TLS 1.3 Extended Key Schedule

draft-jhoyla-tls-extended-key-schedule-00

Jonathan Hoyland, Christopher A. Wood

IETF 106 - TLS WG - Singapore

Overview

TLS 1.3 has very carefully defined properties

Is there a need for new / different / stronger properties?

- Privacy properties for the ClientHello
- Hybrid Key Exchange

Can we achieve new / different / stronger properties without breaking anything?

- Non-trivial to get new properties without touching the key schedule
- Important that we don’t create new attack surface

Key Schedule Extension Use Cases

Hybrid key exchange

ESNI

External PSK Importers

Semi-static DH

We need an extensible framework

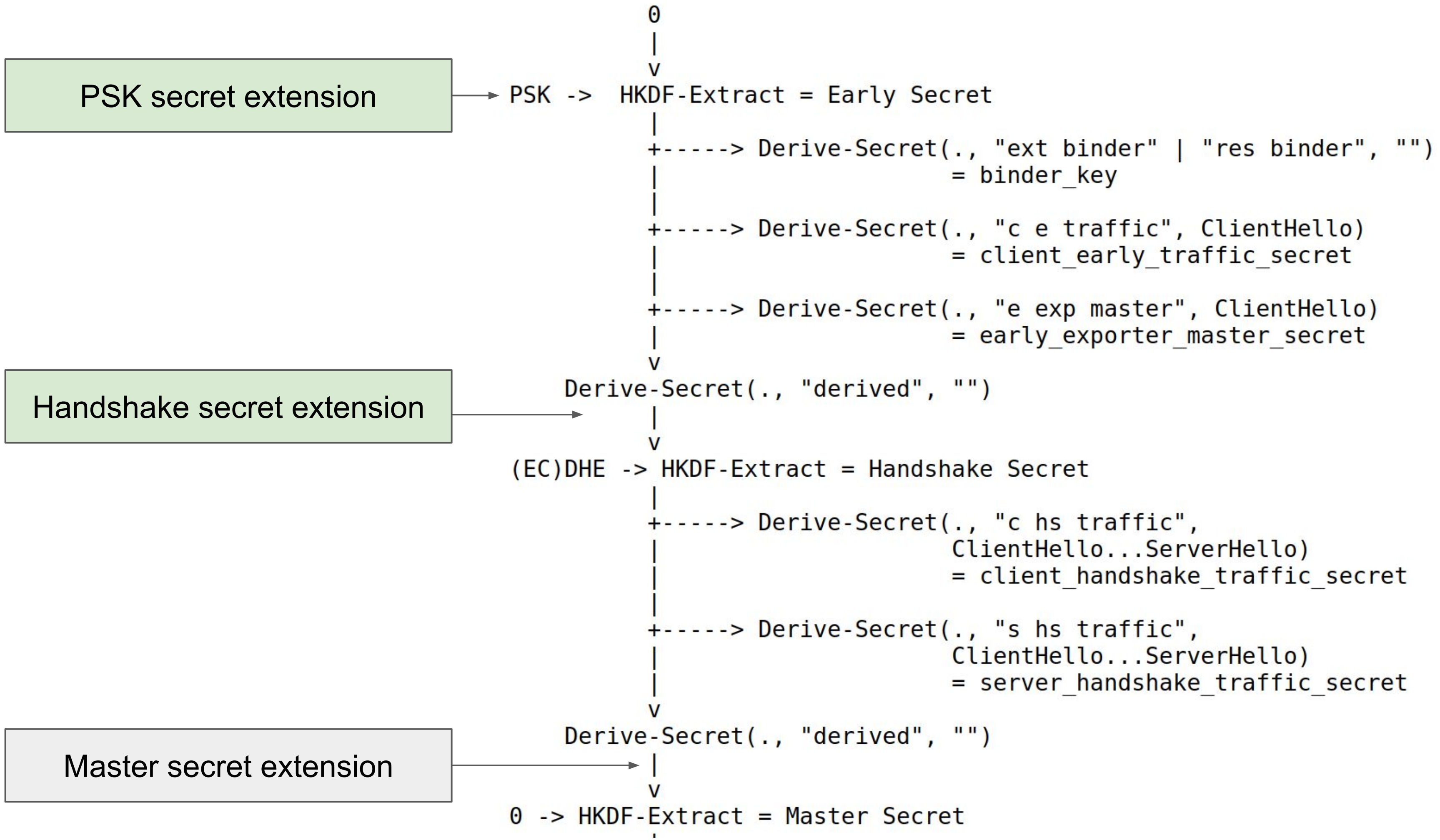
All the drafts mentioned want to modify the key schedule...

