IPsec Crypto Offload To Network Devices

Boris Pismenny

Netdev 1.2

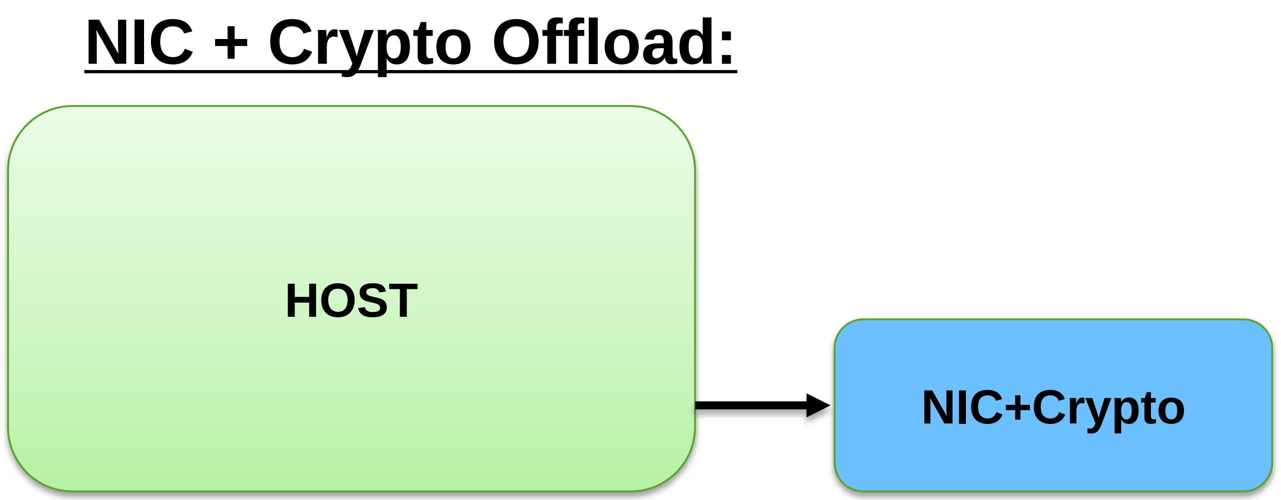
Connect. Accelerate. Outperform.™

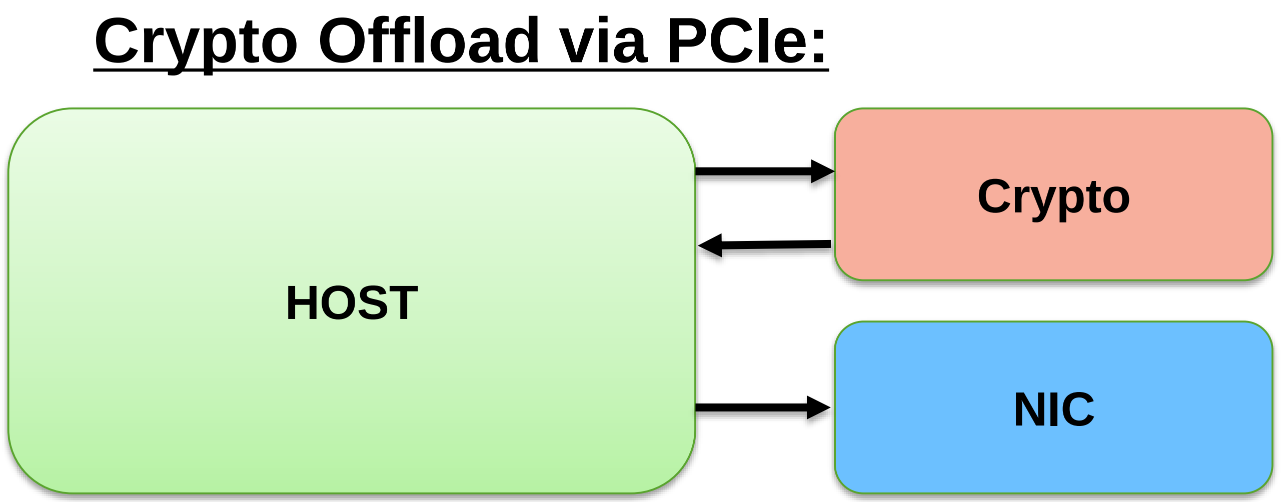
Abstract

- Motivation
- Model
- Challenges
  - LSO
  - IV processing (seqiv)
  - Checksum offload
- Performance
- Status
- Limitations
- Future
- XFRM Device Ops
- Transmit and Receive Flows

Motivation

- Encryption is CPU intensive
- Crypto offload today via PCIe requires passing the PCIe thrice
- LSO and checksum offload aren't supported for IPsec





