jhoyla — J. Hoyland
Internet-Draft — Cloudflare Ltd.
Intended status: Standards Track — C.A. Wood
Expires: 10 September 2020 — Apple, Inc.
9 March 2020

TLS 1.3 Extended Key Schedule
draft-jhoyla-tls-extended-key-schedule-01

Abstract

TLS 1.3 is sometimes used in situations where it is necessary to inject extra key material into the handshake. This draft aims to describe methods for doing so securely. This key material must be injected in such a way that both parties agree on what is being injected and why, and further, in what order.

Note to Readers

Discussion of this document takes place on the TLS Working Group mailing list (tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/jhoyla/draft-jhoyla-tls-key-injection (https://github.com/jhoyla/draft-jhoyla-tls-key-injection).

Copyright Notice

Table of Contents

1. Introduction

Introducing additional key material into the TLS handshake is a non-trivial process because both parties need to agree on the injection content and context. If the two parties do not agree then an attacker may exploit the mismatch in so-called channel synchronization attacks.

Injecting key material into the TLS handshake allows other protocols to be bound to the handshake. For example, it may provide additional protections to the ClientHello message, which in the standard TLS handshake only receives protections after the server's Finished message has been received. It may also permit the use of combined shared secrets, possibly from multiple key exchange algorithms, to be included in the key schedule. This pattern is common for Post Quantum key exchange algorithms, as discussed in [I-D.stebila-tls-hybrid-design].

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. Key Schedule Extension

This section describes two places in which additional secrets can be injected into the TLS 1.3 key schedule.

3.1. Handshake Secret Injection

To inject key material into the Handshake Secret it is recommended to use an extra derive secret.

```
             |
             v
       Derive-Secret(., "derived early", "")
             |
             v
    Input -> HKDF-Extract
             |
             v
       Derive-Secret(., "derived", "")
             |
             v
  (EC)DHE -> HKDF-Extract = Handshake Secret
             |
             v
```

As shown in the figure above, the key schedule has an extra derive secret and HKDF-Extract step. This extra step isolates the Input material from the rest of the handshake secret, such that even maliciously chosen values cannot weaken the security of the key schedule overall.

The additional Derive-Secret with the "derived early" label enforces the separation of the key schedule from vanilla TLS handshakes, because HKDFs can be assumed to ensure that keys derived with different labels are independent.

3.2. Master Secret Injection

To inject key material into the Master Secret it is recommended to use an extra derive secret.

```
                |
                v
          Derive-Secret(., "derived early", "")
                |
                v
       Input -> HKDF-Extract
                |
                v
          Derive-Secret(., "derived", "")
                |
                v
       0 -> HKDF-Extract = Master Secret
                |
                v
```

This structrue mirrors the Handshake Injection point, the key schedule has an extra Extract, Derive-Secret pattern. This, again, should isolate the Input material from the rest of the Master Secret.

4. Key Schedule Extension Structure

In some cases, protocols may require more than one secret to be injected at a particular stage in the key schedule. Thus, we require a generic and extensible way of doing so. To accomplish this, we use a structure - KeyScheduleInput - that encodes well-ordered sequences of secret material to inject into the key schedule. KeyScheduleInput is defined as follows:

```
struct {
    KeyScheduleSecretType type;
    opaque secret_data<0..2^16-1>;
} KeyScheduleSecret;

enum {
    (65535)
} KeyScheduleSecretType;

struct {
    KeyScheduleSecret secrets<0..2^16-1>;
} KeyScheduleInput;
```

Each secret included in a KeyScheduleInput structure has a type and corresponding secret data. Each secret MUST have a unique KeyScheduleSecretType. When encoding KeyScheduleInput as the key schedule Input value, the KeyScheduleSecret values MUST be in ascending sorted order. This ensures that endpoints always encode the same KeyScheduleInput value when using the same secret keying material.

5. Security Considerations

[[OPEN ISSUE: This draft has not seen any security analysis.]]

6. IANA Considerations

[[TODO: define secret registry structure]]

7. References

7.1. Normative References

[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>.

[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>.

7.2. Informative References

[I-D.stebila-tls-hybrid-design]
Steblia, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-stebila-tls-hybrid-design-03, 12 February 2020, <http://www.ietf.org/internet-drafts/draft-stebila-tls-hybrid-design-03.txt>.

Acknowledgments

We thank Karthik Bhargavan for his comments.

Authors' Addresses

Jonathan Hoyland
Cloudflare Ltd.

Email: jonathan.hoyland@gmail.com

Christopher A. Wood
Apple, Inc.

Email: cawood@apple.com