Token Binding Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 17, 2017

N. Harper
Google Inc.

September 13, 2016

Token Binding for 0-RTT TLS 1.3 Connections
draft-nharper-0-rtt-token-binding-00

Abstract

This document describes how Token Binding can be used in the 0-RTT data of a TLS 1.3 connection. This involves defining a 0-RTT exporter for TLS 1.3 and updating how Token Binding negotiation works. A TokenBindingMessage sent in 0-RTT data has different security properties than one sent after the TLS handshake has finished, which this document also describes.

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 March 17, 2017.

Copyright Notice

Table of Contents

1. Introduction

Token Binding ([I-D.ietf-tokbind-protocol]) cryptographically binds security tokens (e.g. HTTP cookies, OAuth tokens) to the TLS layer on which they are presented. It does so by signing an [RFC5705] exporter value from the TLS connection. TLS 1.3 introduces a new mode that allows a client to send application data on its first flight. If this 0-RTT data contains a security token, the client would want to prove possession of its private key. However, the TLS exporter cannot be run until the handshake has finished. This document describes changes to Token Binding to allow for a client to send a proof of possession in its 0-RTT application data, albeit with weaker security properties.

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. Proposed Design

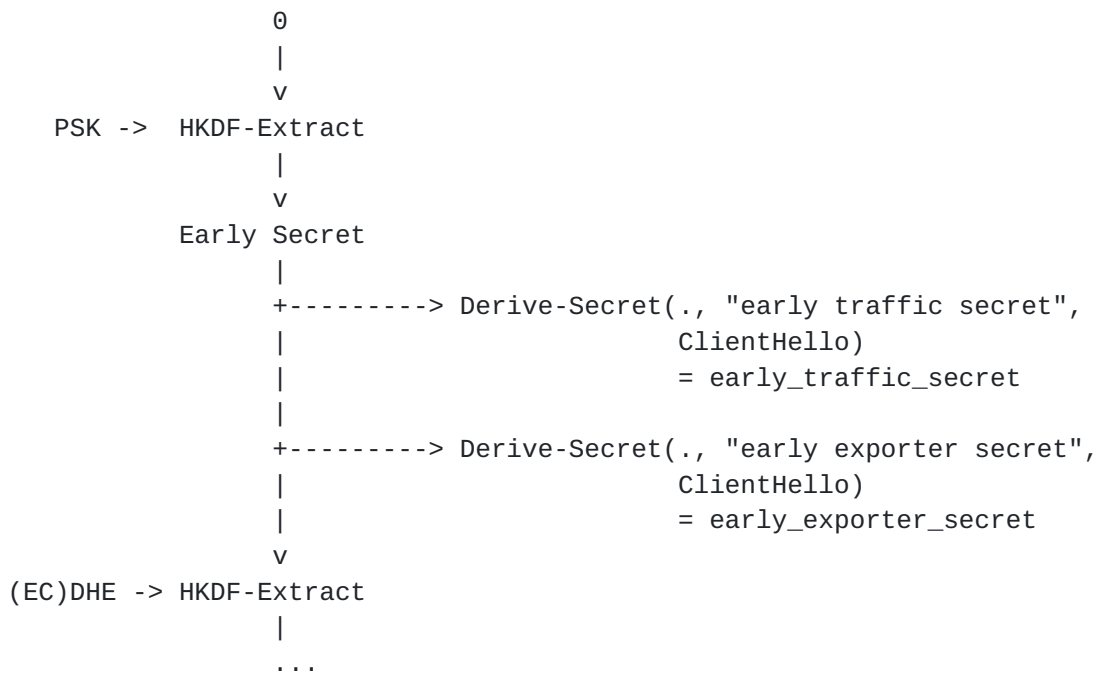
A TokenBinding struct as defined in [I-D.ietf-tokbind-protocol] contains a signature of the EKM value from the TLS layer. When a client is building 0-RTT data to send on a TLS 1.3 connection, there

is no EKM value available. This design changes the definition of the TokenBinding.signature field to use a different exporter for 0-RTT data, as well as defines that exporter. Since no negotiation for the connection can happen before the client sends this TokenBindingMessage in 0-RTT data, this document also describes how a client decides what TokenBindingMessage to send in 0-RTT data and how a server should interpret that message.

If a client does not send any 0-RTT data, or if the server rejects the client's 0-RTT data, then the client MUST use the existing 1-RTT exporter, as defined in [I-D.ietf-tokbind-protocol].

