Internet Engineering Task Force   A. Popov, Ed.
Internet-Draft   M. Nystroem
Intended status: Standards Track   Microsoft Corp.
Expires: November 24, 2018   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

Table of Contents

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] 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>.

   [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