rte_security: enabling hardware acceleration of security protocols

Boris Pismenny (Mellanox)
Declan Doherty (Intel)
Hemant Agrawal (NXP)

DPDK Summit Userspace - Dublin- 2017

Agenda

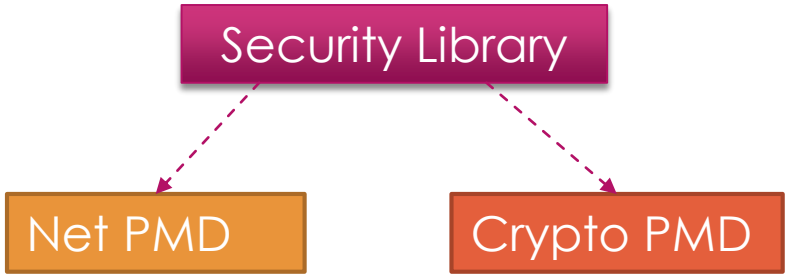
- Introduction
- Acceleration enablement modes
  - Inline Crypto
  - Lookaside Security Protocol
  - Inline Security Protocol
- Security Library Details
- Summary & Future Work
- Discussion and Q&A

Introduction

Introduction and IPsec basics

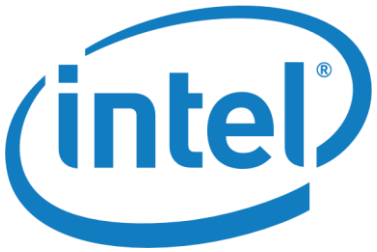
Introduction

- Framework for management and provisioning of hardware acceleration of security protocols.
- Generic APIs to manage security sessions.
- Security acceleration functions are accessed through security instances which can instantiated on any device type, current supports security instances on Crypto and Ethernet devices.
- Rich capabilities discovery APIs
- Current only targets the support of IP Security (IPsec) protocol.
- Could support a wide variety of protocols/applications
  - Enterprise/SMB VPNs — IPsec
  - Wireless backhaul — IPsec, PDCP
  - Data-center — SSL
  - WLAN backhaul — CAPWAP/DTLS
  - Control-plane options for above — PKCS, RNG



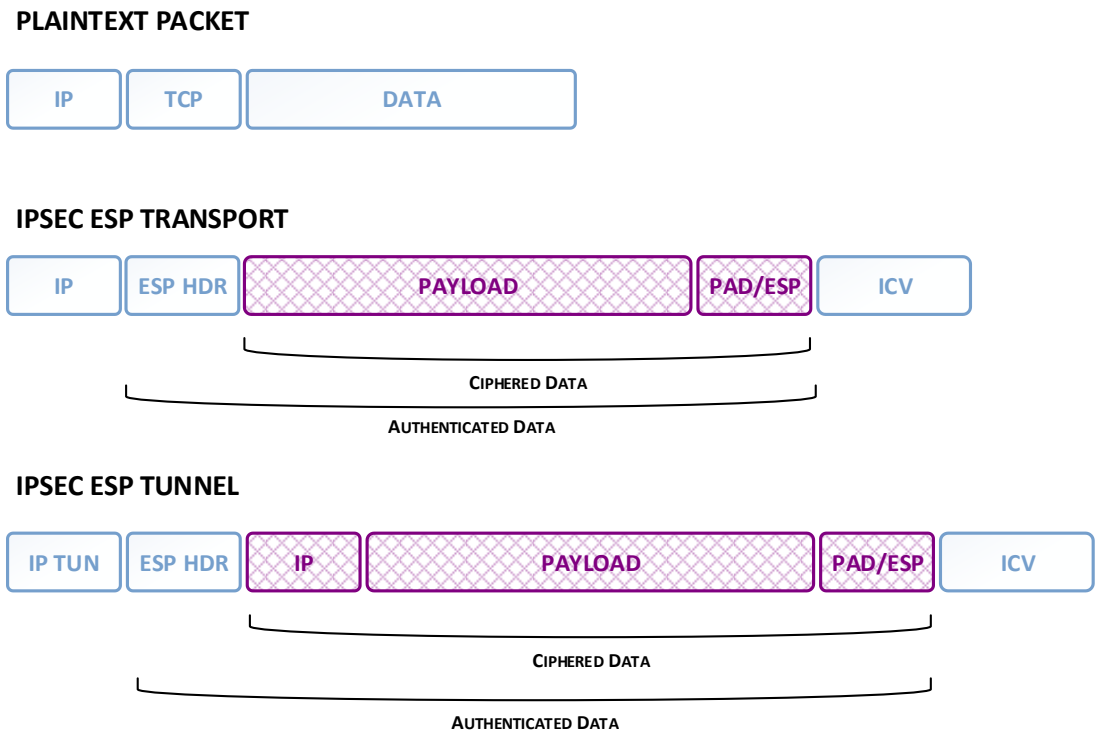
Community Collaboration

- Collaborative work between Intel, Mellanox and NXP with contributions from:
  - Hemant Agrawal, Declan Doherty, Akhil Goyal, Radu Nicolau, Boris Pismenny, and Aviad Yehezkel.
- Security library approach evolved out of the following RFC's
  - [1] http://dpdk.org/ml/archives/dev/2017-July/070793.html
  - [2] http://dpdk.org/ml/archives/dev/2017-July/071893.html
  - [3] http://dpdk.org/ml/archives/dev/2017-August/072900.html
- The joint proposal has been developed on dpdk.org dpdk-draft-ipsec repo.



