The NoiseSocket Protocol

Alexey Ermishkin (scratch@virgilsecurity.com)

Revision 0draft, 2017-07-12

Contents

Abstract

NoiseSocket is an extension of the Noise Protocol Framework (developed by the authors of Signal and currently used by WhatsApp) that enables quick and seamless secure link between multiple parties with minimal code space overhead, small keys, and extremely fast speed. It uses raw public keys, modern AEAD ciphers and hash functions. NoiseSocket is designed to overcome the shortcomings of existing TLS implementations and targets IoT devices, microservices, back-end applications such as datacenter-to-datacenter communications, and use cases where third-party certificate of authority infrastructure is not optimal.

1. Messages

There are two types of messages which differ by structure:

- Section 1.1 Handshake message
- Section 1.2 Transport message

1.1. Handshake message

For the simplicity of processing, all handshake messages have identical structure:

- negotiation_data_len (2 bytes)
- negotiation_data
- noise_message_len (2 bytes)
- noise_message

All numbers are big-endian.

They are sent according to the corresponding Noise protocol.

To implement Noise_XX 3 messages need to be sent:

```
-> ClientHello
<- ServerAuth
-> ClientAuth
```

2 for Noise_IK

```
-> ClientHello
<- ServerAuth
```

3 If Fallback is used:

```
-> ClientHello
<- ServerHello
-> ClientAuth
```

1.1.1. Negotiation data

The negotiation_data field is used to identify the protocols used, versions, signs of using a fallback protocol and other data that must be processed before reading the actual noise_message.

Though it can be present in every handshake message, it can safely be used only when it is included into the Noise handshake through Prologue or other mechanisms like calling MixHash() before writing the message

An example first message negotiation_data which allows to determine which algorithms and pattern were used:

- version_id (2 bytes) (has value `1` by default)
- pattern_id (1 byte)
- dh_id (1 byte)
- cipher_id (1 byte)
- hash_id (1 byte)

pattern_id, dh_id, cipher_id and hash_id can be taken from the Noise-c implementation

For example, NoiseXX_25519_AESGCM_SHA256 would be

- pattern_id : 9
- dh_id : 1
- cipher_id : 2
- hash_id : 3

An example of second message negotiation data:

- version_id (1 byte) (usually same as client)
- status_id (1 byte) (0 if handshake continues, 1 if fallback, 0xFF if server does not understand client)

Third message's negotiation data is always zero length.

1.2. Transport message

Each transport message has a special 'data_len' field inside its plaintext payload, which specifies the size of the actual data. Everything after the data is considered padding. 65517 is the max value for data_len: 65535 (noise_message_len) - 16 (for authentication tag) - 2 (for data_len field itself)

The encrypted packet has the following structure:

- packet_len (2 bytes)
- encrypted data

Plaintext payload has the following structure:

- data_len (2 bytes)
- data
- remaining bytes: padding

2. Prologue

Client uses following extra data and fields from the first message to calculate the Noise prologue:

- "NoiseSocketInit" string
- negotiation_data_len

- negotiation_data

If server decides to start a new protocol instead of responding to the first handshake message, it calculate the Noise prologue using the **full first message contents** plus the length and negotiation_data of its own response. String identifier also changes to "NoiseSocketReInit".

Thus the prologue structure:

- "NoiseSocketReInit" string
- negotiation_data_len
- negotiation_data
- noise_message_len
- noise_message
- negotiation_data_len
- negotiation_data

5. API

Client and server calls these in sequence.

`InitializeClient`:

- INPUT: dh, cipher, hash
- OUTPUT: session object

`WriteHandshakeMessage`:

- INPUT: [negotiation_data][, cleartext_body]
  - negotiation_data is zero-length if omitted
  - cleartext_body is zero-length if omitted
- OUTPUT: handshake_message

`PeekHandshakeMessage`:

- INPUT: handshake_message
- OUTPUT: negotiation_data

`ReadHandshakeMessage`:

- INPUT: handshake_message
- OUTPUT: message_body

After WriteClientAuth / ReadClient, both parties can call Write and Read:

`Write`:

- INPUT: transport_body[, padded_len]
  - padded_len is zero (no padding) if omitted
- OUTPUT: transport_message

`Read`:

- INPUT: transport_message
- OUTPUT: transport_body

7. IPR

The NoiseSocket specification (this document) is hereby placed in the public domain.

8. References