Signatures for Noise

Trevor Perrin(noise@trevp.net)

Revision 1, 2018-12-16, unofficial/unstable

Contents

1. Introduction

This document describes Noise handshake modifiers and tokens for replacing DH-based authentication with signature-based authentication.

2. Overview

Noise handshakes depend on some Diffie-Hellman function (usually an Elliptic Curve Diffie-Hellman function such as X25519).

Noise uses this function both for forward-secrecy (supplied by the `"ee"` token) as well as authentication (supplied by the `"se"` and `"es"` tokens).

In some cases the parties might prefer to use signing key pairs for authentication, rather than DH key pairs. For example, maybe they have an existing PKI based on signing keys, or would like to use the signing keys to sign other pieces of data.

This document describes Noise handshake modifers (`"sig"`, `"sigi"`, and `"sigr"`) and new tokens (`"sig1"`, `"sig2"`, etc.) that replace DH authentication with signatures.

3. Crypto functions

A signature algorithm in Noise is defined by the following functions and constants:

- **`GENERATE_SIGNATURE(key_pair, message)`**: Returns a signature on the input, of `SIGLEN` bytes.
- **`VERIFY_SIGNATURE(public_key, message, signature)`**: Verifies a signature on the `message` and returns true or false.
- **`SIGPUBLICKEYLEN`** = A constant specifying the size in bytes of signature public keys.
- **`SIGLEN`** = A constant specifying the size in bytes of signatures.

4. Signature tokens

New tokens are introduced by this document. These tokens have a number suffix which refers to the algorithm at the specified index of the Noise public key algorithms name section. The first algorithm specified in this section will be a DH algorithm, and the next algorithm (index 1) might be a signature algorithm. For example, the name `"25519_Ed25519"` would refer to the `25519` DH algorithm with the `Ed25519` signature algorithm at index 1.

- **`"s1", "s2", ...`** = These tokens direct the sender to transfer a static signature public key (instead of DH public key) to the recipient.
- **`"sig1", "sig2", "sig3", ...`** = These tokens direct the sender to call `GENERATE_SIGNATURE()` using their static signature key pair and using the hash value `h` as the signature's `message` input. The signature is then transferred to the recipient as if it was a static DH public key (i.e. using `EncryptAndHash()` if a key is available). On receiving this token, the recipient will call `VERIFY_SIGNATURE()` and will abort the protocol if the signature fails to verify (i.e. false is returned).

5. Signature modifiers

The `"sig"` pattern modifier replaces all `"se"` and `"es"` tokens with `"sig"` tokens. When the `"sig"` modifier is used, the DH name section must contain a signature algorithm name (along with other algorithm names, separated by plus signs).

The `"sig"` token has a number appended to indicate the index of the public-key algorithm name it refers to, and the `"s"` tokens have the same number appended. For example, if the Noise pattern name is `"XXsig"` then `"s1"` and `"sig1"` tokens will be used, and the Noise public-key algorithm name section will be something like `"448_Ed25519"`. If the Noise pattern name is `"XXhfs_sig"` then algorithm 1 is used for hybrid-forward secrecy, so `"s2"` and `"sig2"` tokens will be used, and the Noise pattern name will be something like `"448_NewHope_Ed25519"`.

The `"sig"` modifier can only be used with patterns where `"se"` is not sent by the responder and `"es"` is not sent by the initiator, and `"ss"` does not appear. Attempting to apply it other patterns is invalid.

The `"sigi"` modifier only replaces `"se"` with a signature token, and can only be used with patterns where `"se"` is sent by the initiator.

The `"sigr"` modifier only replaces `"es"` with a signature token, and can only be used with patterns where `"es"` is sent by the responder.