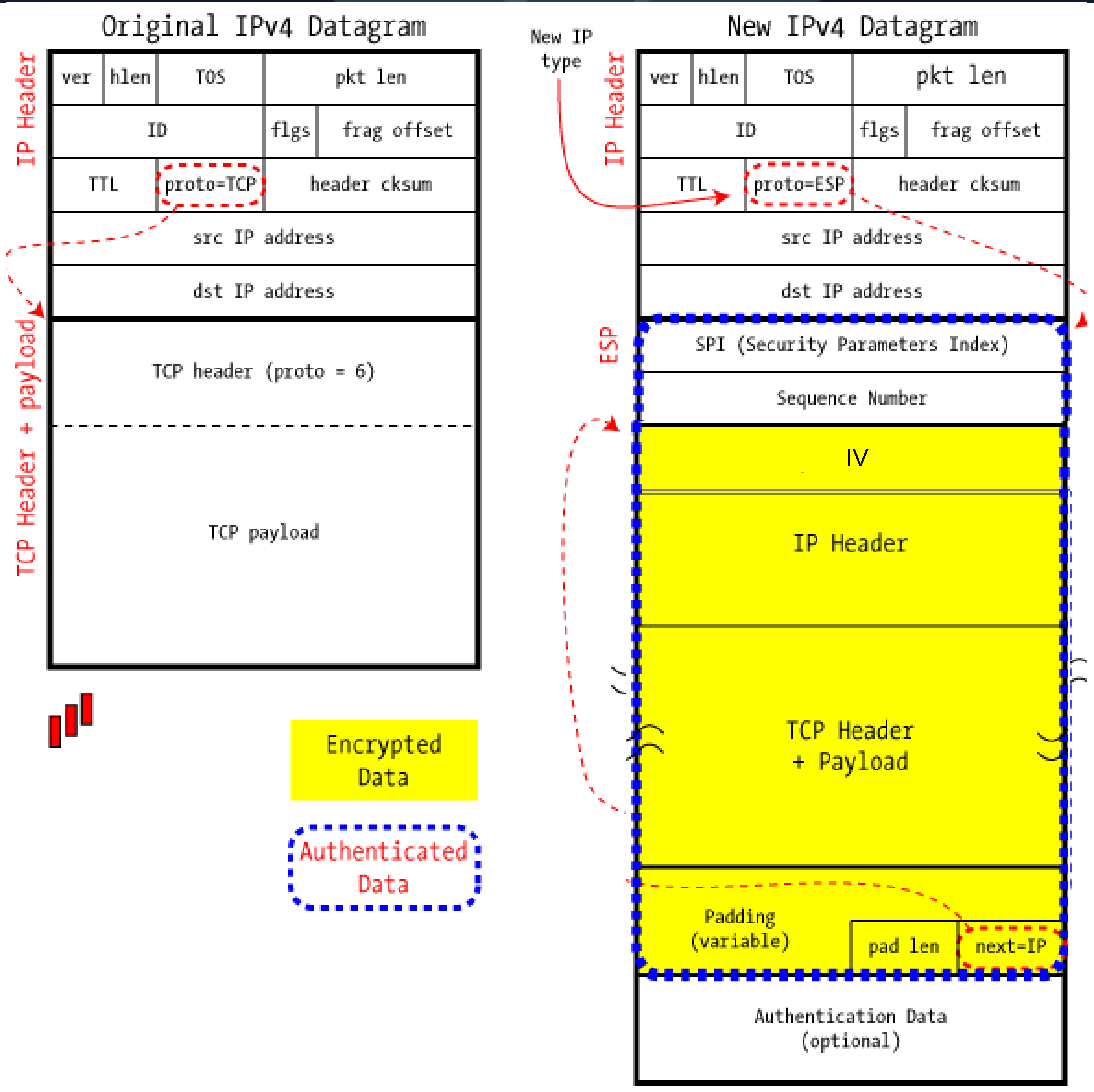
Model

- Software Responsibilities:
  - Handling packet headers
  - IPsec replay protection
  - IPsec policy checks

- NIC offloads crypto
  - Encrypt/Decrypt and authenticate packets as they go through the device
  - Receive authentication failure packets remain unchanged

- Software fallback
  - Hardware might not encrypt/decrypt some packet
  - All operations performed by hardware can be performed by software

