lifetime of tokens to maintain user privacy SHOULD apply the same validity time limits to Token Binding keys.

8.4. Token Binding Key Sharing Between Applications

Existing systems provide a variety of platform-specific mechanisms for certain applications to share tokens, e.g. to enable single sign-on scenarios. For these scenarios to keep working with bound tokens, the applications that are allowed to share tokens will need to also share Token Binding keys. Care must be taken to restrict the sharing of Token Binding keys to the same group(s) of applications that share the same tokens.

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

The Token Binding protocol relies on the exported keying material (EKM) to associate a TLS connection with a 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] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

9. Acknowledgements

This document incorporates comments and suggestions offered by Eric Rescorla, Gabriel Montenegro, Martin Thomson, Vinod Anupam, Bill Cox, Nick Harper and others.

10. References

10.1. Normative References

[ANSI.X9-62.2005]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 2005.

- [I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., and A. Langley,
"Token Binding over HTTP", draft-ietf-tokbind-https-02
(work in progress), October 2015.
- [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-01 (work in progress), October 2015.
- [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>>.
- [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography
Standards (PKCS) #1: RSA Cryptography Specifications
Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February
2003, <<http://www.rfc-editor.org/info/rfc3447>>.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B.
Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites
for Transport Layer Security (TLS)", RFC 4492,
DOI 10.17487/RFC4492, May 2006,
<<http://www.rfc-editor.org/info/rfc4492>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
IANA Considerations Section in RFCs", BCP 26, RFC 5226,
DOI 10.17487/RFC5226, May 2008,
<<http://www.rfc-editor.org/info/rfc5226>>.
- [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>>.
- [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>>.

- [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>>.
- [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>>.

10.2. Informative References

- [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 (editor)
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: November 24, 2018

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.
J. Hodges
PayPal
May 23, 2018

The Token Binding Protocol Version 1.0
draft-ietf-tokbind-protocol-19

Abstract

This document specifies Version 1.0 of the Token Binding protocol. The Token Binding protocol allows client/server applications to create long-lived, uniquely identifiable TLS bindings spanning multiple TLS sessions and connections. Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks. To protect privacy, the Token Binding identifiers are only conveyed over TLS and can be reset by the user at any time.

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 <https://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 November 24, 2018.

Copyright Notice

Copyright (c) 2018 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 (<https://trustee.ietf.org/license-info>) in effect on the date of 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	2
1.1. Requirements Language	3
2. Token Binding Protocol Overview	3
3. Token Binding Protocol Message	4
3.1. TokenBinding.tokenbinding_type	6
3.2. TokenBinding.tokenbindingid	7
3.3. TokenBinding.signature	7
3.4. TokenBinding.extensions	8
4. Establishing a Token Binding	9
4.1. Client Processing Rules	9
4.2. Server Processing Rules	9
5. Bound Security Token Creation and Validation	10
6. IANA Considerations	11
6.1. Token Binding Key Parameters Registry	11
6.2. Token Binding Types Registry	12
6.3. Token Binding Extensions Registry	13
6.4. Registration of Token Binding TLS Exporter Label	13
7. Security Considerations	14
7.1. Security Token Replay	14
7.2. Downgrade Attacks	14
7.3. Privacy Considerations	14
7.4. Token Binding Key Sharing Between Applications	15
7.5. Triple Handshake Vulnerability in TLS 1.2 and Older TLS Versions	15
8. Acknowledgements	15
9. References	15
9.1. Normative References	16
9.2. Informative References	17
Authors' Addresses	17

1. Introduction

Servers often generate various security tokens (e.g. HTTP cookies, OAuth [RFC6749] tokens) for applications to present when accessing protected resources. In general, any party in possession of bearer security tokens gain access to certain protected resource(s).

Attackers take advantage of this by exporting bearer tokens from user's application connections or machines, presenting them to application servers, and impersonating authenticated users. The idea of Token Binding is to prevent such attacks by cryptographically binding application security tokens to the underlying TLS [RFC5246] layer.