2.1. 0-RTT Exporter

In the key schedule for TLS 1.3, step is added between Early Secret and HKDF-Extract(ECDHE, Early Secret) to derive a value early_exporter_secret. With this modification, the key schedule (from [I-D.ietf-tls-tls13] section 7.1) looks like the following:



This definition does not affect the value of anything else derived in this key schedule.

The 0-RTT exporter is defined similarly to exporter in section 7.3.3, and has the same interface as the [RFC5705] exporter. It is defined as:

```
HKDF-Expand-Label(early_exporter_secret,
                  label, context_value, key_length)
```

Where HKDF-Expand-Label is the same function defined in [I-D.ietf-tls-tls13].

2.2. TokenBinding Signature Definition

In [I-D.ietf-tokbind-protocol], the signature field of the TokenBinding struct is defined to be the signature of a concatentation that includes the EKM value. This document changes that EKM value to be one of two possible values.

The first exporter value is the output of the 0-RTT exporter defined above, which can be used in any TokenBindingMessage. The second is the exporter defined in section 7.3.3 of [I-D.ietf-tls-tls13], which can only be used once the handshake is complete. In both cases, the exporter is called with the following input values:

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

These are the same values as defined in section 3 of [I-D.ietf-tokbind-protocol].

The rules for a client choosing which exporter to use are as follows. A client which is not sending any 0-RTT data on a connection MUST use the exporter defined in [I-D.ietf-tls-tls13] for all TokenBindingMessages on that connection so that it is compatible with [I-D.ietf-tokbind-protocol]. A client that sends a TokenBindingMessage in 0-RTT data must use the 0-RTT exporter defined in this document since the one in [I-D.ietf-tls-tls13] cannot be used at that time. A client that sends 0-RTT data which is not rejected by the server MUST use the 0-RTT exporter for the rest of the connection. If the server rejects the client's 0-RTT data, then the client MUST use the exporter defined in [I-D.ietf-tls-tls13] for the remainder of the connection, as if no 0-RTT data had ever been sent.

2.3. Negotiation TLS Extension

The behavior of the Token Binding negotiation TLS extension does not change for a 0-RTT connection: the client and server should process this extension the same way regardless of whether the client also sent the EarlyDataIndication extension.

For the sake of choosing a key parameter to use in 0-RTT data, the client MUST use the same key parameter that was used on the connection during which the ticket (now being used for resumption) was established. The server MUST NOT accept early data if the negotiated Token Binding key parameter does not match the parameter from the initial connection. This is the same behavior as ALPN and SNI extensions.

3. Implementation Challenges

The client has to be able to modify the message it sends in 0-RTT data if the 0-RTT data gets rejected and needs to be retransmitted in 1-RTT data. Even if the Token Binding integration with 0-RTT were modified so that Token Binding never caused a 0-RTT reject that required rewriting a request, the client still has to handle the server rejecting the 0-RTT data for other reasons.

HTTP2 allows for requests to different domains to share the same TLS connection if the SAN of the cert covers those domains. If one.example.com supports 0-RTT and Token Binding, but two.example.com only supports Token Binding as defined in [I-D.ietf-tokbind-protocol], those servers cannot share a cert and use HTTP2.

4. Alternatives Considered

4.1. Use Both 0-RTT and 1-RTT Exporters on Same Connection

The client could be required to use the 0-RTT EKM when the TokenBindingMessage is sent in 0-RTT data, and the 1-RTT EKM when it is sent in 1-RTT data. This creates synchronization issues on both the client and server to know when the application layer switched from writing in early data to writing after the handshake finished (and this switch could be in the middle of an HTTP request).

This constraint could be relaxed slightly. A ratcheting mechanism could be used where the client uses the 0-RTT EKM while it thinks that it's writing early data (even if it isn't writing early data), and once it knows the handshake is finished, it uses the 1-RTT EKM. Once the server sees a TokenBindingMessage using the 1-RTT EKM, the server would no longer accept the 0-RTT EKM. In practice, this is difficult to implement because multiple HTTP/2 streams can be multiplexed on the same connection, requiring the ratchet to be synchronized across the streams.

Relaxing this further where the server will always accept either the 0-RTT or 1-RTT EKM (but the client keeps the behavior as above) is another possibility. This is more complicated than always using the

0-RTT exporter, and provides no additional security benefits (since the server would have to accept a client only using the 0-RTT exporter).