The table below lists some example unmodified patterns on the left, and some signature-based patterns on the right:

```
NK1:                           NK1sig:
  <- s                           <- s1
  ...                            ...
  -> e                           -> e
  <- e, ee, es                   <- e, ee, sig1


NX:                            NXsig:
  -> e                           -> e
  <- e, ee, s, es                <- e, ee, s1, sig1


XN:                            XNsig:
  -> e                           -> e
  <- e, ee                       <- e, ee
  -> s, se                       -> s1, sig1
```

```
XK:
  <- s
  ...
  -> e, es
  <- e, ee
  -> s, se
```

```
XK1:
  <- s
  ...
  -> e
  <- e, ee, es
  -> s, se
```

```
X1K1:
  <- s
  ...
  -> e
  <- e, ee, es
  -> s
  <- se
```

```
XX:
  -> e
  <- e, ee, s, es
  -> s, se
```

```
X1X:
  -> e
  <- e, ee, s, es
  -> s
  <- se
```

```
XX1:
  -> e
  <- e, ee, s
  -> es, s, se
```

```
XKsigi:
  <- s
  ...
  -> e, es
  <- e, ee
  -> s1, sig1
```

```
XK1sig:
  <- s1
  ...
  -> e
  <- e, ee, sig1
  -> s1, sig1
```

```
X1K1sigr:
  <- s1
  ...
  -> e
  <- e, ee, sig1
  -> s
  <- se
```

```
XXsig:
  -> e
  <- e, ee, s1, sig1
  -> s, sig1
```

```
X1Xsigr:
  -> e
  <- e, ee, s1, sig1
  -> s
  <- se
```

```
XX1sigi:
  -> e
  <- e, ee, s
  -> es, s1, sig1
```

```
K1N:                            K1Nsig:
  -> s                           -> s1
  ...                            ...
  -> e                           -> e
  <- e, ee                       <- e, ee
  -> se                          -> sig1


K1K:                            K1Ksigi:
  -> s                            -> s
  <- s                            <- s
  ...                             ...
  -> e, es                        -> e, es
  <- e, ee                        <- e, ee
  -> se                           -> sig1


KK1:                            KK1sigr:
  -> s                            -> s
  <- s                            <- s1
  ...                             ...
  -> e                            -> e
  <- e, ee, se, es                <- e, ee, se, sig1


K1K1:                           K1K1sig:
  -> s                            -> s1
  <- s                            <- s1
  ...                             ...
  -> e                            -> e
  <- e, ee, es                    <- e, ee, sig1
  -> se                           -> sig1


KX:                             KXsigr:
  -> s                            -> s
  ...                             ...
  -> e                            -> e
  <- e, ee, se, s, es             <- e, ee, se, s1, sig1
```

```
K1X:
  -> s
  ...
  -> e
  <- e, ee, s, es
  -> se
```

```
K1X1:
  -> s
  ...
  -> e
  <- e, ee, s
  -> se, es
```

```
I1N:
  -> e, s
  <- e, ee
  -> se
```

```
I1K:
  <- s
  ...
  -> e, es, s
  <- e, ee
  -> se
```

```
IK1:
  <- s
  ...
  -> e, s
  <- e, ee, se, es
```

```
I1K1:
  <- s
  ...
  -> e, s
  <- e, ee, es
  -> se
```

```
K1Xsig:
  -> s1
  ...
  -> e
  <- e, ee, s1, sig1
  -> sig1
```

```
K1X1sigi:
  -> s1
  ...
  -> e
  <- e, ee, s
  -> sig1, es
```

```
I1Nsig:
  -> e, s1
  <- e, ee
  -> sig1
```

```
I1Ksigi:
  <- s
  ...
  -> e, es, s
  <- e, ee
  -> sig1
```

```
IK1sigr:
  <- s1
  ...
  -> e, s
  <- e, ee, se, sig1
```

```
I1K1sig:
  <- s1
  ...
  -> e, s1
  <- e, ee, sig1
  -> sig1
```

```
IX:                               IXsigr:
  -> e, s                           -> e, s
  <- e, ee, se, s, es               <- e, ee, se, s, sig1


I1X:                              I1Xsig:
  -> e, s                           -> e, s1
  <- e, ee, s, es                   <- e, ee, s1, sig1
  -> se                             -> sig1


I1X1:                             I1X1sigi:
  -> e, s                           -> e, s
  <- e, ee, s                       <- e, ee, s
  -> se, es                         -> sig1, es
```

6. Signature functions

6.1. The `Ed25519` signature functions

- **`GENERATE_SIGNATURE`** and **`VERIFY_SIGNATURE`**: Executes the Ed25519 algorithm from [1].
- **`SIGPUBLICKEYLEN`** = 32
- **`SIGLEN`** = 64

7. Security considerations

Reusing a key pair for Noise signatures and non-Noise signatures introduces the risk of cross-protocol attacks. The non-Noise signatures might be substituted by an attacker for a Noise signature, or vice versa.

To avoid this threat, it must be impossible for the input to the non-Noise signature to be a valid Noise `h` value. For example, the non-Noise signatures might be applied to messages that are always a different length from a Noise `h` value.

8. IPR

This document is hereby placed in the public domain.

9. Acknowledgements

Justin Cormack helped with discussion and design.

10. References

[1] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, “High-speed high-security signatures.” 2012. https://ed25519.cr.yp.to/ed25519-20110926.pdf