A Token Binding is established by a user agent generating a private-public key pair (possibly within a secure hardware module, such as a Trusted Platform Module) per target server, providing the public key to the server, and proving possession of the corresponding private key, on every TLS connection to the server. The proof of possession involves signing the exported keying material (EKM) [RFC5705] from the TLS connection with the private key. The corresponding public key is included in the Token Binding identifier structure (described in the Section 3.2 "TokenBinding.tokenbindingid"). Token Bindings are long-lived, i.e., they encompass multiple TLS connections and TLS sessions between a given client and server. To protect privacy, Token Binding IDs are never conveyed over insecure connections and can be reset by the user at any time, e.g., when clearing browser cookies.

When issuing a security token to a client that supports Token Binding, a server includes the client's Token Binding ID (or its cryptographic hash) in the token. Later on, when a client presents a security token containing a Token Binding ID, the server verifies that the ID in the token matches the ID of the Token Binding established with the client. In the case of a mismatch, the server rejects the token (details are application-specific).

In order to successfully export and replay a bound security token, an attacker needs to also be able to use the client's private key, which is hard to do if the key is specially protected, e.g., generated in a secure hardware module.

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. Token Binding Protocol Overview

In the course of a TLS handshake, a client and server can use the Token Binding Negotiation TLS Extension [I-D.ietf-tokbind-negotiation] to negotiate the Token Binding

protocol version and the parameters (signature algorithm, length) of the Token Binding key. This negotiation does not require additional round-trips.

As described in [I-D.ietf-tokbind-negotiation], version 1.0 of the Token Binding protocol is represented by `TB_ProtocolVersion.major = 1` and `TB_ProtocolVersion.minor = 0` in the Token Binding Negotiation TLS Extension.

The Token Binding protocol consists of one message sent by the client to the server, proving possession of one or more client-generated asymmetric private keys. This message is not sent if the Token Binding Negotiation has been unsuccessful. The Token Binding message is sent with the application protocol data over TLS.

A server receiving the Token Binding message verifies that the key parameters in the message match the Token Binding parameters negotiated (e.g., via [I-D.ietf-tokbind-negotiation]), and then validates the signatures contained in the Token Binding message. If either of these checks fails, the server rejects the binding, along with all associated bound tokens. Otherwise, the Token Binding is successfully established with the ID contained in the Token Binding message.

When a server supporting the Token Binding protocol receives a bound token, the server compares the Token Binding ID in the token with the Token Binding ID established with the client. If the bound token is received on a TLS connection without a Token Binding, or if the Token Binding IDs do not match, the token is rejected.

This document defines the format of the Token Binding protocol message, the process of establishing a Token Binding, the format of the Token Binding ID, and the process of validating a bound token. Token Binding Negotiation TLS Extension [I-D.ietf-tokbind-negotiation] describes the negotiation of the Token Binding protocol and key parameters. Token Binding over HTTP [I-D.ietf-tokbind-https] explains how the Token Binding message is encapsulated within HTTP/1.1 [RFC7230] or HTTP/2 [RFC7540] messages. [I-D.ietf-tokbind-https] also describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

3. Token Binding Protocol Message

The Token Binding message is sent by the client to prove possession of one or more private keys held by the client. This message **MUST** be sent if the client and server successfully negotiated the use of the Token Binding protocol (e.g., via [I-D.ietf-tokbind-negotiation]) or a

different mechanism), and MUST NOT be sent otherwise. This message MUST be sent in the client's first application protocol message. This message MAY also be sent in subsequent application protocol messages, proving possession of additional private keys held by the same client, which can be used to facilitate token binding between more than two communicating parties. For example, Token Binding over HTTP [I-D.ietf-tokbind-https] specifies an encapsulation of the Token Binding message in HTTP application protocol messages, as well as scenarios involving more than two communicating parties.