IPsec Protocol Basics

- IPsec is a layer 3 IP security service.
- Security services offered by IPsec includes:
  - Connectionless integrity
  - Data confidentiality (encryption)
  - Sequence Integrity (partial, anti-replay windowing)
  - limited traffic flow confidentiality (tunnel mode).
- These security services are provided by the use of two traffic security protocols.
  - Authentication Header (AH)
  - Encapsulating Security Payload (ESP)
- IPsec is designed to be crypto algorithm independent



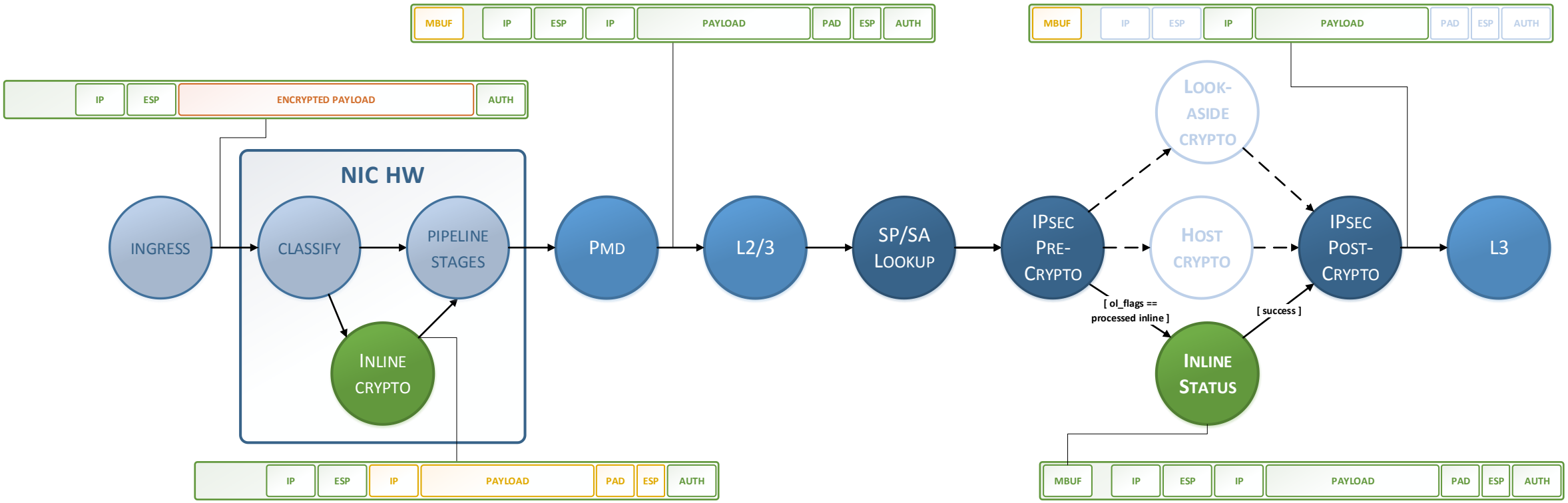
IPsec Hardware Acceleration Modes

Inline Crypto, Lookaside Protocol and Inline Protocol

Inline Crypto Acceleration

- IO based acceleration performed on the physical interface as packet ingress/egress the system.
- No packet headers modifications* on the hardware, only encryption/decryption and authentication operations are preformed.
  - Hardware may support extra features like payload padding of etc.
- Requires that Ethernet CRC is also offloaded to the physical interface also.