ESP Tunnel Packets

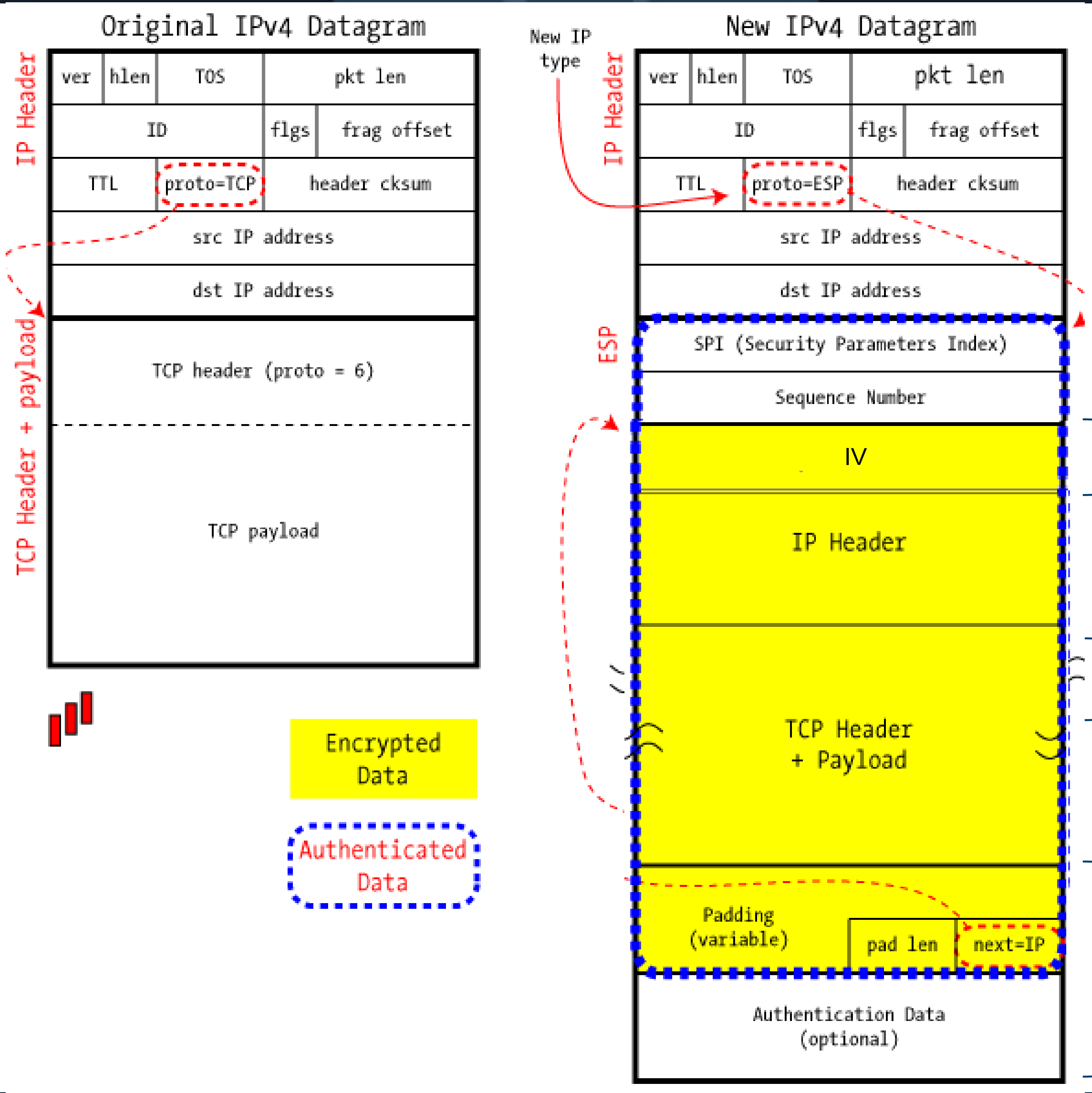


Challenges: LSO

- Without hardware crypto it is impossible to use LSO for IPsec packets

- **LSO requirements:**
- Checksum offload
  - See next slide
- Increment IPsec sequence number
  - IPsec sequence numbers need to be incremented in addition to TCP sequence numbers
- Generate the IPsec trailer for each packet
  - GSO packets can’t have a trailer for each mss
- Generate the correct IV for each packet
  - IV must be synced between software and hardware

