tls D. Benjamin
Internet-Draft Google, LLC.
Intended status: Standards Track C.A. Wood
Expires: 10 October 2020 8 April 2020

Importing External PSKs for TLS
draft-ietf-tls-external-psk-importer-04

Abstract

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

Table of Contents

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, this document specifies a PSK Importer interface by which external PSKs may be imported and subsequently bound to a specific KDF and hash function for use in TLS 1.3. In particular, it describes a mechanism for differentiating external PSKs by the target KDF, (D)TLS protocol version, and an optional context string. This process yields a set of candidate PSKs, each of which are bound to a target KDF and protocol. This expands what would normally have been a single PSK and identity into a set of PSKs and identities.

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

The PSK Importer interface mirrors that of the TLS Exporters interface in that it diversifies a key based on some contextual information. In contrast to the Exporters interface, wherein differentiation is done via an explicit label and context string, the PSK Importer interface defined herein takes an external PSK and identity and "imports" it into TLS, creating a set of "derived" PSKs and identities. Each of these derived PSKs are bound a target protocol, KDF identifier, and optional context string. Additionally, the resulting PSK binder keys are modified with a new derivation label to prevent confusion with non-imported PSKs.

Imported keys do not require negotiation for use since a client and server will not agree upon identities if imported incorrectly. 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 MUST NOT use either the external keys or the derived keys for any other purpose. Moreover, each external PSK MUST be associated with at most one hash function, as per the rules in Section 4.2.11 from [RFC8446]. See Section 7 for more discussion.

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: A sequence of bytes used to identify 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.

* Imported Identity: A sequence of bytes used to identify an IPSK.

4. PSK Import

This section describes the PSK Importer interface and its underlying diversification mechanism and binder key computation modification.

4.1. External PSK Diversification

The PSK Importer interface 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 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 (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.

Given an ImportedIdentity and corresponding EPSK with base key

"epsk", an Imported PSK IPSK with base key "ipskx" is computed as follows:

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

L 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, i.e., the serialized content of ImportedIdentity is used as the content of PskIdentity.identity in the PSK extension. The corresponding TLS 1.3 binder key is "ipskx".

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]. See Section 7 for more details.

Endpoints SHOULD 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.

EPSKs may be imported before the start of a connection if the target KDFs, protocols, and context string(s) are known a priori. EPSKs may also 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 parameters (if used for QUIC), etc., must be provisioned alongside these EPSKs.

4.2. Binder Key Derivation

To prevent confusion between imported and non-imported PSKs, imported PSKs change the PSK binder key derivation label. In particular, the standard TLS 1.3 PSK binder key computation is defined as follows:


```
                0
                |
                v
   PSK ->  HKDF-Extract = Early Secret
                |
                +-----> Derive-Secret(., "ext binder" | "res binder", "")
                |                     = binder_key
                V
```

Imported PSKs replace the string "ext binder" with "imp binder" when deriving "binder_key". This means the binder key is computed as follows:

```
                0
                |
```

```
              v
PSK ->  HKDF-Extract = Early Secret
              |
              +-----> Derive-Secret(., "ext binder"
              |                      | "res binder"
              |                      | "imp binder", "")
              |                     = binder_key
              V
```

This new label ensures a client and server will negotiate use of an external PSK if and only if (a) both endpoints import the PSK or (b) neither endpoint imports the PSK. As "binder_key" is a leaf key, changing its computation does not affect any other key.

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 and target KDF and protocol. 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

The PSK Importer security goals can be roughly stated as follows: avoid PSK re-use across KDFs while properly authenticating endpoints. When modeled as computational extractors, KDFs assume that input keying material (IKM) is sampled from some "source" probability distribution and that any two IKM values are chosen independently of each other [Kraw10]. This source-independence requirement implies that the same IKM value cannot be used for two different KDFs.

PSK-based authentication is functionally equivalent to session resumption in that a connection uses existing key material to authenticate both endpoints. Following the work of [BAA15], this is a form of compound authentication. Loosely speaking, compound authentication is the property that an execution of multiple authentication protocols, wherein at least one is uncompromised, jointly authenticates all protocols. Authenticating with an externally provisioned PSK, therefore, should ideally authenticate both the TLS connection and the external provisioning process. Typically, the external provision process produces a PSK and corresponding context from which the PSK was derived and in which it should be used. If available, this is used as the ImportedIdentity.context value. We refer to an external PSK without such context as "context-free".

Thus, in considering the source-independence and compound authentication requirements, the PSK Import interface described in this document aims to achieve the following goals:

1. Externally provisioned PSKs imported into a TLS connection achieve compound authentication of the provisioning process and connection.

2. Context-free PSKs only achieve authentication within the context of a single connection.

3. Imported PSKs are not used as IKM for two different KDFs.

4. Imported PSKs do not collide with existing PSKs used for TLS 1.2 and below.

5. Imported PSKs do not collide with future protocol versions and KDFs.

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:

KDF Description	Value
Reserved	0x0000
HKDF_SHA256	0x0001
HKDF_SHA384	0x0002

Table 1: Target KDF Registry

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

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-27, 21 February 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt>.

[RFC1035] Mockapetris, P.V., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/info/rfc5056>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

10.2. Informative References

[BAA15] Bhargavan, K., Delignat-Lavaud, A., and A. Pironti, "Verified Contributive Channel Bindings for Compound Authentication", DOI 10.14722/ndss.2015.23277, Proceedings 2015 Network and Distributed System Security Symposium, 2015, <https://doi.org/10.14722/ndss.2015.23277>.

[CCB] Bhargavan, K., Delignat-Lavaud, A., and A. Pironti, "Verified Contributive Channel Bindings for Compound Authentication", DOI 10.14722/ndss.2015.23277, Proceedings

2015 Network and Distributed System Security Symposium, 2015, <https://doi.org/10.14722/ndss.2015.23277>.

[Kraw10] Krawczyk, H., "Cryptographic Extraction and Key Derivation: The HKDF Scheme", Proceedings of CRYPTO 2010 , 2010, <https://eprint.iacr.org/2010/264>.

[Selfie] Drucker, N. and S. Gueron, "Selfie: reflections on TLS 1.3 with PSK", 2019, <https://eprint.iacr.org/2019/347.pdf>.

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. Hugo Krawczyk provided guidance for the security 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.

```
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

Email: caw@heapingbits.net