Noise Extension: Disco

David Wong (david.wong@nccgroup.trust)

Revision 5, 2018-01-14, unofficial/unstable

Contents

1. Introduction

This specification is extending Noise revision 34[1] and Strobe v1.0.2[2]. It relies for the most part on Noise's specification, heavily modifying some of its foundations to rely on Strobe as an opaque cryptographic primitive.

Major changes between versions of Disco are listed at the end of this document.

1.1. Motivation

Noise[1] is a framework for crypto protocols based on the Diffie-Hellman key agreement. One of Noise's property is to authenticate the protocol transcript by continuously hashing messages being sent and received, as well as continuously deriving new keys based on the output of key exchanges and previously derived keys. This interesting property stops at the end of the handshake.
Strobe[2] is a protocol framework based on a duplex construction[3]. It naturally benefits from similar property, effectively absorbing every operation to influence the next ones. The Strobe specification is comparable in aspect to Noise, but focuses only on the symmetric parts of a protocol. By merging both protocol frameworks into one, Disco achieves the following goals:

- The Noise specification is greatly simplified by removing all symmetric cryptographic algorithms and all symmetric objects. These are replaced by a single Strobe object acting as an opaque cryptographic primitive.
- The encryption of Disco messages is influenced by all previous messages that have been exchanged in addition to the output of key exchanges. This includes post-handshake messages as well.
- Implementations of the Disco extension consequently benefit from the use of Strobe which works on top of a single primitive. This allows for a drastic reduction in size of the codebase, facilitating security audits and pleasing the constraints of embedded device development.
- Thanks to Strobe's total reliance on SHA-3's internal permutation keccak, hardware support for the latter will benefit Strobe and consequently Disco. Cryptanalysis from the SHA-3 competition indicate that Strobe and Disco are built on top of solid foundations.
- The use of a Strobe library in an implementation of Disco also provides access the following functions for free: generation of random numbers, derivation of keys, hashing, encryption and authentication.

1.2. How to Read This Document and Implement Disco

To implement the Disco extension, a Strobe implementation respecting the functions of the section 3 of this document is required. None of the cipher and hash functions of Noise are required. Furthermore, the CipherState is not necessary while the SymmetricState has been simplified by Strobe calls. When implementing Noise with the Disco extension, simply ignore the CipherState section of Noise and implement the SymmetricState described in section 5 of this document. For PSK handshakes see section 6 and for advanced features, refer to section 7.

Note that like the Noise protocol framework, additional layers might need to be added on top of Disco for it to be usable (for example a framing layer to indicate the messages' lengths). For more information see Noise's section 13 on application responsabilities.

2. Protocol naming

The name of a Noise protocol extended with Disco follows the same convention, but replaces the symmetric cryptographic algorithms by the version of Strobe used:

`Noise_[PATTERN]_[KEYEXCHANGE]_STROBEvX.Y.Z`

For example, with the current version of Strobe[2] being STROBEv1.0.2:

`Noise_XX_25519_STROBEv1.0.2`

3. The `StrobeState` object

A `StrobeState` depends on a Strobe object (as defined in section 5 of the Strobe Specification) as well as the following associated constant:

- **`StrobeR`**: The blocksize of the Strobe state (computed as `N - (2*sec)/8 - 2`, see section 4 of the Strobe specification).

While a Strobe object responds to many functions (see Strobe's specification[4]), only the following ones need to be implemented in order for the Disco extension to work properly:

`InitializeStrobe(protocol_name)`: Initialize the Strobe object with a custom protocol name.

`KEY(key)`: Permutes the Strobe's state and replaces the new state with the key.

`PRF(output_len)`: Permutes the Strobe's state and removes `output_len` bytes from the new state. Outputs the removed bytes to the caller.

`send_ENC(plaintext)`: Permutes the Strobe's state and XOR the plaintext with the new state to encrypt it. The new state is replaced by the resulting ciphertext, while the resulting ciphertext is output to the caller.

`recv_ENC(ciphertext)`: Permutes the Strobe's state and XOR the ciphertext with the new state to decrypt it. The new state is replaced by the ciphertext, while the resulting plaintext is output to the caller.

`AD(additionalData)`: Absorbs the additional data in the Strobe's state.

`send_MAC(output_length)`: Permutes the Strobe's state and retrieves the next `output_length` bytes from the new state.

