Additional Symmetric Keys

Mathias hall-Andersen (mathias@hall-andersen.dk)

Revision 1, 2018-07-11, unofficial/unstable

Contents

1. Introduction

This extension is intended for applications wishing to use key material derived by Noise for auxiliary purposes or other extensions of Noise. Examples include:

- Resumption PSKs - Deriving a "resumption PSK" from an initial Noise session. The resumption PSK can be used to perform a PSK Noise handshake later. ASK can also be used to derive associated data, such as labels to identify PSKs.
- Deriving keys used to generate random padding or other length-hiding / traffic-hiding countermeasures, include the derivation of obfuscation keys used to discourage middle boxes from parsing fields inside messages.

- Generate application-layer keys in case the Noise handshake is being used as the handshake / key-establishment component within a larger secure transport layer protocol.

2. Overview

The Additional Symmetric Keys API is centered around labels and chains: **labels** are arbitrary byte strings, each label allows the construction of a **chain** based on the value of the label and the current `SymmetricState` value. An arbitrary amount of key material can be extracted from every chain. The API consists of 2 methods:

- **InitASK(label)**: Initializes a chain for the given label.
- **GetASK(label)**: Assuming that a chain has been initialized for the label this method returns the next key from the appropriate chain and advances the chain to enable forward secrecy for ASK outputs.

3. Security considerations

The Additional Symmetric Keys extension is design to meet the following security goals:

- The ASKs should be independent from the Noise chaining key. Recovering knowledge about the Noise chaining key from any number of ASK outputs must be infeasible.
- The ASKs should be mutual independent; deriving any of the ASK outputs from any other should be infeasible.
- ASKs output should be capable of serving as collision resistant hashes of the session transcript at the security level expected by the employed hash function (256/512-bit). Since the GetASK method only outputs 256-bit, we require that the concatenation of two GetASK outputs (from the same or distinct chains) offer the same level of collision resistance as the underlying hash function.

4. Design

In addition to the security goals stated above, the extension seeks to meet the following design goals:

- The mechanism should not be restricted to using the output key-material for particular purposes (e.g. the use cases listed above).

- When not used, the ASK mechanism should incur any significant additional computational cost. This allows libraries to implement the extension without introducing significant overhead on applications not using this extension.

5. Implementation

Implementation of ASK uses HKDF [1] and augments the operation of **MixKey** and **MixKeyAndHash** by clearing the chains whenever the `SymmetricState` object are updated: after any call to either **MixKey** or **MixKeyAndHash**, set `ask_chains = {}` (empty mapping).

The API is implemented as follows:

- **InitASK(label)**:
  - If `ask_chains` contains `label`:
    Return an appropriate error
  - Set `ask_chains[label] = HKDF(ck, h || label, info="ask")`
  - Return `tmp_k2`
- **GetASK(label)**:
  - If `ask_chains` does not contain `label`:
    Return an appropriate error
  - Let `ask_ck = ask_chains[label]`
  - Let `temp_k1, temp_k2 = HKDF(ask_ck, zerolen, 2)`
  - Set `ask_chains[label] = temp_k1`
  - Return `temp_k2[..32]`, the first 32-bytes of `temp_k2`.

6. Rational

Although `GetASK(label)` and `InitASK(label)` could be combined, these are kept separate to enable the common use case where a chain is initialized before a `Split()` call (when `ck` is still available) and used after the `CipherState` objects have been instantiated.

The output of `GetASK` is truncated to 32 bytes to ensure that the API has the same behavior regardless of which hash function is employed.

7. IPR

8. Acknowledgements

This extension is based on the "Resumption PSKs" discussion between:

- str4d (str4d@i2pmail.org)
- Trevor Perrin (trevp@trevp.net)
- Christopher Wood (christopherwood07@gmail.com)
- David Wong (davidwong.crypto@gmail.com)

And in particular the ASK proposal outlined by Trevor Perrin from which both terminology and implementation details has been lifted into this document.

9. References

[1] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)." RFC 5869 (Informational); RFC Editor, May-2010. https://www.rfc-editor.org/rfc/rfc5869.txt