ESP Tunnel Packets: LSO



Generate IV per GSO segment

Calculate checksum for each GSO segment

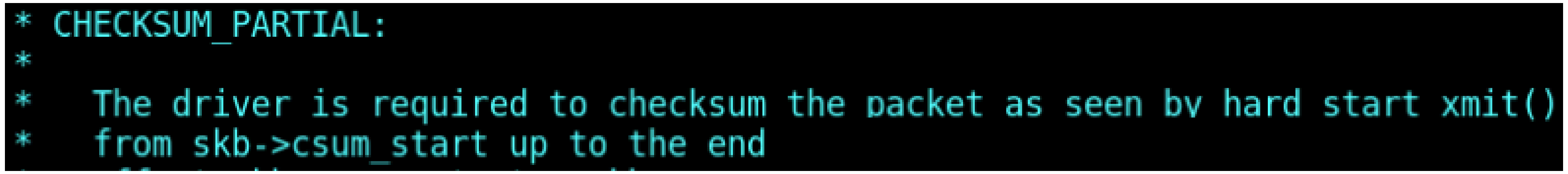
Generate ESP trailer per GSO segment

Challenges: Checksum offload

- Without hardware crypto offload it is impossible to use checksum offload for IPsec packets.
  - Checksum is computed before data encryption or after decryption

Transmit Checksum Offload:

- **Problem:** IPsec packets have a trailer, packets with a trailer don’t support CHECKSUM_PARTIAL. From include/linux/skbuff.h:



