The NoiseLink Protocol

Alexey Ermishkin (scratch@virgilsecurity.com)

Revision 0draft, 2017-07-12

Contents

Abstract

NoiseLink (ex. 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. NoiseLink 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 four types of messages:

- Section 1.1 ClientHello
- Section 1.2 ServerAuth
- Section 1.3 ClientAuth
- Section 1.4 Transport message

1.1. ClientHello

Client chooses a protocol version and sends it along with an optional negotiation data and Noise message to server using the following structure:

- packet len
- client_version (4 bytes)
- negotiation data len (2 bytes)
- negotiation data
- remaining bytes: noise message

Version choice and negotiation data contents are up to the app (see section 3).

1.2. ServerAuth

Server answers with the server_version == client_version and the Noise message using the following structure:

- packet len (2 bytes)
- server_version (4 bytes)
- remaining bytes: noise message

If server is able to process ClientHello, it must use server_version == client_version

1.3. ClientAuth

If Client needs to authenticate further it sends the ClientAuth packet which has the following structure:

- packet length (2 bytes)
- noise message

1.4. 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 length (2 bytes)
- encrypted data
- authentication tag (16 bytes)

authentication tag is added by the AEAD algorithm automatically when encrypting

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 prologue:

- "NoiseLinkInit" string
- client_version
- negotiation data len
- negotiation_data

If server does not understand clients version, it uses the following data as the prologue:

- "NoiseLinkReinit" string
- client_version
- negotiation data len
- negotiation_data
- noise_message
- server_version

3. Versions

Version 0xFFFFFFFF is reserved for server to answer that it's unable to process the first message.

In all other cases versions are up to the app

One of the possible approaches to using the version could be the follows:

Use 4 bit fields inside 32 bit number for defining Noise Pattern and used algorithms. Also, leave first 4 bits as a reserved “proto_version” field

proto_version (4 bits)
reserved (12 bits)
pattern (4 bits)
dh (4 bits)
cipher (4 bits)
hash (4 bits)

Total: 32 bits

Proposed field values:

- proto_version: zero
- pattern:
  - XX: 0
  - IK: 1
- dh:
  - Curve25519: 0
  - Curve448: 1
- cipher:
  - AESGCM: 0
  - ChaChaPoly1305: 1
- hash:
  - SHA256: 0
  - SHA512: 1
  - Blake2s: 2
  - Blake2b: 3

5. API

We present a set of methods which will help to implement NoiseSocket flow

6. Test vectors

7. IPR

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

8. References