Inline Crypto Ingress Data Path

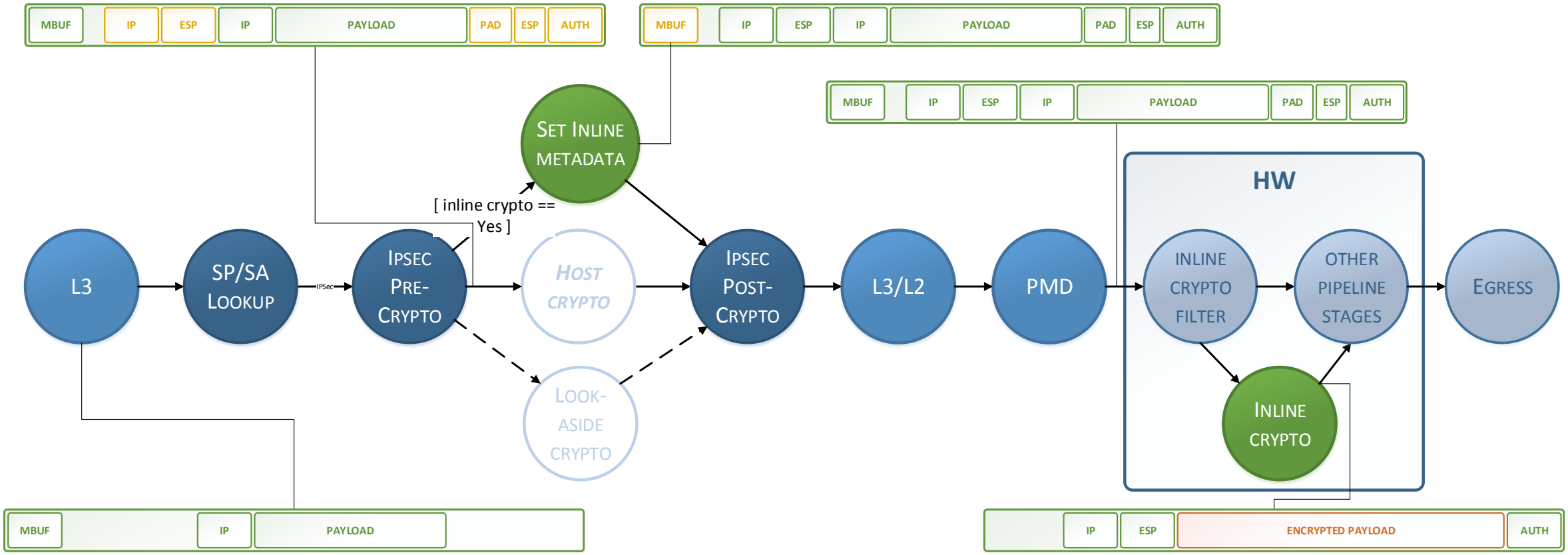


Inline Crypto Ingress Data Path

- HW performs the following for matching (SIP, DIP, ESP)* packets:
  - Decryption and authentication processing – mark the result in metadata
  - Remove the ESP trailer*
- PMD provides the following info per packet:
  - Crypto result – success/failure.
  - Inner ESP next protocol*
  - Packet without a trailer*
- Application:
  - Check mbuf->ol_flags for PKT_RX_SEC_OFFLOAD / PKT_RX_SEC_OFFLOAD_FAILED
  - Read the inner ESP next protocol to remove the ESP header

* Support is hardware dependent, there may be some variation in the ex

Inline Crypto Egress Data Path



Inline Crypto Egress Data Path

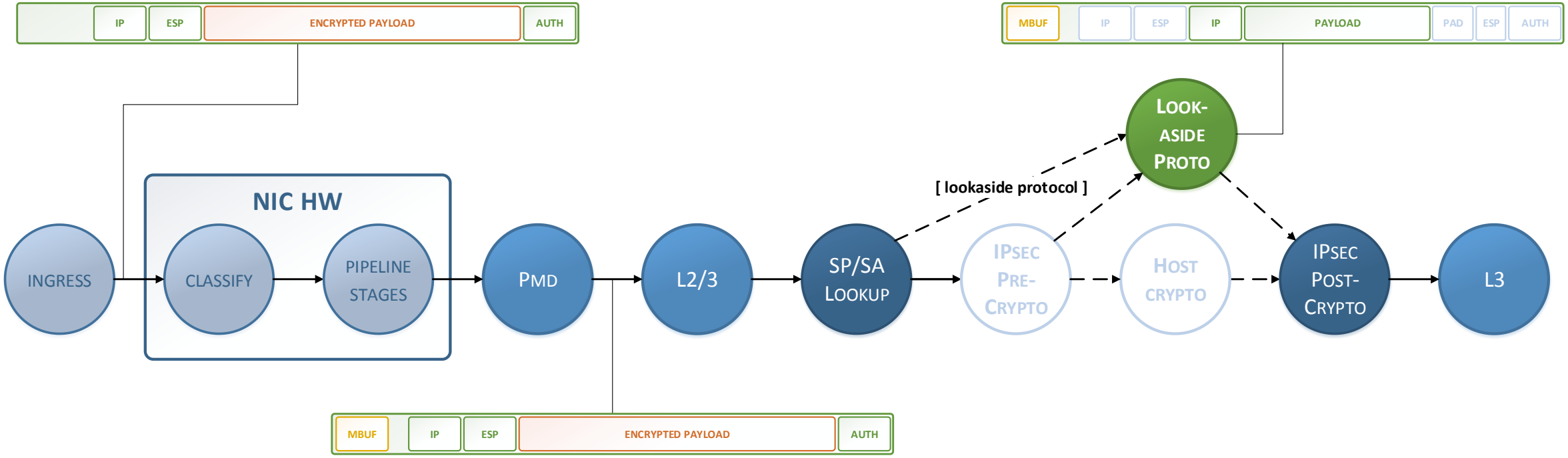
- Application:
  - Mark mbuf->ol_flags using PKT_TX_SEC_OFFLOAD
  - Marks packet with IPsec as ESP tunnel
  - Set all security metadata in mbuf as needed, including inner_esp_next_proto according to inner packet*
- PMD can preform special processing on packets with the PKT_TX_SEC_OFFLOAD flag set, including:
  - Inner ESP next protocol*
  - Security Association Index for hardware*
  - More information for LSO support*
- HW performs the following for marked packets:
  - Add the ESP trailer if supported
  - Encryption and authentication