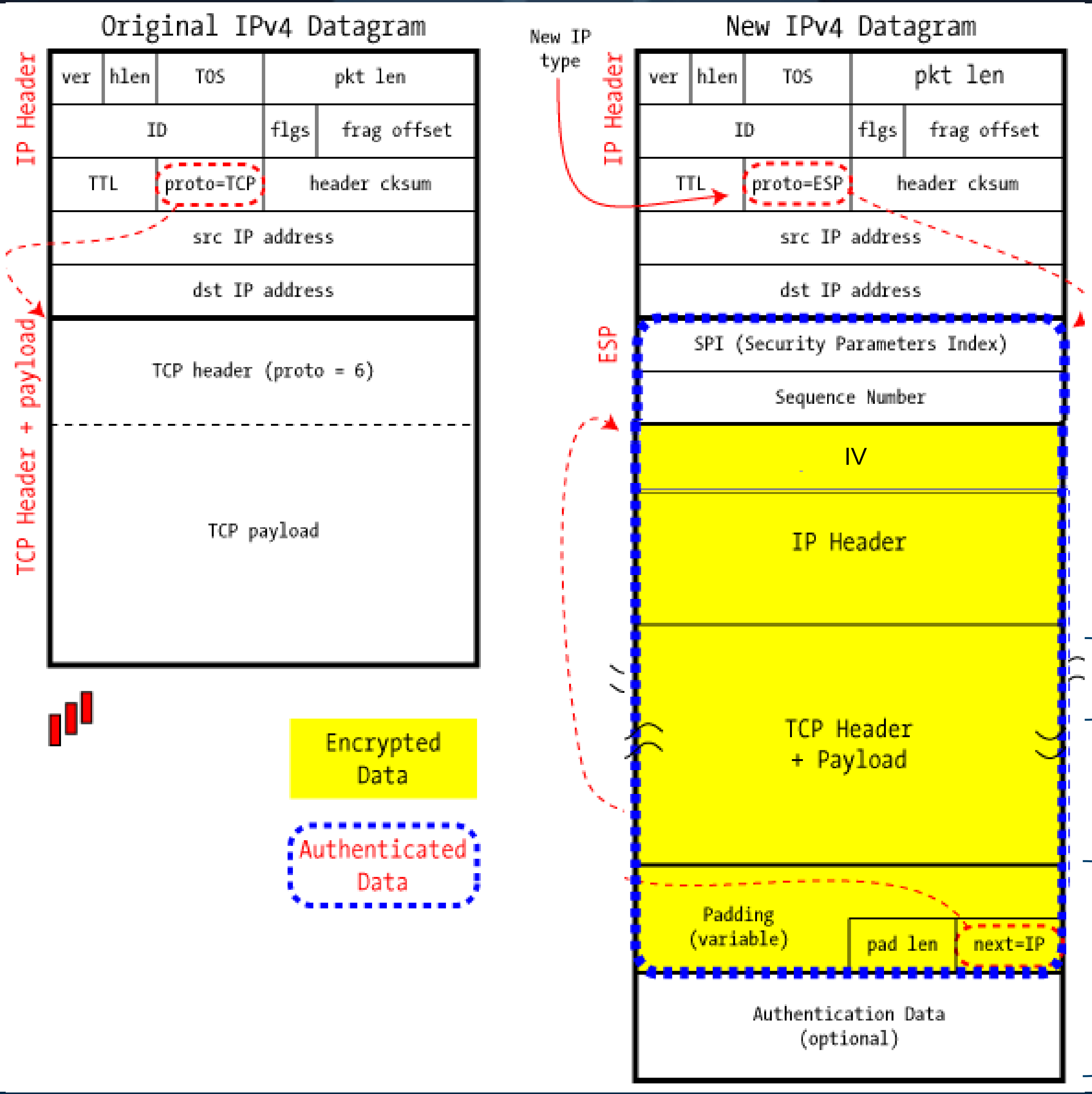
```
* CHECKSUM_PARTIAL:
*
*   The driver is required to checksum the packet as seen by hard_start_xmit()
*   from skb->csum_start up to the end
```

- **Soltuion1:** IPsec packets whose encryption is offloaded will be sent without a trailer. The trailer will be generated by hardware.
- **Soltuion2:** The driver will parse offloaded IPsec packets, calculate the length of the trailer and request hardware to calculate the checksum without the trailer.

Receive Checksum Offload:

- Need to add support for CHECKSUM_COMPLETE for ESP packets.

