Abstract

This document describes an interface for importing external PSK (Pre-Shared Key) into TLS 1.3.

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 April 4, 2020.

Copyright Notice

Copyright (c) 2019 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.
1. Introduction

TLS 1.3 [RFC8446] supports pre-shared key (PSK) authentication, wherein PSKs can be established via session tickets from prior connections or externally via some out-of-band mechanism. The protocol mandates that each PSK only be used with a single hash function. This was done to simplify protocol analysis. TLS 1.2 [RFC5246], in contrast, has no such requirement, as a PSK may be used with any hash algorithm and the TLS 1.2 PRF. This means that external PSKs could possibly be re-used in two different contexts with the same hash functions during key derivation. Moreover, it requires external PSKs to be provisioned for specific hash functions.

To mitigate these problems, external PSKs can be bound to a specific KDF and hash function when used in TLS 1.3, even if they are associated with a different hash function when provisioned. This document specifies an interface by which external PSKs may be imported for use in a TLS 1.3 connection to achieve this goal. In particular, it describes how KDF-bound PSKs can be differentiated by the target (D)TLS protocol version and KDF for which the PSK will be used. This produces a set of candidate PSKs, each of which are bound to a specific target protocol and KDF. This expands what would normally have been a single PSK identity into a set of PSK identities. However, importantly, it requires no change to the TLS 1.3 key schedule.
2. Conventions and Definitions

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.

3. Overview

Key importers mirror the concept of key exporters in TLS in that they diversify a key based on some contextual information before use in a connection. In contrast to key exporters, wherein differentiation is done via an explicit label and context string, the key importer defined herein uses an optional context string along with a target protocol and KDF identifier to differentiate an external PSK into one or more PSKs for use.

Imported keys do not require negotiation for use, as a client and server will not agree upon identities if not imported correctly. Thus, importers induce no protocol changes with the exception of expanding the set of PSK identities sent on the wire. Endpoints may incrementally deploy PSK importer support by offering non-imported keys for TLS versions prior to TLS 1.3. Non-imported and imported PSKs are distinct since their identities are different on the wire. See Section 6 for more details.

Clients which import external keys TLS MUST NOT use these keys for any other purpose. Moreover, each external PSK MUST be associated with at most one hash function.

3.1. Terminology

- **External PSK (EPSK):** A PSK established or provisioned out-of-band, i.e., not from a TLS connection, which is a tuple of (Base Key, External Identity, Hash).

- **Base Key:** The secret value of an EPSK.

- **External Identity:** The identity of an EPSK.

- **Target protocol:** The protocol for which a PSK is imported for use.

- **Target KDF:** The KDF for which a PSK is imported for use.

- **Imported PSK (IPSK):** A PSK derived from an EPSK, external identity, optional context string, and target protocol and KDF.
4. Key Import

A key importer takes as input an EPSK with external identity "external_identity" and base key "epsk", as defined in Section 3.1, along with an optional context, and transforms it into a set of PSKs and imported identities for use in a connection based on supported (target) protocols and KDFs. In particular, for each supported target protocol "target_protocol" and KDF "target_kdf", the importer constructs an ImportedIdentity structure as follows:

```
struct {
  opaque external_identity<1...2^16-1>;
  opaque context<0..2^16-1>;
  uint16 target_protocol;
  uint16 target_kdf;
} ImportedIdentity;
```

The list of "target_kdf" values is maintained by IANA as described in Section 9. External PSKs MUST NOT be imported for versions of (D)TLS 1.2 or prior versions. See Section 6 for discussion on how imported PSKs for TLS 1.3 and non-imported PSKs for earlier versions co-exist for incremental deployment.

ImportedIdentity.context MUST include the context used to derive the EPSK, if any exists. For example, ImportedIdentity.context may include information about peer roles or identities to mitigate Selfie-style reflection attacks. See Appendix B for more details. If the EPSK is a key derived from some other protocol or sequence of protocols, ImportedIdentity.context MUST include a channel binding for the deriving protocols [RFC5056].

ImportedIdentity.target_protocol MUST be the (D)TLS protocol version for which the PSK is being imported. For example, TLS 1.3 [RFC8446] and QUICv1 [QUIC] use 0x0304. Note that this means future versions of TLS will increase the number of PSKs derived from an external PSK.

An Imported PSK derived from an EPSK with base key ‘epsk’ bound to this identity is then computed as follows:

```
epskx = HKDF-Extract(0, epsk)
ipskx = HKDF-Expand-Label(epskx, "derived psk",
                         Hash(ImportedIdentity), L)
```

L is corresponds to the KDF output length of ImportedIdentity.target_kdf as defined in Section 9. For hash-based
KDFs, such as HKDF_SHA256(0x0001), this is the length of the hash function output, i.e., 32 octets. This is required for the IPSK to be of length suitable for supported ciphersuites.