*Exact requirements, if any, are hardware dependent

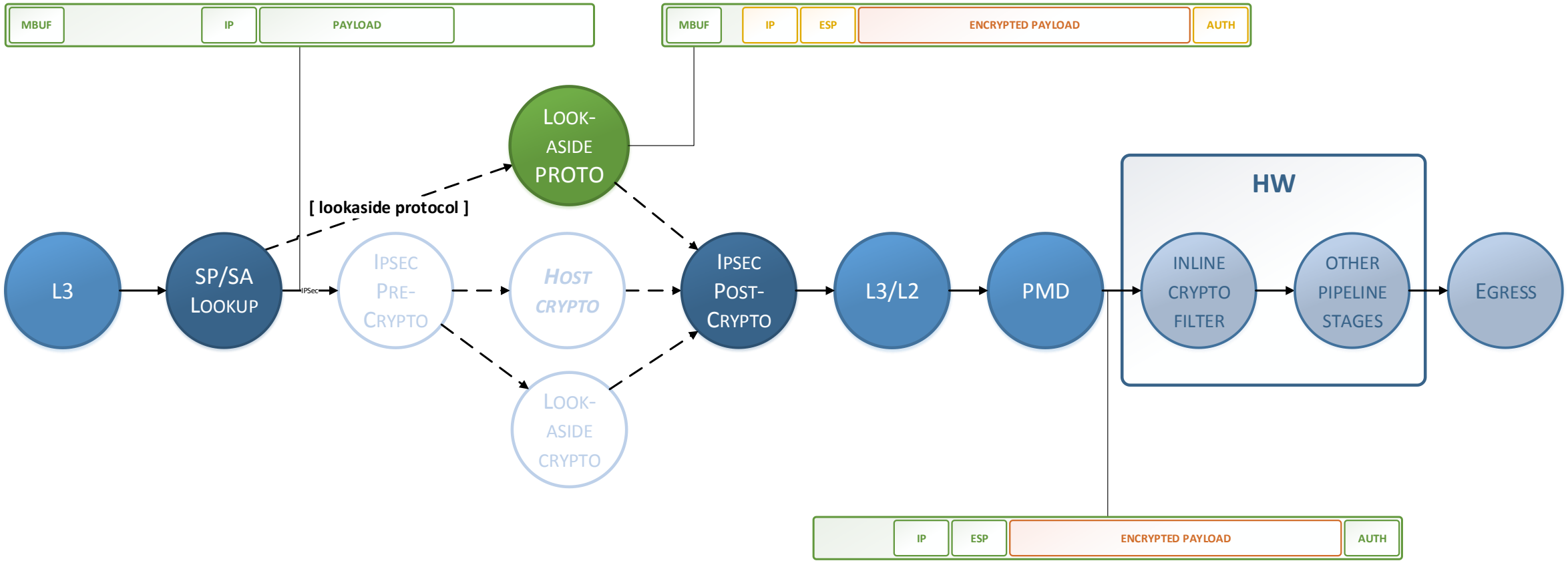
Lookaside Protocol Acceleration

- Lookaside acceleration model where packet is given to an accelerator for processing and then returned to the host after processing is complete.
- Security function is provided as an extension of a librte_cryptodev crypto PMD.
  - Session is used to configure the IPsec SA material on the accelerator.
  - Security session is used in place of crypto session in crypto op when enqueuing and dequeuing packets to the crypto PMD.
- Supports full protocol (IPsec) processing on the accelerator. Including:
  - Add/remove protocol headers, including IP tunnel headers as well as IPsec (AH/ESP) headers.
  - Handling SA state information:
    - Sequence numbers
    - Anti-replay window

Lookaside Protocol Acceleration (Ingress)



Lookaside Protocol Acceleration (Egress)



Inline Protocol Acceleration**

- IO based acceleration performed on the physical interface as the packet ingresses/egresses the platform.
- Interface performs all crypto processing for the security protocol (e.g. IPsec) during transmission and reception.
- Packet headers modification is performed on hardware including all state management and encryption/decryption and authentication operations.
  - Hardware may support extra features like padding of payload etc.
- Requires that ARP entries for MAC headers are programmed along with the security action, as host may not know destination IP in case of a tunnel mode SA

