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]

Table of Contents

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 an the client.

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

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

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

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

- Therefore, in the previous step, the Token Provider may want to include in the token the Token-Binding public key that the client uses when communicating with the Token Consumer, thus _binding_ the token to client's Token-Binding keypair. The client proves possession of the private key when communicating with the Token Consumer through the Token Binding Protocol [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.

- 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-
sensitive
```

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

    Client,                      Token Consumer,       Token Provider,
    aka:                         aka:                  aka:
    User Agent                   OpenID Client,        OpenID Provider,
                                 OIDC Relying Party,   OIDC Provider,
                                 SAML Relying Party    SAML Identity Provider
                                 [ server "1" ]        [ server "2" ]
    +--------+                        +----+                +-----+
    | Client |                        | TC |                | TP  |
    +--------+                        +----+                +-----+
        |                               |                      |
        |                               |                      |
        |                               |                      |
        | 0. Client interacts w/TC      |                      |
        | over HTTPS, establishes Ks1 & Kp1, TBID1             |
        | ETBMSG[[{EKM1}Ks1,TBID1,provided_token_binding]]     |
        |------------------------------>|                      |
        |                               |                      |
        |                               |                      |
        |                               |                      |
        | 1a. OIDC ID Token request, aka|                      |
```

```
      | "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 manterial 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 men-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