The identity of `ipskx` as sent on the wire is ImportedIdentity.

The hash function used for HKDF [RFC5869] is that which is associated with the EPSK. It is not the hash function associated with ImportedIdentity.target_kdf. If no hash function is specified, SHA-256 MUST be used. Diversifying EPSK by ImportedIdentity.target_kdf ensures that an IPSK is only used as input keying material to at most one KDF, thus satisfying the requirements in [RFC8446].

Endpoints generate a compatible ipskx for each target ciphersuite they offer. For example, importing a key for TLS_AES_128_GCM_SHA256 and TLS_AES_256_GCM_SHA384 would yield two PSKs, one for HKDF-SHA256 and another for HKDF-SHA384. In contrast, if TLS_AES_128_GCM_SHA256 and TLS_CHACHA20_POLY1305_SHA256 are supported, only one derived key is necessary.

The resulting IPSK base key `ipskx` is then used as the binder key in TLS 1.3 with identity ImportedIdentity. With knowledge of the supported KDFs, one may import PSKs before the start of a connection.

EPSKs may be imported for early data use if they are bound to protocol settings and configurations that would otherwise be required for early data with normal (ticket-based PSK) resumption. Minimally, that means ALPN, QUIC transport settings, etc., must be provisioned alongside these EPSKs.

5. Deprecating Hash Functions

If a client or server wish to deprecate a hash function and no longer use it for TLS 1.3, they remove the corresponding KDF from the set of target KDFs used for importing keys. This does not affect the KDF operation used to derive Imported PSKs.

6. Incremental Deployment

Recall that TLS 1.2 permits computing the TLS PRF with any hash algorithm and PSK. Thus, an EPSK may be used with the same KDF (and underlying HMAC hash algorithm) as TLS 1.3 with importers. However, critically, the derived PSK will not be the same since the importer differentiates the PSK via the identity, target protocol, and target KDF. Thus, PSKs imported for TLS 1.3 are distinct from those used in TLS 1.2, and thereby avoid cross-protocol collisions. Note that this
does not preclude endpoints from using non-imported PSKs for TLS 1.2. Indeed, this is necessary for incremental deployment.

7. Security Considerations

DISCLAIMER: This is a WIP draft and has not yet seen significant security analysis.

8. Privacy Considerations

External PSK identities are typically static by design so that endpoints may use them to lookup keying material. However, for some systems and use cases, this identity may become a persistent tracking identifier.

9. IANA Considerations

This specification introduces a new registry for TLS KDF identifiers and defines the following target KDF values:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>0x0000</td>
</tr>
<tr>
<td>HKDF_SHA256</td>
<td>0x0001</td>
</tr>
<tr>
<td>HKDF_SHA384</td>
<td>0x0002</td>
</tr>
</tbody>
</table>

New target KDF values are allocated according to the following process:

- Values in the range 0x0000-0xfeff are assigned via Specification Required [RFC8126].
- Values in the range 0xff00-0xffff are reserved for Private Use [RFC8126].

10. References

10.1. Normative References


10.2. Informative References


Appendix A. Acknowledgements

The authors thank Eric Rescorla and Martin Thomson for discussions that led to the production of this document, as well as Christian Huitema for input regarding privacy considerations of external PSKs. John Mattsson provided input regarding PSK importer deployment considerations.

Appendix B. Addressing Selfie

The Selfie attack [Selfie] relies on a misuse of the PSK interface. The PSK interface makes the implicit assumption that each PSK is known only to one client and one server. If multiple clients or multiple servers with distinct roles share a PSK, TLS only authenticates the entire group. A node successfully authenticates its peer as being in the group whether the peer is another node or itself.

Applications which require authenticating finer-grained roles while still configuring a single shared PSK across all nodes can resolve this mismatch either by exchanging roles over the TLS connection after the handshake or by incorporating the roles of both the client and server into the IPSK context string. For instance, if an application identifies each node by MAC address, it could use the following context string.

```c
struct {
    opaque client_mac<0..2^16-1>;
    opaque server_mac<0..2^16-1>;
} Context;
```

If an attacker then redirects a ClientHello intended for one node to a different node, the receiver will compute a different context string and the handshake will not complete.

Note that, in this scenario, there is still a single shared PSK across all nodes, so each node must be trusted not to impersonate another node’s role.

Authors’ Addresses

David Benjamin
Google, LLC.

Email: davidben@google.com
Christopher A. Wood
Apple, Inc.

Email: cawood@apple.com