** Currently no supported implementation, so implementation will be subject to change.

Library Implementation

Core features of the librte_security

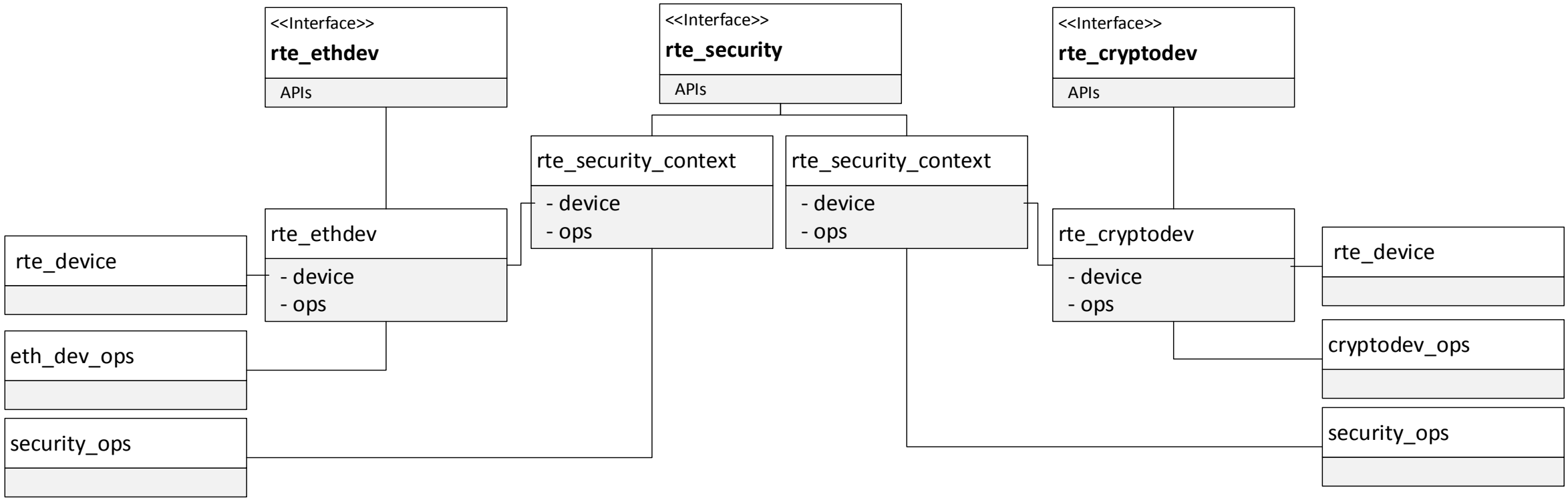
Library Features

- Security instance management and abstraction from base device type.
- Protocol agnostic session API for the management of protocol state on underlying hardware.
- Definitions of supported protocols, currently only IPsec, and the parameters for configuring the options. For IPsec this includes:
  - Acceleration type – inline crypto/lookaside protocol/inline protocol
  - Defining security association (SA) parameters such as Tunnel/Transport, ESP/AH, Ingress/Egress as well as associated crypto processing and key material
- Crypto operations are defined using primitives defined in librte_cryptodev limit any redefinition of parameters within DPDK.
- Capabilities APIs to allow dynamic discovery of a instances features.

Security Instance Management

- The library is not specifically associated with a specific device instance in DPDK
- Any driver can register itself as security capably using the rte_security_register() API.
- The library maintains an array of active instances, which define the supported rte_security_ops and a void pointer to the supporting device.
- API can be supported by multiple device types or possibly even as a stand-alone device.

A multi-device API (Object Model)



Protocols and actions

- Select the session Protocol: “rte_security_session_protocol”
  - IPSEC, MACSEC, SSL, PDCP etc.
- Select the Security Action Type: “rte_security_session_action_type”
  - `RTE_SECURITY_ACTION_TYPE_INLINE_CRYPTO: Inline crypto processing as NIC offload during recv/transmit.`
  - `RTE_SECURITY_ACTION_TYPE_INLINE_PROTOCOL: Inline security protocol processing as NIC offload during recv/transmit.`
  - `RTE_SECURITY_ACTION_TYPE_LOOKASIDE_PROTOCOL: Security protocol processing including crypto on a crypto accelerator.`
- Action type can be a input for the given application during SA creation
- Based on the action type and other SA related information, application configures session parameters for security offload.

Security Session Management

- Session APIs support
  - Create Session

struct rte_security_session *
rte_security_session_create(uint16_t id,

struct **rte_security_session_conf** *conf,

struct **rte_mempool** *mp);

  - Update
  - Destroy
  - Query (Get Stats)

```
struct rte_security_session_conf {
        enum rte_security_session_action_type action_type;
        /**< Type of action to be performed on the session */

        enum rte_security_session_protocol protocol;
        /**< Security protocol to be configured */
        union {
                struct rte_security_ipsec_xform ipsec;
         };
        /**< Configuration parameters for security session */

        struct rte_crypto_sym_xform *crypto_xform;
        /**< Security Session Crypto Transformations */
}
```

rte_security_ipsec_xform

- The current set support some of the basic feature set of IPSEC – More to be added incrementally.
- Many parameters are applicable for Full Protocol Offload only

```
struct rte_security_ipsec_xform {
        uint32_t spi;           /**< SA security parameter index  */
        uint32_t salt;          /**< Salt */
        struct rte_security_ipsec_sa_options options;
        enum rte_security_ipsec_sa_direction dir;
        /**< SA Direction Egress/Ingress */
        enum rte_security_ipsec_sa_protocol proto;
        /**< SA Protocol AH/ESP  */
        enum rte_security_ipsec_sa_mode mode;
        /**< SA Mode Transport/Tunnel */
        struct rte_security_ipsec_tunnel_param tunnel;
        /**<
         * SA Tunnel Parameter. Only applicable when SA is tunnel mode
         * and offload type is either inline/lookaside protocol
         */
};
```

IPsec SA options include

- Extended Sequence Number Support
- NAT/UDP encapsulation,
- NAT-T L4 Checksum verify/update
- Qos/TOS copy support (inner to outer - tunnel)
- DF copy support (inner to outer - tunnel)
- TTL decrement support
- Address copy support (inner to outer for tunnel)
- Trailer addition support

Capabilities Example

- Capabilities API allow user to get all the capabilities of the devices or to query a single capability