`recv_MAC(tag)`: Permutes the Strobe's state and compare (in **constant-time**) the received tag with the next 16 bytes from the new state.

`RATCHET(length)`: Permutes the Strobe's state and set the next `length` bytes from the new state to zero.

The following **meta** functions:

`meta_AD(additionalData)`: XOR the additional data in the Strobe's state.

The following function which is not specified in Strobe:

`Clone()`: Returns a copy of the Strobe state.

4. Post-handshake phase and modifications to the Handshake State

Processing the final handshake message via `WriteMessage()` and `ReadMessage()` now returns two new `StrobeState` objects by calling `Split()`. The first for encrypting transport messages from initiator to responder, and the second for messages in the other direction. At this point the `StrobeState` of the `SymmetricState` should not be deleted as it is the first `StrobeState` object returned by `Split()` (and will be used by the initiator to encrypt messages).

The peers can then encrypt (resp. decrypt) messages by calling `send_ENC` followed by `send_MAC` (resp. `recv_ENC` followed by `recv_MAC`) on the relevant `StrobeState` object.

5. Modifications to the Symmetric State

A SymmetricState object contains:

- **`StrobeState`**: a Strobe state responding to the functions mentioned in the previous section.
- **`isKeyed`**: a boolean value indicating if a Diffie-Hellman key exchange has already occurred.

A SymmetricState responds to the following functions:

`InitializeSymmetric(protocol_name)`: Calls `InitializeStrobe(protocol_name)` on the Strobe state.

`MixKey(input_key_material)`: Calls `AD(input_key_material)` on the Strobe state. It then sets `isKeyed` to `true`.

`MixHash(data)`: Calls `AD(data)` on the Strobe state.

`MixKeyAndHash(input_key_material)`: Calls `AD(input_key_material)` on the Strobe state.

`GetHandshakeHash()`: Calls `PRF(32)`. This function should only be called at the end of a handshake, i.e. after the Split() function has been called. This function is used for channel binding, as described in Section 11.2 of the Noise specification.

`EncryptAndHash(plaintext)`: Returns a ready to be sent payload to the caller by following these steps:

- If `isKeyed` is set to `false`, call `send_CLR(plaintext)` and return the `plaintext`.
- Call `send_ENC(plaintext)` followed by `send_MAC(16)` on the Strobe state. Return the contatenation of both results to the caller.

`DecryptAndHash(ciphertext)`: Returns the received payload by following steps:

- If `isKeyed` is set to `false`, call `recv_CLR(ciphertext)` and return the `ciphertext`.
- Otherwise, check that the length of the received `ciphertext` is at least 16 bytes. If it is not, return an error to the caller and abort the handshake.
- Call `recv_ENC(ciphertext[:-16])` and store the result in a `plaintext` buffer.
- Call `recv_MAC(ciphertext[-16:])` on the Strobe state. If `recv_MAC` returns `false`, the peer must return an error to the caller and abort the connection. Otherwise return the `plaintext` buffer.

`Split()`: Returns a pair of Strobe states for encrypting transport messages by executing the following steps:

- Let `s1` be the Strobe state and `s2` the result returned by `Clone()`.
- Calls `meta_AD("initiator")` on `s1` and `meta_AD("responder")` on `s2`.
- Calls `RATCHET(16)` on `s1` and on `s2`.
- Returns the pair `(s1, s2)`.

6. Modifications to pre-shared symmetric keys

For PSK handshakes, the "`e`" token does not need to call MixKey(e.public_key). Hence, no further modifications to the Symmetric State functions are needed for such handshakes.

7. Modifications to Advanced Features

7.1 Channel Binding

Right before calling `Split()`, a binding value could be obtained from the StrobeState by calling `PRF()`.

7.2 Rekey

To enable this, Strobe supports a `RATCHET()` function.

7.3 Out-of-order transport messages

In order to build out-of-order protocols out of Disco, the `Split()` function must return nonce-based objects. For this, the `Split()` function is modified in the next section to return a pair of `DiscoSecureChannel` objects which are defined in the section following it.

Transport messages are then encrypted and decrypted by calling `Encrypt()` and `Decrypt()` on the relevant `DiscoSecureChannel`.

