carrier network protocol

Arvid E. Picciani (arvid@devguard.io)

Revision 1, December 19, 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 Message broker

a peer on the network can open itself to receiving messages by signing itself into the global services table.

```
message AnnounceRequest {
    bytes key = 1;
}
signature
```

the static key is used to start 0-RTT encryption to the service. Anyone one the network can send a single message to service, which is already encrypted. This allows for command and control servers to be stateless (for example PHP websites). This is the equivalent of QOS level 0 in MQTT.

TODO: IoT systems are usuually vastly less powereful than a server on the internet, making it vulternable to DoS if anyone on the internet could send it messages. To create power-equilibrium, a stake needs to be included by the sender.

3 Cryptosystem

3.1 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, somewhat weakening forward secrecy for when the responders static key is stolen. For this reason, the first message only contains identity signatures. An attacker will only be able to make out which identity connected, but not for what purpose. This is identical to wireguard, which uses the X25519 key as identity instead.

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), t]
  <- e, ee
```

The initiator starts by sending its identity and a semi-static x25519 which is generated before boot, and then signed by its actual ed25519 identity. This is done, so that the ed25519 secret can be kept offline. If a semi-static exchange identity is compromised, the offline key can be used to generate a new one with a new serial number.

A serial number is any integer that always must go up for the lifetime of the offline identity. The responder keeps track of this number per identity and rejects any handshake that does not increase it. This prevents both replay attacks as well as using stolen keys.

The serial does not actually have to be based on a clock, as long as the sender guarantees that it always goes up, even after reboot. The network will keep a distributed append-only log of that number.

The epoch should however be synchronized with the network ocassionally, since at some point we might want to purge the log of used identities. Any identity before the cutoff epoch will be invalid.

3.2 Packet Format

3.2.1 from iniiator to responder:

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) |
------------------------------------------------------------
| Routing Key = 0 (63 bits) | Direction = 0 (1bit)     |
------------------------------------------------------------
| Packet Counter = 1 (8 bytes unsigned big endian)     |
------------------------------------------------------------
              ------- noise -------
------------------------------------------------------------
| Authentication Tag (16 bytes)                        |
------------------------------------------------------------
| Ephermal (32 bytes)                                  |
------------------------------------------------------------
              ------ encrypted ----
------------------------------------------------------------
| Effective Identity (32 bytes)                        |
------------------------------------------------------------
| Timestamp (8 bytes unsigned big endian)              |
------------------------------------------------------------
| Depth of Delegation (2 bytes unsigned big endian)    |
------------------------------------------------------------
| Delegate 1 Identity (32 bytes)                       |
------------------------------------------------------------
| Delegate 1 Epoch (4 bytes unsigned int)              |
------------------------------------------------------------
| Delegate 1 Attestation Signature (32 bytes)          |
------------------------------------------------------------
| 0x00 Padding to 255 bytes boundary                   |
------------------------------------------------------------
| Handshake Signature (64 bytes)                       |
```

```
------------------------------------------------------
```

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].

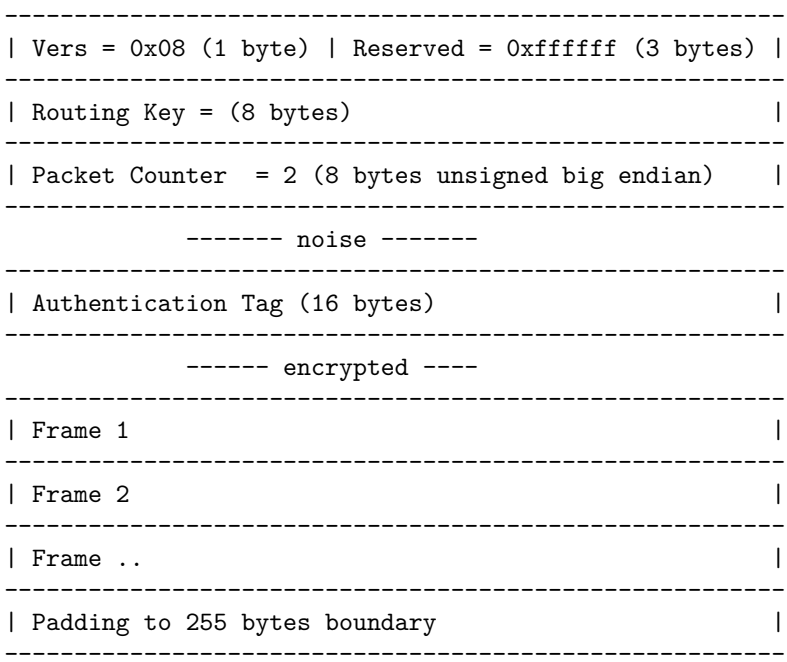
It is also used for measuring packet loss. Packets are never resent, instead reordering is done based on individual stream counters.