ESP Tunnel Packets: LSO



The checksum is calculated **before** encryption

Not included in TCP checksum

Challenges: IV processing (seqiv)

- **Reminder:** HW needs to generate IV for LSO

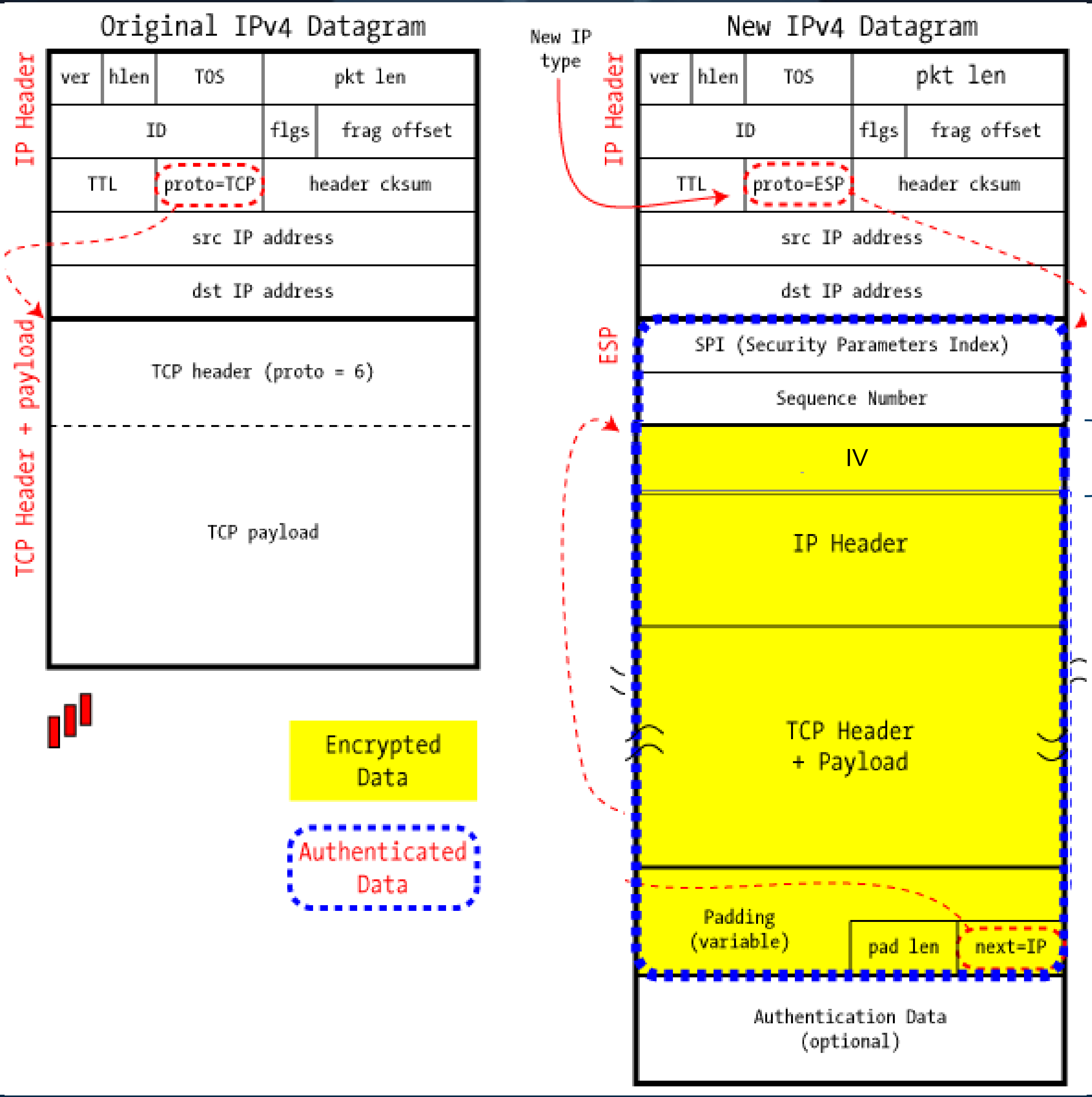
- According to RFC4106 (The use of GCM in ESP) the initialization vector of ESP packets for a given key MUST NOT repeat.