- List of Capabilities

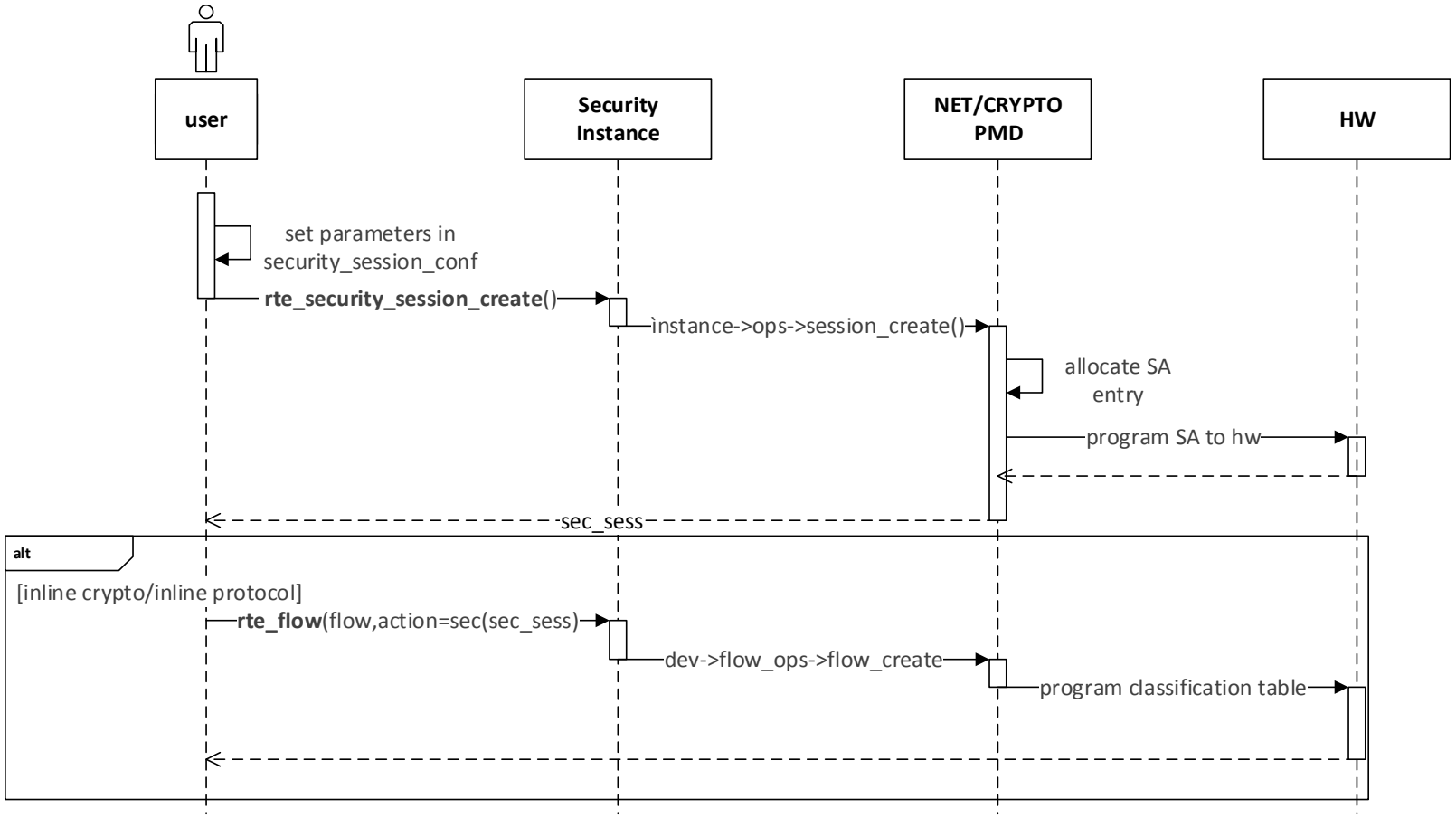
```
const struct rte_security_capability *
        rte_security_capabilities_get(uint16_t id);
```

- Check on Specific Capability

```
const struct rte_security_capability *
rte_security_capability_get(uint16_t id,
        struct rte_security_capability_idx *idx);
```

```
{  /* IPsec Inline Crypto Ingress ESP Transport  */
        .action = RTE_SECURITY_ACTION_TYPE_INLINE_CRYPTO,
        .protocol = RTE_SECURITY_PROTOCOL_IPSEC,
        .ipsec = {
                .proto = RTE_SECURITY_IPSEC_SA_PROTO_ESP,
                .mode = RTE_SECURITY_IPSEC_SA_MODE_TRANSPORT,
                .direction = RTE_SECURITY_IPSEC_SA_DIR_INGRESS
        },
        .crypto_capabilities = { /* AES-GCM (128-bit) */
                .op = RTE_CRYPTO_OP_TYPE_SYMMETRIC,
                .sym = {
                        .xform_type = RTE_CRYPTO_SYM_XFORM_AEAD,
                        .aead = {
                                .algo = RTE_CRYPTO_AEAD_AES_GCM,
                                .block_size = 16,
                                .key_size =    { .min = 16, .max = 16, .increment = 0 },
                                .digest_size =         { .min = 8, .max = 16, .increment = 4 },
                                .aad_size =   { .min = 4, .max = 8, .increment = 4 },
                                .iv_size =      { .min = 12, .max = 12, .increment = 0 }
                        },
                },
        },
},
```