The Token Binding message format is defined using TLS Presentation Language (see Section 4 of [RFC5246]):

```
enum {
    rsa2048_pkcs1.5(0), rsa2048_pss(1), ecdsap256(2), (255)
} TokenBindingKeyParameters;

struct {
    opaque modulus<1..2^16-1>;
    opaque publicexponent<1..2^8-1>;
} RSAPublicKey;

struct {
    opaque point <1..2^8-1>;
} TB_ECPoint;

struct {
    TokenBindingKeyParameters key_parameters;
    uint16 key_length; /* Length (in bytes) of the following
                        TokenBindingID.TokenBindingPublicKey */
    select (key_parameters) {
        case rsa2048_pkcs1.5:
        case rsa2048_pss:
            RSAPublicKey rsapubkey;
        case ecdsap256:
            TB_ECPoint point;
    } TokenBindingPublicKey;
} TokenBindingID;

enum {
    (255) /* No initial TB_ExtensionType registrations */
} TB_ExtensionType;

struct {
    TB_ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} TB_Extension;
```

```
enum {  
    provided_token_binding(0), referred_token_binding(1), (255)  
} TokenBindingType;  
  
struct {  
    TokenBindingType tokenbinding_type;  
    TokenBindingID tokenbindingid;  
    opaque signature<64..2^16-1>; /* Signature over the concatenation  
                                   of tokenbinding_type,  
                                   key_parameters and exported  
                                   keying material (EKM) */  
    TB_Extension extensions<0..2^16-1>;  
} TokenBinding;  
  
struct {  
    TokenBinding tokenbindings<132..2^16-1>;  
} TokenBindingMessage;
```

The Token Binding message consists of a series of TokenBinding structures, each containing the type of the token binding, the TokenBindingID, a signature using the Token Binding key, optionally followed by TB_Extension structures.

3.1. TokenBinding.tokenbinding_type

This document defines two Token Binding types:

- o provided_token_binding - used to establish a Token Binding when connecting to a server.
- o referred_token_binding - used when requesting tokens that are intended to be presented to a different server.

Token Binding over HTTP [I-D.ietf-tokbind-https] describes a use case for referred_token_binding where Token Bindings are established between multiple communicating parties: User Agent, Identity Provider and Relying Party. User Agent sends referred_token_binding to the Identity Provider in order to prove possession of the Token Binding key it uses with the Relying Party. The Identity Provider can then bind the token it is supplying (for presentation to the Relying Party) to the Token Binding ID contained in the referred_token_binding.

An implementation MUST ignore any unknown Token Binding types.

3.2. TokenBinding.tokenbindingid

The ID of the Token Binding established as a result of Token Binding message processing contains the identifier of the negotiated key parameters, the length (in bytes) of the Token Binding public key, and the Token Binding public key itself. The Token Binding ID can be obtained from the TokenBinding structure by discarding the Token Binding type, signature and extensions.

When `rsa2048_pkcs1.5` or `rsa2048_pss` is used, `RSAPublicKey.modulus` and `RSAPublicKey.publicexponent` contain the modulus and exponent of a 2048-bit RSA public key represented in big-endian format, with leading zero bytes omitted.

When `ecdsap256` is used, `TB_ECPoint.point` contains the X coordinate followed by the Y coordinate of a Curve P-256 key. The X and Y coordinates are unsigned 32-byte integers encoded in big-endian format, preserving any leading zero bytes. Future specifications may define Token Binding keys using other elliptic curves with their corresponding signature and point formats.

Token Binding protocol implementations SHOULD make Token Binding IDs available to the application as opaque byte sequences, so that applications do not rely on a particular Token Binding ID structure. E.g., server applications will use Token Binding IDs when generating and verifying bound tokens.

3.3. TokenBinding.signature

When `rsa2048_pkcs1.5` is used, `TokenBinding.signature` contains the signature generated using the RSASSA-PKCS1-v1_5 signature scheme defined in [RFC8017] with SHA256 as the hash function.