… in mutually incompatible ways

Generally with limited security analysis, especially of the properties when used with other key schedule modifications

This draft attempts to do two things

1. Capture the work already done on key schedule changes
2. Provide a framework for using these changes in a consistent, secure, and composable way



```
                 0
                 |
                 v
   PSK ->  HKDF-Extract = Early Secret
                 |
                 +-----> Derive-Secret(., "ext binder" | "res binder", "")
                 |                     = binder_key
                 |
                 +-----> Derive-Secret(., "c e traffic", ClientHello)
                 |                     = client_early_traffic_secret
                 |
                 +-----> Derive-Secret(., "e exp master", ClientHello)
                 |                     = early_exporter_master_secret
                 v
           Derive-Secret(., "derived", "")
                 |
                 v
(EC)DHE -> HKDF-Extract = Handshake Secret
                 |
                 +-----> Derive-Secret(., "c hs traffic",
                 |                     ClientHello...ServerHello)
                 |                     = client_handshake_traffic_secret
                 |
                 +-----> Derive-Secret(., "s hs traffic",
                 |                     ClientHello...ServerHello)
                 |                     = server_handshake_traffic_secret
                 v
           Derive-Secret(., "derived", "")
                 |
                 v
      0 -> HKDF-Extract = Master Secret
```

TLS 1.3’s Expand Step

```
Derive-Secret(Secret, Label, Messages) =
            HKDF-Expand(Secret,
                “tls13 ” + Label + Hash(Messages),
                Hash.length)
```

TLS 1.3 uses a Hash-based Key Derivation Function (HKDF)

A KDF has two functions

- Extract - a randomness extractor
- Expand - a pseudorandom generator

These functions do different things:

- Extract takes imperfect randomness and “extracts” uniform randomness
- Expand takes a pseudorandom key and “expands” it into a number of cryptographically unrelated keys

Proposed Changes

1. PSK Extension

Current:

```
struct {
    opaque identity<1..2^16-1>;
    uint32 obfuscated_ticket_age;
} PskIdentity;
```

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

Proposed:

```
struct {
  opaque external_identity<1...2^16-1>;
  opaque context<0...2^16>;
} PSKIDWithAdditionalData;
```

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

What does this give us?

We can bind extra information to the handshake

- Analogous to TLS Exporter interface
- We can layer TLS on top of other protocols

Doesn’t affect the current security proofs*

- Derive-secret is an HKDF-Expand
- Assume that the key space for these new labels is computationally disjoint (i.e. negligible probability of collisions)

*Analysis underway

2. Handshake Secret Extension

Add an extra “Expand - Extract”

Attempt to isolate the injected material from the input

Inputs are injected in a fixed order

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

Extension Points

Make an IANA registry that fixes the order of injection

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

enum {
    (65535)
} KeyScheduleSecretType;

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

What does this give us?

We can add new secrets to the handshake

- Encrypted ClientHello extensions
- Add quantum-safe secrets without giving up classical crypto

Limits an attacker’s ability to bias the output key

- If there is no PSK and a weak DH key then a malicious input may be able to bias the Handshake Secret

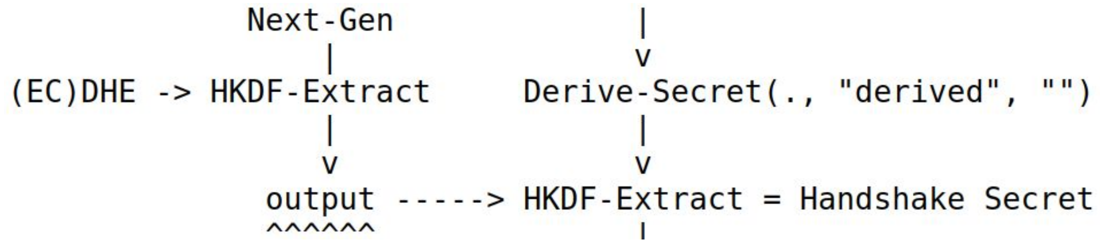
Alternative design - Stebila Comb-KDF-1

(+) Simpler to implement

(+) Can use the dual-PRF property of HKDF-Extract

(-) Key schedule no longer linear

(-) Less flexible



```
                 Next-Gen                   |
                    |                       v
(EC)DHE -> HKDF-Extract     Derive-Secret(., "derived", "")
                    |                       |
                    v                       v
                 output -----> HKDF-Extract = Handshake Secret
                 ^^^^^^                     |
```

Further Questions

Do we want to extend the Master Secret?

Should we just concatenate the injected secrets?

How worried should we be about the number of new hash invocations?

How should we handle extension negotiation?

- Should the client produce a single PSK?
- A group of PSKs?
- Rely on HelloRetry if the server wants to pick and chose?

Interest in adoption?

Backup

Hash-based Key Derivation Function

- An HKDF takes four arguments:
- HKDF(XTS, SKM, CTXinfo, L)
    - XTS - an eXTractor Salt
    - SKM - the Source Key Material
    - CTXinfo - a ConTeXt value that is bound to the output key
    - L - the desired length of the output

HKDF-Extract(XTS, SKM) -> PRK

HKDF-Expand(PRK, CTXinfo, L) -> OKM

Summary

- Extract “cleans” or “smooths” the random input
- Expand takes “clean” randomness and produces separated keys