4.2. Don't Remember Key Parameter From Previous Connection

The proposed design uses the same Token Binding key parameter from the previous connection, and the TLS extension must negotiate the same key parameter as the previous connection. This mirrors how ALPN is negotiated in TLS 1.3. Instead of remembering this parameter, the client could put the in first entry of their key parameters list the key type being used in 0-RTT, and allow the client and server to potentially negotiate a new type to use once the handshake is complete. This alternate gains a slight amount of key type agility in exchange for implementation difficulty. Other variations of this are also possible, for example requiring the server to reject early data if it doesn't choose the first parameter, or requiring the client to send only one key parameter.

4.3. Token Binding and 0-RTT Data Are Mutually Exclusive

If a TokenBindingMessage is never allowed in 0-RTT data, then no changes are needed to the exporter or negotiation. A server that wishes to support Token Binding must not create any NewSessionTicket messages with the allow_early_data flag set. A client must not send the token binding negotiation extension and the EarlyDataIndication extension in the same ClientHello.

5. Security Considerations

Token Binding messages that use the 0-RTT exporter have weaker security properties than with the [RFC5705] exporter. If either party of a connection using Token Binding does not wish to use 0-RTT token bindings, they can do so: a client can choose to never send 0-RTT data on a connection where it uses token binding, and a server can choose to reject any 0-RTT data sent on a connection that negotiated token binding.

0-RTT data in TLS 1.3 can be replayed by an attacker. Token Binding is not designed to prevent 0-RTT data from being replayed.

5.1. Exporter Weaknesses

The exporter specified in [I-D.ietf-tokbind-protocol] is chosen so that a client and server have the same exporter value only if they are on the same TLS connection. This prevents an attacker who can read the plaintext of a TokenBindingMessage sent on that connection from replaying that message on another connection (without also

having the token binding private key). The 0-RTT exporter only covers the ClientHello and the PSK of the connection, so it does not provide this guarantee.

An attacker with possession of the PSK secret and a transcript of the ClientHello and early data sent by a client under that PSK can extract the TokenBindingMessage, create a new connection to the server (using the same ClientHello and PSK), and send different application data with the same TokenBindingMessage. Note that the ClientHello contains public values for the (EC)DHE key agreement that is used as part of deriving the traffic keys for the TLS connection, so if the attacker does not also have the corresponding private values, they will not be able to read the server's response or send a valid Finished message in the handshake for this TLS connection. Nevertheless, by that point the server has already processed the attacker's message with the replayed TokenBindingMessage.

If the client secures the PSK with the same level of protection as the Token Binding key, then for an attacker to steal the PSK to attack the 0-RTT exporter would mean that the attacker could also steal the Token Binding key directly. Therefore, it is recommended that any client implementing Token Binding on 0-RTT connections also secure their PSK resumption secrets with the same strength as their Token Binding keys.

This sort of replayability of a TokenBindingMessage is different than the replayability caveat of 0-RTT application data in TLS 1.3. A network observer can replay 0-RTT data from TLS 1.3 without knowing any secrets of the client or server, but the application data that is replayed is untouched. This replay is done by a more powerful attacker who is able to view the plaintext and then spoof a connection with the same parameters so that the replayed TokenBindingMessage still validates when sent with different application data.

5.2. Early Data Ticket Age Window

When an attacker with control of the PSK secret replays a TokenBindingMessage, it has to use the same ClientHello that the client used. The ClientHello includes an "obfuscated_ticket_age" in its EarlyDataIndication extension, which the server can use to narrow the window in which that ClientHello will be accepted. Even if a PSK is valid for a week, the server will only accept that particular ClientHello for a smaller time window based on the ticket age. A server should make their acceptance window for this value as small as practical to limit an attacker's ability to replay a ClientHello and send new application data with the stolen TokenBindingMessage.

6. Acknowledgements

The author would like to thank David Benjamin, Steven Valdez, and Bill Cox for their feedback and suggestions.

7. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-15 (work in progress), August 2016.

[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-05 (work in progress), September 2016.

[I-D.ietf-tokbind-protocol]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "The Token Binding Protocol Version 1.0", draft-ietf-tokbind-protocol-10 (work in progress), September 2016.

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

[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>.

Author's Address

Nick Harper
Google Inc.

Email: nharper@google.com