# IPsec encapsulation over TCP

Sabrina Dubroca sd@queasysnail.net

IPsec workshop 2019

# UDP encapsulation

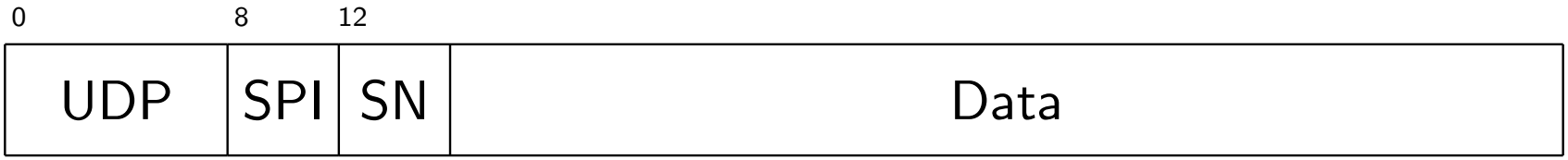


Figure: ESP over UDP packet format


Figure: IKE over UDP port 4500 packet format

- RX node uses the first 32 bits of UDP payload to differentiate ESP/IKE

# RFC 8229: packet formats

TCP stream, composed of concatenated messages

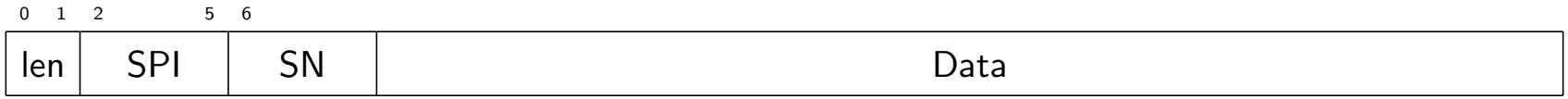


Figure: ESP over TCP message format

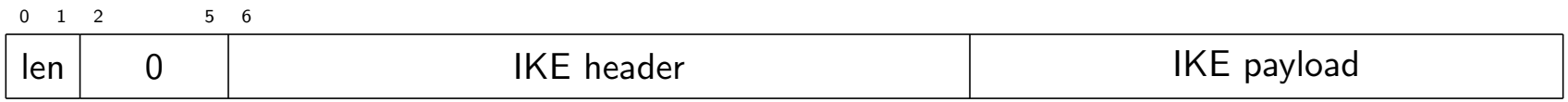


Figure: IKE over TCP message format

# Userspace API

# Userspace API: creating XFRM states

- almost identical to UDP encap: `s/espinudp/espintcp/`

```
CPORT=<local port of the client socket>
SPORT=4500
CADDR=<IP address of the client>
SADDR=<IP address of the server>

ip xfrm state add src $CADDR dst $SADDR proto esp spi $SPI1  \
                  aead 'rfc4106(gcm(aes))' $KEY $ICVLEN      \
                  mode transport sel src $CADDR dst $SADDR   \
                  encap espintcp $CPORT $SPORT 0.0.0.0
ip xfrm state add src $SADDR dst $CADDR proto esp spi $SPI2  \
                  aead 'rfc4106(gcm(aes))' $KEY $ICVLEN      \
                  mode transport sel src $SADDR dst $CADDR   \
                  encap espintcp $SPORT $CPORT 0.0.0.0
```

# Userspace API: client program

```
sock = socket(AF_INET, SOCK_STREAM, 0);

struct xfrm_userpolicy_info policy = {
        .action = XFRM_POLICY_ALLOW,
        .sel.family = AF_INET,
};

policy.dir = XFRM_POLICY_OUT;
setsockopt(sock, IPPROTO_IP, IP_XFRM_POLICY,
           &policy, sizeof(policy));
policy.dir = XFRM_POLICY_IN;
setsockopt(sock, IPPROTO_IP, IP_XFRM_POLICY,
           &policy, sizeof(policy));
setsockopt(sock, SOL_TCP, TCP_ULP,
           "espintcp", sizeof("espintcp"));
connect(sock, ...);
```

# Socket behavior

- TCP socket, but behaves like a UDP socket once encapsulation is enabled
- length prefix and non-ESP marker added/removed by kernel: userspace handles pure IKE messages
- one `send()` = 1 IKE message
  - `MSG_MORE` not implemented
- one `recv()` = 1 IKE message
  - recv buffer smaller than actual message ⇒ partial read, rest of the message dropped
  - `MSG_PEEK` returns the first N bytes of the message at the head of the receive queue

# Kernel implementation

# Overview

- minimal changes outside of XFRM (some skb helpers)
- stream parser to extract messages
- reuse ESP implementation

# RX handling

- data arrives on TCP encap socket → `__strp_rcv()` → `parse_msg` + `rcv_msg`
- `parse_msg`
  - reads length field
- `rcv_msg`
  - reads SPI/non-ESP marker
  - continues processing as ESP or IKE
- ESP messages → XFRM → decrypt/verify
- IKE messages → userspace queue → `recv()` (via custom `recvmsg` op)

# TX handling

- IKE daemon → `send()` → custom `sendmsg` op → add length prefix/non-ESP marker → enqueue to TCP
- data packets → XFRM → add length prefix → enqueue to TCP

# TX handling: interleaving of messages

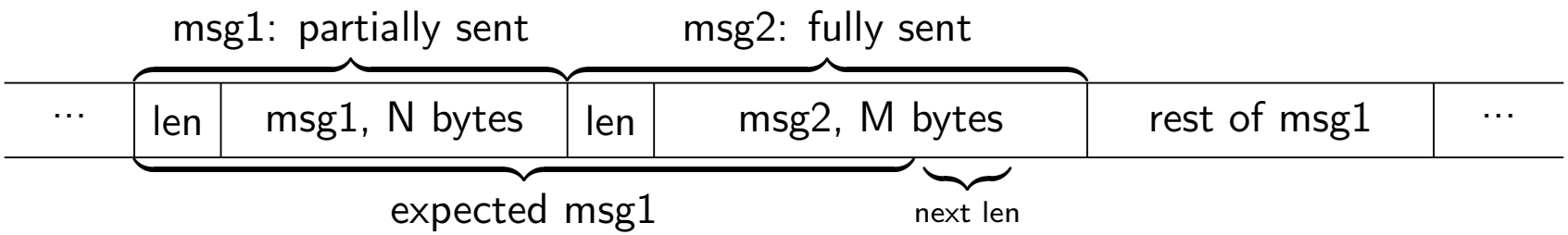


Figure: Interleaving problem between IKE and ESP

- avoid interleaving messages over TCP
  - temporary "queue" in front of the TCP socket
  - keep that partially-sent message on the queue