When `rsa2048_pss` is used, `TokenBinding.signature` contains the signature generated using the RSASSA-PSS signature scheme defined in [RFC8017] with SHA256 as the hash function. MGF1 with SHA256 MUST be used as the mask generation function, and the salt length MUST equal 32 bytes.

When `ecdsap256` is used, `TokenBinding.signature` contains a pair of 32-byte integers, R followed by S, generated with ECDSA using Curve P-256 and SHA256 as defined in [ANSI.X9-62.2005] and [FIPS.186-4.2013]. R and S are encoded in big-endian format, preserving any leading zero bytes.

The signature is computed over the byte string representing the concatenation of:

- o TokenBindingType value contained in the TokenBinding.tokenbinding_type field;
- o TokenBindingKeyParameters value contained in the TokenBindingID.key_parameters field;
- o Exported keying material (EKM) value obtained from the current TLS connection.

Please note that TLS 1.2 and earlier versions support renegotiation, which produces a new TLS master secret for the same connection, with associated session keys and EKM value. TokenBinding.signature MUST be a signature of the EKM value derived from the TLS master secret that produced the session keys encrypting the TLS application_data record(s) containing this TokenBinding. Such use of the current EKM for the TLS connection makes replay of bound tokens within renegotiated TLS sessions detectable, but requires the application to synchronize Token Binding message generation and verification with the TLS handshake state.

Specifications defining the use of Token Binding with application protocols, such as Token Binding over HTTP [I-D.ietf-tokbind-https], MAY prohibit the use of TLS renegotiation in combination with Token Binding, obviating the need for such synchronization. Alternatively, such specifications need to define a way to determine which EKM value corresponds to a given TokenBindingMessage, and a mechanism preventing a TokenBindingMessage from being split across TLS renegotiation boundaries (i.e., due to TLS message fragmentation - see Section 6.2.1 of [RFC5246]). Note that application layer messages conveying a TokenBindingMessage may cross renegotiation boundaries in ways that make processing difficult.

The EKM is obtained using the Keying Material Exporters for TLS defined in [RFC5705], by supplying the following input values:

- o Label: The ASCII string "EXPORTER-Token-Binding" with no terminating NUL.
- o Context value: No application context supplied.
- o Length: 32 bytes.

3.4. TokenBinding.extensions

A Token Binding message may optionally contain a series of TB_Extension structures, each consisting of an extension_type and extension_data. The structure and meaning of extension_data depends on the specific extension_type.

Initially, no extension types are defined (see Section 6.3 "Token Binding Extensions Registry"). One of the possible uses of extensions envisioned at the time of this writing is attestation: cryptographic proof that allows the server to verify that the Token Binding key is hardware-bound. The definitions of such Token Binding protocol extensions are outside the scope of this specification.

4. Establishing a Token Binding

4.1. Client Processing Rules

The client **MUST** include at least one `TokenBinding` structure in the Token Binding message. The key parameters used in a `provided_token_binding` **MUST** match those negotiated with the server (e.g., via [I-D.ietf-tokbind-negotiation] or a different mechanism).

The client **MUST** generate and store Token Binding keys in a secure manner that prevents key export. In order to prevent cooperating servers from linking user identities, the scope of the Token Binding keys **MUST NOT** be broader than the scope of the tokens, as defined by the application protocol.

When the client needs to send a `referred_token_binding` to the Identity Provider, the client **SHALL** construct the referred `TokenBinding` structure in the following manner:

- o Set `TokenBinding.tokenbinding_type` to `referred_token_binding`.
- o Set `TokenBinding.tokenbindingid` to the Token Binding ID used with the Relying Party.
- o Generate `TokenBinding.signature`, using the EKM value of the TLS connection to the Identity Provider, the Token Binding key established with the Relying Party and the signature algorithm indicated by the associated key parameters. Note that these key parameters may differ from the key parameters negotiated with the Identity Provider.

Conveying referred Token Bindings in this fashion allows the Identity Provider to verify that the client controls the Token Binding key used with the Relying Party.