7.3.1 Modifications to the `Split()` function

Modify the `Split()` function to add the following steps before returning the pair `(s1, s2)` of `Strobe` objects:

- Call `meta_RATCHET(0)` on both `s1` and `s2`.
- Create two `CipherState` named `c1` and `c2`.
- Associate `s1` (resp. `s2`) to `c1` (resp. `c2`).
- Set both the nonces `n` of `c1` and `c2` to zero.
- Return the pair `(c1, c2)`.

7.3.2 Modifications to the CipherState object

A CipherState can encrypt and decrypt data based on its associated StrobeState object as well as the following variable:

- **n**: An 8-byte (64-bit) unsigned integer nonce.

A CipherState responds to the following functions:

EncryptWithAd(ad, plaintext):

- If n is equal to 2^{64}-1 the function returns an error to the caller and aborts the Disco session.
- Create a new StrobeState by calling Clone() on the StrobeState object.
- Call AD(n) on the new StrobeState.
- Call send_ENC(plaintext) on the new StrobeState and add the result to a ciphertext buffer.
- Call send_MAC(16) on the new StrobeState and add the result to the ciphertext buffer.
- Increment the nonce n and discard the new StrobeState object.
- Return the ciphertext buffer containing the encrypted data.

DecryptWithAd(ad, ciphertext)::

- Check that the length of the received ciphertext is at least 16 bytes. If it is not, return an error to the caller and abort the session.
- If n is equal to 2^{64}-1 the function returns an error to the caller and aborts the Disco session.
- Create a new StrobeState by calling Clone() on the StrobeState object.
- Call AD(n) on the new StrobeState.
- Call recv_ENC(ciphertext[:-16]) on the new StrobeState and store the result in a plaintext buffer.
- Call recv_MAC(ciphertext[-16:]) on the new StrobeState, if the function returns false, return an error to the caller and abort the Disco session.
- Increment the nonce n and discard the new StrobeState object.
- Return the plaintext buffer containing the decrypted data.

Rekey(): calls RATCHET(16) on the Strobe Object.

7.4 Half-duplex protocols

To use Disco in half-duplex mode, modify Split() to return the StrobeState without modifications.

The same security considerations from the Noise specification applies to this section: if the two peers do not properly take turns to write and read on the channel, the protocol will fail catastrophically.

8. Security Considerations

The same security considerations that apply to both Noise and Strobe are to be considered.

9. Acknowledgements

Thanks to Trevor Perrin and Mike Hamburg for being the foundations and main help in building this specification.

Thanks to Trevor Perrin again for suggesting the name to this extension.

10. Change log

this section will be removed in the final document

draft-5:

- Added a pre-shared key section

draft-4:

- `DiscoSecureChannel` has been renamed to `CipherState`.
- Added a post-handshake phase.

draft-3:

- The specification now extends Noise draft-33.
- A `isSetKey` boolean value was added to the SymmetricState, it is set when `MixKey()` is called.
- `EncryptAndHash()` and `DecryptAndHash()` now look for the value of `isSetKey` and branch to `send_CLR()` and `recv_CLR()` if the boolean value is set to `false`.
- a `StrobeState` object has been introduced to formalize the integration of Strobe in Disco.
- Added the missing `GetHandshakeHash()` function to the `SymmetricState`.
- removed the `TAGLEN` field and set it to 16 everywhere. Following Noise's way of defining the tag length.
- Half-duplex protocols are introduced in Advanced features.
- Out-of-order protocols are introduced in Advanced features, along with `DiscoSecureChannel` objects.

draft-2:

- The `SymmetricState` object has been simplified with Strobe's calls (instead of modifying the `HandshakeState`).

draft-1:

- Protocol names don't have the symmetric algorithms, but instead the version of Strobe.
- The `CipherState` object has been removed.
- The Handshake object makes calls to Strobe functions, affecting a unique Strobe state.
- The Handshake returns two Strobe states.
- The document extends Noise draft-32.

11. References

[1] Trevor Perrin, "Noise protocol framework." 2017. http://noiseprotocol.org

[2] Mike Hamburg, "The STROBE protocol framework." Cryptology ePrint Archive, Report 2017/003, 2017. http://eprint.iacr.org/2017/003

[3] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications." Cryptology ePrint Archive, Report 2011/499, 2011. https://eprint.iacr.org/2011/499

[4] Mike Hamburg, "The STROBE protocol framework specification." 2017. https://strobe.sourceforge.io/