Control Path




```
/** security session configuration parameters */
struct rte_security_session_conf sess_conf = {
        .action_type = RTE_SECURITY_ACTION_TYPE_INLINE_CRYPTO,
        .protocol = RTE_SECURITY_PROTOCOL_IPSEC,
        .ipsec = {
                .spi = /** Security Protocol Index */,
                .salt = /** Salt value */,
                .direction = RTE_SECURITY_IPSEC_SA_DIR_INGRESS,
                .proto = RTE_SECURITY_IPSEC_SA_PROTO_ESP,
                .mode = RTE_SECURITY_IPSEC_SA_MODE_TUNNEL
        },
        .crypto_xform = /** crypto transforms*/
};
```

```
/** flow parameters */
attr->ingress = 1;

pattern[0].type = RTE_FLOW_ITEM_TYPE_ETH;
pattern[1].type = RTE_FLOW_ITEM_TYPE_IPV4;
pattern[2].type = RTE_FLOW_ITEM_TYPE_ESP;
pattern[3].type = RTE_FLOW_ITEM_TYPE_END;

action[0].type = RTE_FLOW_ACTION_TYPE_SECURITY;
action[0].conf = /** security session */;
action[1].type = RTE_FLOW_ACTION_TYPE_END;
```

rte_flow Implementation

ethdev control path for rte_security

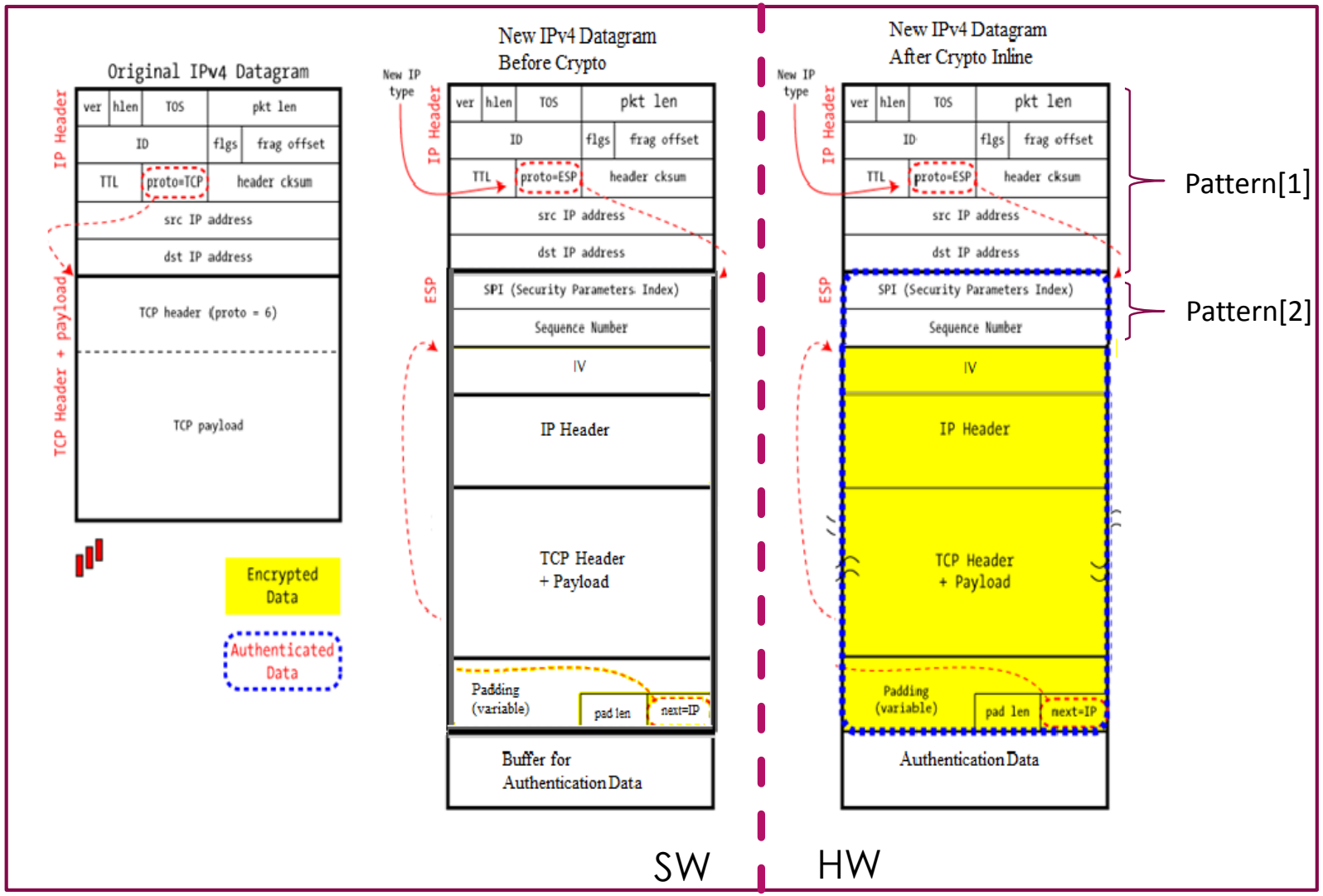
Inline Crypto Offload

```
/** security session configuration parameters */
struct rte_security_session_conf sess_conf = {
        .action_type =
RTE_SECURITY_ACTION_TYPE_INLINE_CRYPTO,
        .protocol = RTE_SECURITY_PROTOCOL_IPSEC,
        .ipsec = {
                .spi = /**< Security Protocol Index */,
                .salt = /** Salt value */,
                .direction = RTE_SECURITY_IPSEC_SA_DIR_INGRESS,
                .proto = RTE_SECURITY_IPSEC_SA_PROTO_ESP,
                .mode = RTE_SECURITY_IPSEC_SA_MODE_TUNNEL
        },
        .crypto_xform = /** crypto transforms*/
};
```

```
/** flow parameters */
attr->ingress = 1; /** attr->egress = 1 */

pattern[0].type = RTE_FLOW_ITEM_TYPE_ETH;
pattern[1].type = RTE_FLOW_ITEM_TYPE_IPV4;
pattern[2].type = RTE_FLOW_ITEM_TYPE_ESP;
pattern[3].type = RTE_FLOW_ITEM_TYPE_END;

action[0].type = RTE_FLOW_ACTION_TYPE_SECURITY;
action[0].conf = sa->sec_session;
action[1].type = RTE_FLOW_ACTION_TYPE_END;
```