4.2. Server Processing Rules

The triple handshake vulnerability in TLS 1.2 and older TLS versions affects the security of the Token Binding protocol, as described in Section 7 "Security Considerations". Therefore, the server **MUST NOT** negotiate the use of the Token Binding protocol with these TLS

versions, unless the server also negotiates the Extended Master Secret [RFC7627] and Renegotiation Indication [RFC5746] TLS extensions.

If the use of the Token Binding protocol was not negotiated, but the client sends the Token Binding message, the server MUST reject any contained bindings.

If the Token Binding type is "provided_token_binding", the server MUST verify that the signature algorithm (including elliptic curve in the case of ECDSA) and key length in the Token Binding message match those negotiated with this client (e.g., via [I-D.ietf-tokbind-negotiation] or a different mechanism). In the case of a mismatch, the server MUST reject the binding. Token Bindings of type "referred_token_binding" may use different key parameters than those negotiated with this client.

If the Token Binding message does not contain at least one TokenBinding structure, or if a signature contained in any TokenBinding structure is invalid, the server MUST reject the binding.

Servers MUST ignore any unknown extensions. Initially, no extension types are defined (see Section 6.3 "Token Binding Extensions Registry").

If all checks defined above have passed successfully, the Token Binding between this client and server is established. The Token Binding ID(s) conveyed in the Token Binding Message can be provided to the server-side application. The application may then use the Token Binding IDs for bound security token creation and validation, see Section 5.

If a Token Binding is rejected, any associated bound tokens presented on the current TLS connection MUST also be rejected by the server. The effect of this is application-specific, e.g., failing requests, a requirement for the client to re-authenticate and present a different token, or connection termination.

5. Bound Security Token Creation and Validation

Security tokens can be bound to the TLS layer in a variety of ways: by embedding the Token Binding ID or its cryptographic hash in the token, or by maintaining a database mapping tokens to Token Binding IDs. The specific method of generating bound security tokens is application-defined and beyond the scope of this document. Note that applicable security considerations are outlined in Section 7.

Either or both clients and servers MAY create bound security tokens. For example, HTTPS servers employing Token Binding for securing their HTTP cookies will bind these cookies. In the case of a server-initiated challenge-response protocol employing Token Binding and TLS, the client can, for example, incorporate the Token Binding ID within the signed object it returns, thus binding the object.

Upon receipt of a security token, the server attempts to retrieve Token Binding ID information from the token and from the TLS connection with the client. Application-provided policy determines whether to honor non-bound (bearer) tokens. If the token is bound and a Token Binding has not been established for the client connection, the server MUST reject the token. If the Token Binding ID for the token does not match the Token Binding ID established for the client connection, the server MUST reject the token.

6. IANA Considerations

This section establishes three IANA registries on a new registry page entitled "Token Binding Protocol": "Token Binding Key Parameters", "Token Binding Types" and "Token Binding Extensions". It also registers a new TLS exporter label in the TLS Exporter Label Registry.

6.1. Token Binding Key Parameters Registry

This document establishes a registry for identifiers of Token Binding key parameters entitled "Token Binding Key Parameters" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies a set of Token Binding key parameters (0-255).
- o Description: The description of the Token Binding key parameters.
- o Specification: A reference to a specification that defines the Token Binding key parameters.

This registry operates under the "Specification Required" policy as defined in [RFC8126]. The designated expert will require the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified set of Token Binding key parameters.

An initial set of registrations for this registry follows:

Value: 0

Description: rsa2048_pkcs1.5

Specification: this document

Value: 1

Description: rsa2048_pss

Specification: this document

Value: 2

Description: ecdsap256

Specification: this document

6.2. Token Binding Types Registry

This document establishes a registry for Token Binding type identifiers entitled "Token Binding Types" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies the Token Binding type (0-255).
- o Description: The description of the Token Binding type.
- o Specification: A reference to a specification that defines the Token Binding type.

This registry operates under the "Specification Required" policy as defined in [RFC8126]. The designated expert will require the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding type.

