carrier network protocol

Arvid E. Picciani (arvid@devguard.io)

Revision 1, July 25, 2018

Contents

1 Channels and Messages

Channels are peer to peer encrypted udp pairs similar to wireguard[1]. They are routed by channel id rather than IP address, allowing a stream to continue seamlessly when migrating to a different IP address.

A stream of messages is a concept similar to QUIC[2], except that the message bounderies are kept intact to support soft-realtime streaming similar to MQTT[3].

Messages will arrive at the receiver authenticated and in the order they where sent.

2 Cryptosystem

2.1 Handshake

A simple Diffie-Hellman Key exchange is done before messages can be transported, based on the the Noise Protocol Framework [4].

We build on wireguard[1]'s idea of not sending responses to unauthenticated packages by using the noise NK pattern. An inititor has to obtain the responders static key from a side channel first before sending the first packet.

Unlike wireguard, we do not use a static key for the initiator, because its identity is established using signatures instead. The signatures are sent in the first message, weakening forward secrecy for when the responders static key is stolen.

However, since we use ED25519 signatures as identities, the static X25519 keys can be rotated frequently.

If u describes an Ed25519 identity and u(h) a signature over the session hash, The pattern is as follows:

```
NKSig:
  <- s
  ...
  -> e, es, [u, u(h)]
  <- e, ee, [u(h)]
```

The initiator starts by sending its ephermal key and a signature. To prevent replay of this message, the signature contains a short lived expiration date.

The responder may then check the signature for authentication. It can choose to not respond at all, if the signature is not authorized. If the signature is fine, it responds with its own identity, which the receipient can use to do additional verification of the side channel.

In case of a compromised side channel for ED25519 to X25519 mapping, the initiators identity is leaked, but none of the data.

2.2 Packet Format

2.2.1 from initiator to responder:

The first message from the initiator is different in that the Receiver is set to zero. This is the equivalent of SYN, i.e. opening a new transport channel.

Note that the packet counter is not nessesarily 1. It may be higher if earlier handshake packages are lost.

```
-------------------------------------------------------
| Vers = 0x08 (1 byte) | Reserved = 0xffffff (3 bytes) |
-------------------------------------------------------
```

```
| Receiver Channel = 0x00000000 (4 bytes)              |
-------------------------------------------------------
| Packet Counter  = 1 (8 bytes)                        |
-------------------------------------------------------
| Ephermal (32 bytes)                                  |
-------------------------------------------------------
| Authentication Tag (12 bytes)                        |
-------------------------------------------------------
             ------ encrypted ----
-------------------------------------------------------
| Sender Channel =  Random (4 bytes)                   |
-------------------------------------------------------
| Length of Stake  (2 bytes big endian)                |
-------------------------------------------------------
| Stake                                                |
-------------------------------------------------------
| Padding to 255 bytes boundary                        |
-------------------------------------------------------
```

Packet Counter

8 byte big-endian integer that increases for each sent packet. it is never reused and each new packet must always have a packet counter higher than the previous packet.

The packet counter is for replay-mitigation based on Appendix C of RFC2401[5].

It is also used for measuring packet loss. Packets are never resent, instead reordering is done based on individual stream counters.

TODO: quic uses 32bit, this seems rather small.

2.2.2 from responder to initiator:

transport looks identical in plaintext to a handshake response. When packet reordering occures on the wire, this would lead an initiator to decode a transport message as handshake, and therefor taking random ciphertext as ephermal.

This is already mitigated by waiting for the application header.

```
-------------------------------------------------------
| Vers = 0x08 (1 byte) | Reserved = 0xffffff (3 bytes) |
-------------------------------------------------------
| Receiver Channel = Initiator (4 bytes)               |
-------------------------------------------------------
| Packet Counter  = 1 (bytes unsigned big endian)      |
-------------------------------------------------------
| Ephermal (32 bytes)                                  |
-------------------------------------------------------
| Authentication Tag (12 bytes)                        |
-------------------------------------------------------
```

```
             ------ encrypted ----
-----------------------------------------------------------
| Responder Channel = Random (4 bytes)                    |
-----------------------------------------------------------
| Length of Stake  (2 bytes big endian)                   |
-----------------------------------------------------------
| Stake                                                   |
-----------------------------------------------------------
| Padding to 255 bytes boundary                           |
-----------------------------------------------------------
```

2.2.3 in transport mode

```
-----------------------------------------------------------
| Vers = 0x08 (1 byte) | Reserved = 0xffffff (3 bytes)    |
-----------------------------------------------------------
| Receiver Channel (4 bytes)                              |
-----------------------------------------------------------
| Packet Counter  = 2 (bytes unsigned big endian)         |
-----------------------------------------------------------
| Authentication Tag (12 bytes)                           |
-----------------------------------------------------------
             ------ encrypted ----
-----------------------------------------------------------
| Frame 1                                                 |
-----------------------------------------------------------
| Frame 2                                                 |
-----------------------------------------------------------
| Frame ..                                                |
-----------------------------------------------------------
| Padding to 255 bytes boundary                           |
-----------------------------------------------------------
```

2.3 Frame types

TODO: implement stream multiplexing

Value	name
0x00	Padding
0x01	Ack
0x02	Ping
0x05	Message

2.3.1 0x00 Padding

Padding in the form of 0x00 bytes can ocure before a frame header, or after a completed frame and must be skipped until there is a value that is not 0x00 (the next useful frame header).

2.3.2 0x01 Ack

```
-------------------------------------------------------
| Frame Type = 0x01 (1 byte)                          |
-------------------------------------------------------
| Ack Delay = delay in ms (2 byte unsigned big endian) |
-------------------------------------------------------
| Ack Count (2 byte unsigned big endian)              |
-------------------------------------------------------
| Ack 1   (8 bytes unsigned big endian)               |
-------------------------------------------------------
| Ack 2   (8 bytes unsigned big endian)               |
-------------------------------------------------------
| Ack ..  (8 bytes unsigned big endian)               |
-------------------------------------------------------
```

the acks are sorted by largest first

2.3.3 0x02 Ping

```
-------------------------------------------------------
| Frame Type = 0x02 (1 byte)                          |
-------------------------------------------------------
```

2.3.4 0x05 Message

```
-------------------------------------------------------
| Frame Type = 0x05 (1 byte)                          |
-------------------------------------------------------
| Reserved (8 bits)                                   |
-------------------------------------------------------
| Message Counter (8 bytes unsigned big endian)       |
-------------------------------------------------------
| Data Size (2 byte unsigned big endian)              |
-------------------------------------------------------
| Data                                                |
-------------------------------------------------------
```

References

[1] J. A. Donenfeld, “WireGuard: Next generation kernel network tunnel,” 30-06-2018. https://www.wireguard.com/papers/wireguard.pdf

[2] J. Iyengar and M. Thomson, “QUIC: A udp-based multiplexed and secure transport,” IETF Secretariat; Working Draft, Internet-Draft draft-ietf-quic-transport-13, 2018. http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-13.txt

[3] A. Banks and R. Gupta, “MQTT version 3.1.1,” 3.1.1, 29-10-2014. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[4] T. Perrin, “The noise protocol framework,” Revision 34, 11-07-2018. http://noiseprotocol.org/noise.pdf

[5] S. Kent and R. Atkinson, “Security architecture for the internet protocol,” RFC Editor; Internet Requests for Comments; RFC Editor, RFC 2401, 1998.