Workgroup: tls
Internet-Draft:
draft-ietf-tls-external-psk-importer-08
Published: 22 April 2022
Intended Status: Standards Track
Expires: 24 October 2022
Authors: D. Benjamin   C.A. Wood
Google, LLC.   Cloudflare

Importing External PSKs for TLS

Abstract

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

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/tlswg/draft-ietf-tls-external-psk-importer.

Status of This Memo

Copyright Notice

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 pseudorandom function (PRF). While there is no known way in which the same external PSK might produce related output in TLS 1.3 and prior versions, only limited analysis has been done. Applications SHOULD provision separate PSKs for (D)TLS 1.3 and prior versions. In cases where this is not possible, e.g., there are already deployed external PSKs or provisioning is otherwise limited, re-using external PSKs across different versions of TLS may produce related outputs, which may in turn lead to security problems; see [RFC8446], Section E.7.

To mitigate against such problems, this document specifies a PSK Importer interface by which external PSKs may be imported and subsequently bound to a specific key derivation function (KDF) and hash function for use in TLS 1.3 [RFC8446] and DTLS 1.3 [DTLS13]. In particular, it describes a mechanism for importing PSKs derived from external PSKs by including the target KDF, (D)TLS protocol version, and an optional context string to ensure uniqueness. This process yields a set of candidate PSKs, each of which are bound to a target KDF and protocol, that are separate from those used in (D)TLS 1.2 and prior versions. 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. Terminology

The following terms are used throughout this document:

* 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 TLS PSK derived from an EPSK, optional context string, target protocol, and target KDF.

* Non-imported PSK: An EPSK which used directly as a TLS PSK without being imported.

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

This document uses presentation language from [RFC8446], Section 3.

4. Overview

The PSK Importer interface mirrors that of the TLS Exporters interface (see [RFC8446]) in that it diversifies a key based on some contextual information. In contrast to the Exporters interface, wherein output uniqueness is achieved 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 that are each unique. Each of these derived PSKs are bound to 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. Through this interface, importing external PSKs with different identities yields distinct PSK binder keys.

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 PSKs for TLS versions prior to TLS 1.3. Non-imported and imported PSKs are distinct since their identities are different. See Section 7 for more details.

Endpoints which import external keys MUST NOT use the keys that are input to the import process for any purpose other than the importer, and MUST NOT use the derived keys for any purpose other than TLS PSKs. Moreover, each external PSK fed to the importer process MUST be associated with at most one hash function. This is analogous to the rules in Section 4.2.11 of [RFC8446]. See Section 8 for more discussion.

5. PSK Import

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

5.1. 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, 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 ImportedIdentity.target_kdf values is maintained by IANA as described in Section 10. External PSKs MUST NOT be imported for (D)TLS 1.2 or prior versions. See Section 7 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 determine the EPSK, if any exists. For example, ImportedIdentity.context may include information about peer roles or identities to mitigate Selfie-style reflection attacks [Selfie]. See Appendix B for more details. Since the EPSK is a key derived from an external protocol or sequence of protocols, ImportedIdentity.context MUST include a channel binding for the deriving protocols [RFC5056]. The details of this binding are protocol specific and out of scope for this document.

ImportedIdentity.target_protocol MUST be the (D)TLS protocol version for which the PSK is being imported. For example, TLS 1.3 [RFC8446] uses 0x0304, which will therefore also be used by QUICv1 [QUIC]. Note that this means the number of PSKs derived from an EPSK is a function of the number of target protocols.

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 10. For hash-based KDFs, such as HKDF_SHA256(0x0001), this is the length of the hash function output, e.g., 32 octets for SHA256. This is required for the IPSK to be of length suitable for supported ciphersuites. Internally, HKDF-Expand-Label uses a label corresponding to ImportedIdentity.target_protocol, e.g., "tls13" for TLS 1.3, as per [RFC8446], Section 7.1, or "dtls13" for DTLS 1.3, as per [I-D.ietf-tls-dtls13], Section 5.10.