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

3.2.2 from responder to initiator

```
------------------------------------------------------
| Vers = 0x08 (1 byte) | Reserved = 0xffffff (3 bytes) |
------------------------------------------------------
| Routing Key = (63 bits) | Direction = 1 (1bit)       |
------------------------------------------------------
| Packet Counter  = 1 (8 bytes unsigned big endian)    |
------------------------------------------------------
             ------- noise -------
------------------------------------------------------
| Authentication Tag (16 bytes)                        |
------------------------------------------------------
| Ephermal (32 bytes)                                  |
------------------------------------------------------
             ------ encrypted ----
------------------------------------------------------
| Identity (32 bytes)                                  |
------------------------------------------------------
| Timestamp (8 bytes unsigned big endian)              |
------------------------------------------------------
| Number of Certificates (2 bytes unsigned big endian) |
------------------------------------------------------
| Certificate 1 Length (2 bytes unsigned big endian)   |
------------------------------------------------------
| Certificate 1                                        |
------------------------------------------------------
| more Certificats....                                 |
------------------------------------------------------
| Padding to 255 bytes boundary                        |
------------------------------------------------------
| Signature (64 bytes)                                 |
------------------------------------------------------
```

3.2.3 in transport mode



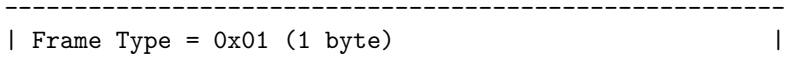
3.3 Frame types

Value	name
0x00	Padding
0x01	Ack
0x02	Ping
0x03	Disconnect
0x04	Open
0x05	Stream
0x06	Close
0x07	Configure

3.3.1 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).

3.3.2 0x01 Ack



```
-------------------------------------------------------
| 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

3.3.3 0x02 Ping

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

3.3.4 0x03 Disconnect

```
-------------------------------------------------------
| Frame Type = 0x03 (1 byte)                          |
-------------------------------------------------------
```

Tells the other side to stop sending any packets, including retransmissions, since the peer is no longer available.

A peer should send close to all streams first and wait for their acks, since an out of order disconnect will terminate all streams, even if they still have packets in flight.

3.3.5 0x04 Open

```
-------------------------------------------------------
| Frame Type = 0x04 (1 byte)                          |
-------------------------------------------------------
| Stream Id (4 bytes unsigned big endian)             |
-------------------------------------------------------
| Data Size (2 byte unsigned big endian)              |
-------------------------------------------------------
| Data                                                |
-------------------------------------------------------
```

Open is followed by stream messages, and then eventually by a close, creating an ordered reliable stream of messages.

It is the first packet within an ordered stream, with order id 1, and must be acked.

Stream id can be arbitrarily chosen by each peer so that: - 0x00 is never chosen - the

initiator of the channel uses ODD-numbered stream ids, - the responder uses EVEN-numbered stream ids.

3.3.6 0x05 Stream

```
------------------------------------------------------
| Frame Type = 0x05 (1 byte)                         |
------------------------------------------------------
| Stream Id (4 bytes unsigned big endian)            |
------------------------------------------------------
| Order (8 bytes unsigned big endian)                |
------------------------------------------------------
| Data Size (2 byte unsigned big endian)             |
------------------------------------------------------
| Data                                               |
------------------------------------------------------
```

order starts at 2, since 1 is header

3.3.7 0x06 Close

```
------------------------------------------------------
| Frame Type = 0x06 (1 byte)                         |
------------------------------------------------------
| Stream Id (4 bytes unsigned big endian)            |
------------------------------------------------------
| Order (8 bytes unsigned big endian)                |
------------------------------------------------------
```

This is the last message in a stream. The sender may still be open to receiving Messages but will not send any new ones.

Close Messages must also be reordered before handling so that any preceeding data is handled first.

3.3.8 0x07 Config

```
------------------------------------------------------
| Frame Type = 0x07 (1 byte)                         |
------------------------------------------------------
| Configurations (1 bytes)                           |
|   10000 0000 Keepalive                             |
|   01000 0000 Sleeping                              |
------------------------------------------------------
| Data Len (2 bytes unsigned big endian)             |
------------------------------------------------------
             ------ data ----
------------------------------------------------------
| Seconds (4 bytes unsigned big endian)              |
------------------------------------------------------
```

Configuration can be sent by any peer and must be acked. A parsers must always read 'Data Len' many bytes after the flags field, even if it cannot interpret them due to version incompatbility.

Keepalive sets the period after which a ping packet is sent by a peer, if no other packet was sent in the period. This can be used to tune for aggressive firewalls, but it makes most sense in combination with the sleeping flag.

Sleeping indicates that within the next Idle period, the sending peer will be unresponsive because it is sleeping. The other side must stop sending any packets except ack until he keepalive period expired, and it must not disconnect the peer due to unresponsiveness.

If a peer cannot accept an excessive sleep period, it must respond with Disconnect instead of Ack.

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.