- However, it is unspecified how uniqueness is ensured

- In Linux, a the ESP sequence number is used to ensure uniqueness, but it is XORed with a nonce randomly generated at xfrm_state initialization.
  - Default for AES-GCM is crypto/seqiv.c

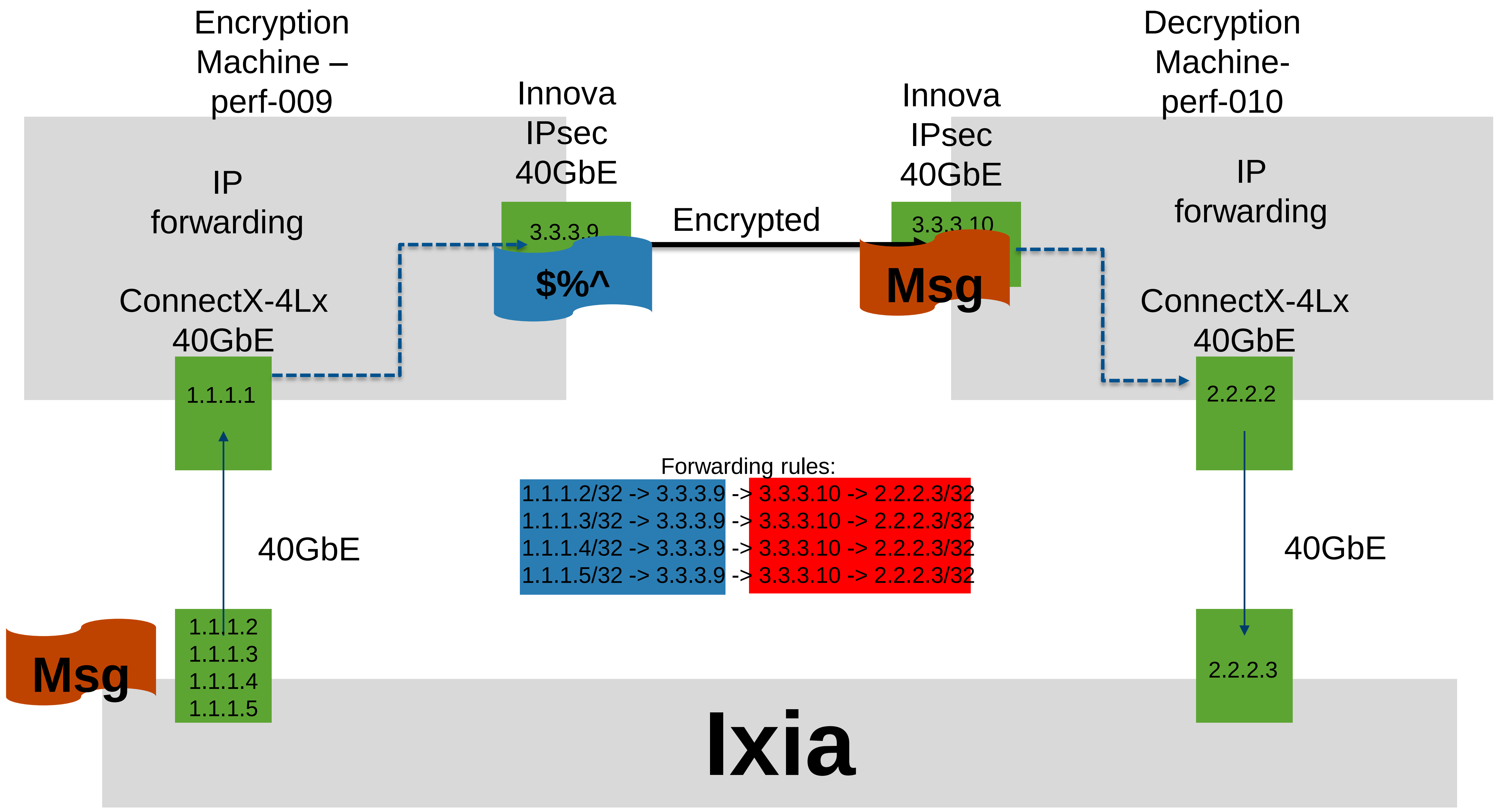
- **Problem:** Linux specific behavior needs to be implemented in hardware. Otherwise, the IV might be repeated.
- **Solution:** IV = ESP, then HW performs seqiv