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 PSK input for the TLS 1.3 key schedule is 'ipskx'.

As the maximum size of the PSK extension is 2^16 - 1 octets, an Imported Identity that exceeds this size is likely to cause a decoding error. Therefore, the PSK Importer interface SHOULD reject any ImportedIdentity that exceeds this size.

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 the EPSK does not have such an associated hash function, SHA-256 [SHA2] SHOULD 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 8 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. Each ciphersuite uniquely identifies the target KDF. Future specifications that change the way the KDF is negotiated will need to update this specification to make clear how target KDFs are determined for the import process.

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 the protocol settings and configuration that are required for sending early data. Minimally, this means that the Application-Layer Protocol Negotiation value [RFC7301], QUIC transport parameters (if used for QUIC), and any other relevant parameters that are negotiated for early data MUST be provisioned alongside these EPSKs.

5.2. 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 use the string "imp binder" rather than "ext binder" or "res 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.

6. Deprecating Hash Functions

If a client or server wishes 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.

7. Incremental Deployment

In deployments that already have PSKs provisioned and in use with TLS 1.2, attempting to incrementally deploy the importer mechanism would then result in concurrent use of the already provisioned PSK both directly as a TLS 1.2 PSK and as an EPSK, which in turn could mean that the same KDF and key would be used in two different protocol contexts. This is not a recommended configuration; see Section 8 for more details. However, the benefits of using TLS 1.3 and of using PSK importers may prove sufficiently compelling that existing deployments choose to enable this noncompliant configuration for a brief transition period while new software (using TLS 1.3 and importers) is deployed. Operators are advised to make any such transition period as short as possible.

8. 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 future protocol versions and KDFs.

There are no known related outputs or security issues caused from the process for computing Imported PSKs from an external PSK and the processing of existing external PSKs used in (D)TLS 1.2 and below, as noted in Section 7. However, only limited analysis has been done, which is an additional reason why applications SHOULD provision separate PSKs for (D)TLS 1.3 and prior versions, even when the importer interface is used in (D)TLS 1.3.

The PSK Importer does not prevent applications from constructing non-importer PSK identities that collide with imported PSK identities.

9. Privacy Considerations

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

Note also that ImportedIdentity.context is visible in cleartext on the wire as part of the PSK identity. Unless otherwise protected by a mechanism such as TLS Encrypted ClientHello [ECH], applications SHOULD NOT put sensitive information in this field.

10. IANA Considerations

This specification introduces a new registry for TLS KDF identifiers, titled "TLS KDF Identifiers", under the existing "Transport Layer Security (TLS) Parameters" heading.

The entries in the registry are:

KDF Description	Value	Reference
Reserved	0x0000	N/A
HKDF_SHA256	0x0001	[RFC5869]
HKDF_SHA384	0x0002	[RFC5869]

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].

The procedures for requesting values in the Specification Required space are specified in Section 17 of [RFC8447].

11. References

11.1. Normative References

[DTLS13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://www.ietf.org/archive/id/draft-ietf-tls-dtls13-43.txt>.

[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>.

[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>.

[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>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

11.2. 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>.

[ECH] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-14, 13 February 2022, <https://www.ietf.org/archive/id/draft-ietf-tls-esni-14.txt>.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://www.ietf.org/archive/id/draft-ietf-tls-dtls13-43.txt>.

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

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI

10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[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>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

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

[SHA2] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-3 , October 2008.

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. Martin Thomson, Jonathan Hoyland, Scott Hollenbeck, Benjamin Kaduk, and others all provided reviews, feedback, and suggestions for improving the document.

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. Note that this case can also occur when there are two nodes sharing a PSK without predetermined roles.

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..2^8-1>;
  opaque server_mac<0..2^8-1>;
} Context;
```

If an attacker then redirects a ClientHello intended for one node to a different node, including the node that generated the ClientHello, 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
Cloudflare

Email: caw@heapingbits.net