An initial set of registrations for this registry follows:

Value: 0

Description: provided_token_binding

Specification: this document

Value: 1

Description: referred_token_binding

Specification: this document

6.3. Token Binding Extensions Registry

This document establishes a registry for Token Binding extensions entitled "Token Binding Extensions" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- o Value: The octet value that identifies the Token Binding extension (0-255).
- o Description: The description of the Token Binding extension.
- o Specification: A reference to a specification that defines the Token Binding extension.

This registry operates under the "Specification Required" policy as defined in [RFC8126]. The designated expert will require the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding extension. This document creates no initial registrations in the "Token Binding Extensions" registry.

6.4. Registration of Token Binding TLS Exporter Label

This document adds the following registration in the TLS Exporter Label Registry:

Value: EXPORTER-Token-Binding

DTLS-OK: Y

Reference: this document

7. Security Considerations

7.1. Security Token Replay

The goal of the Token Binding protocol is to prevent attackers from exporting and replaying security tokens, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can be replayed by malware present in User Agents, which may be undetectable by a server. However, in order to export bound tokens to other machines and successfully replay them, attackers also need to export 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.

The manner in which a token is bound to the TLS layer is application-defined and beyond the scope of this document. However, the resulting bound token needs to be integrity-protected, so that an attacker cannot remove the binding or substitute a Token Binding ID of their choice without detection.

The Token Binding protocol does not prevent cooperating clients from sharing a bound token. A client could intentionally export a bound token with the corresponding Token Binding private key, or perform signatures using this key on behalf of another client.

7.2. Downgrade Attacks

The Token Binding protocol MUST be negotiated using a mechanism that prevents downgrade. E.g., [I-D.ietf-tokbind-negotiation] uses a TLS extension for Token Binding negotiation. TLS detects handshake message modification by active attackers, therefore it is not possible for an attacker to remove or modify the "token_binding" extension without breaking the TLS handshake. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the negotiated parameters.

7.3. Privacy Considerations

The Token Binding protocol uses persistent, long-lived Token Binding IDs. To protect privacy, Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. Some applications offer a special privacy mode where they don't store or use tokens supplied by the server, e.g., "in private" browsing. When operating in this special privacy mode, applications SHOULD use newly generated Token Binding keys and delete them when exiting this mode, or else SHOULD NOT negotiate Token Binding at all.

In order to prevent cooperating servers from linking user identities, the scope of the Token Binding keys MUST NOT be broader than the scope of the tokens, as defined by the application protocol.

A server can use tokens and Token Binding IDs to track clients. Client applications that automatically limit the lifetime or scope of tokens to maintain user privacy SHOULD apply the same validity time and scope limits to Token Binding keys.

7.4. Token Binding Key Sharing Between Applications

Existing systems provide a variety of platform-specific mechanisms for certain applications to share tokens, e.g. to enable single sign-on scenarios. For these scenarios to keep working with bound tokens, the applications that are allowed to share tokens will need to also share Token Binding keys. Care must be taken to restrict the sharing of Token Binding keys to the same group(s) of applications that share the same tokens.

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

The Token Binding protocol relies on the TLS Exporters [RFC5705] to associate a TLS connection with a 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] and Renegotiation Indication [RFC5746] TLS extensions have also been negotiated.

8. 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, Benjamin Kaduk, Alexey Melnikov 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.

9. References

9.1. Normative References

- [ANSI.X9-62.2005]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 2005.
- [FIPS.186-4.2013]
National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS 186-4, 2013.
- [I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., Harper, N., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-15 (work in progress), May 2018.
- [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-13 (work in progress), May 2018.
- [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>>.
- [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>>.
- [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, <<https://www.rfc-editor.org/info/rfc5746>>.
- [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>>.

- [RFC7540] Belshé, 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>>.
- [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, <<https://www.rfc-editor.org/info/rfc7627>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [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>>.

9.2. Informative References

- [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>>.
- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/info/rfc8017>>.
- [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 (editor)
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz
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