ESP Tunnel Packets: LSO



XOR IV with nonce (seqiv)

Test System Description



Forwarding rules:

1.1.1.2/32 -> 3.3.3.9 -> 3.3.3.10 -> 2.2.2.3/32
1.1.1.3/32 -> 3.3.3.9 -> 3.3.3.10 -> 2.2.2.3/32
1.1.1.4/32 -> 3.3.3.9 -> 3.3.3.10 -> 2.2.2.3/32
1.1.1.5/32 -> 3.3.3.9 -> 3.3.3.10 -> 2.2.2.3/32

Performance Results (ESP-Tunnel IPv4)

Single stream direction	Metric	ESP no-offload	ESP offload
Tx	Throughput	4.5Gbps	25.5Gbps
Tx	CPU	100%	100%
Rx	Throughput	4.5Gbps	18.2Gbps
Rx	CPU	75%	100%

Current Status

HW & Driver:

- ESP4 tunnel mode
- AES-GCM
  - 128 or 256 bit keys
  - 8,12 or 16 ICV
- LSO
- Checksum offload
- IV processing
- Statistics
- Capabilities

IPsec Stack:

- ESP4 + ESP6
- GSO
- Checksum offload
- Expose capabilities

Userspace:

- iproute2
- strongswan

Limitations

- Cannot support IP fragments
- Offloaded packets must be routed to the offloading device
  - Software fallback when routed to wrong device?

Future

- ESP4 transport mode
- IPv6
- AES-CBC with HMAC-SHA1
- Extended Sequence Numbers
- Encapsulation support: IPsec over [VXLAN, Geneve, etc.]
- Offload replay protection
- RSS using inner headers

Implementation Details

New NDO

- New NDO called xfrmdev_ops

- int (*xdo_dev_state_add) (struct xfrm_state *x);
  - Attempt to offload xfrm_state to hardware – might fail due to:
    - Crypto unsupported
    - Protocol unsupported (AH, IP compression)
    - Encapsulation is not supported
- void (*xdo_dev_state_delete) (struct xfrm_state *x);
  - Stop offloading xfrm_state in hardware
- void (*xdo_dev_state_free) (struct xfrm_state *x);
  - Free hardware resources
- int (*xdo_dev_offload_ok) (struct sk_buff *skb, struct xfrm_state *x);
  - Is it possible to offload crypto for this sk_buff?

Receive Flow

- Hardware identifies offloaded IPsec packet according to [dst IP, SPI, ip protocol]
- Decrypt and authenticate packet in hardware
  - completion contains metadata regarding xfrm_state used and crypto operation result
- Populate skb->sp->ovec and skb->sp->xvec in driver
  - New member of struct sec_path contains crypto offload information
- xfrm_input skips decryption, authentication and xfrm_state_lookup
- Process headers according to CHECKSUM_COMPLETE

- **Note:** Raw sockets (tcpdump) see plaintext ESP packets

Transmit Flow

- **xfrm_output:**
  - xfrm_offload_ok(skb, x)
    - Was xfrm_state offloaded?
    - Can we offload this skb?
  - For offload packets:
    - Set skb->sp (SKB_CRYPTO_OFFLOAD)
    - Set skb->encapsulation
    - Skip checksum

- **xfrm_output_one**:
  - GSO ESP packets need ESP header but no trailer
  - New replay protection for GSO

- **Note:** Raw sockets (tcpdump) see plaintext ESP packets

- **Network Device:**
  - Offload crypto according to skb->sp
  - LSO and checksum offload leverage skb->inner_*
  - Remove ESP trailer (if needed)

Thank You

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™