Inline Crypto Offload - Example 1

rte_flow allows for extensible inline crypto support, without redefining network headers in crypto library:

For example:

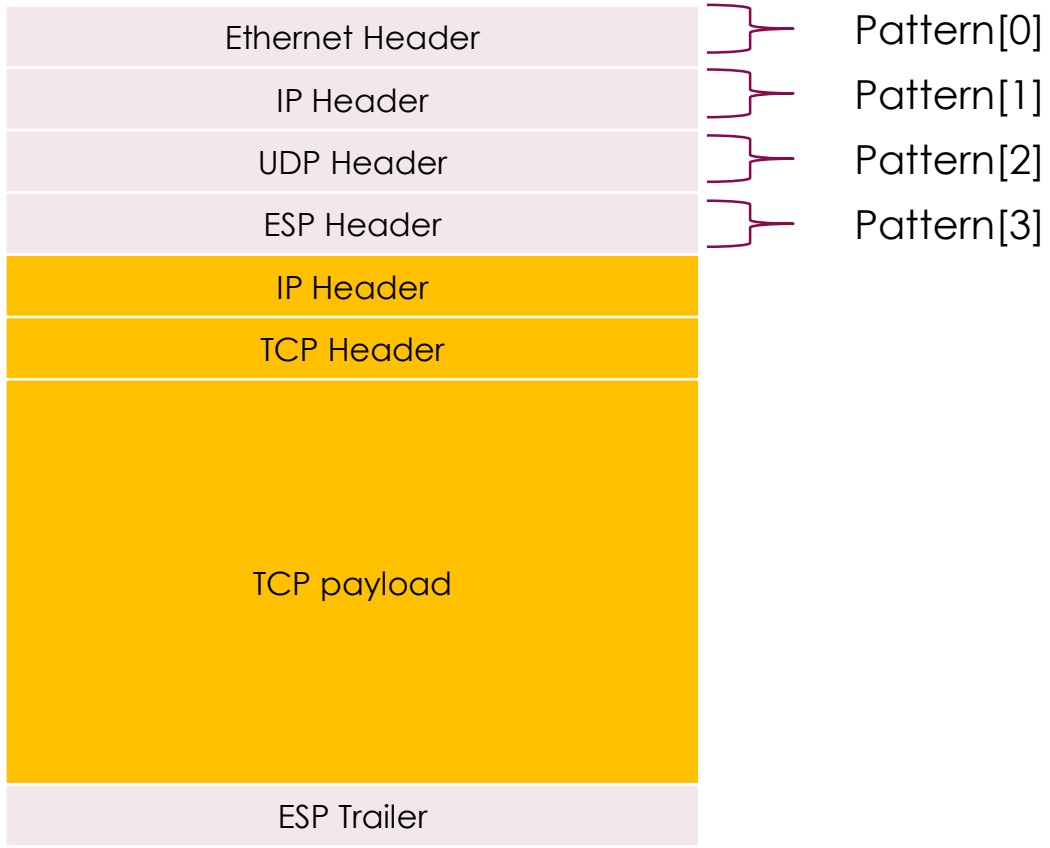
- UDP Encap using rte_flow

```
/** flow parameters */
attr->ingress = 1; /** attr->egress = 1 */

pattern[0].type = RTE_FLOW_ITEM_TYPE_ETH;
pattern[1].type = RTE_FLOW_ITEM_TYPE_IPV4;
pattern[2].type = RTE_FLOW_ITEM_TYPE_UDP;
pattern[3].type = RTE_FLOW_ITEM_TYPE_ESP;
pattern[4].type = RTE_FLOW_ITEM_TYPE_END;

action[0].type = RTE_FLOW_ACTION_TYPE_SECURITY;
action[0].conf = sa->sec_session;
action[1].type = RTE_FLOW_ACTION_TYPE_END;
```

UDP ESP example:



Inline Crypto Offload - Example 2

rte_flow allows for extensible inline crypto support, without redefining network headers in crypto library:

For example:

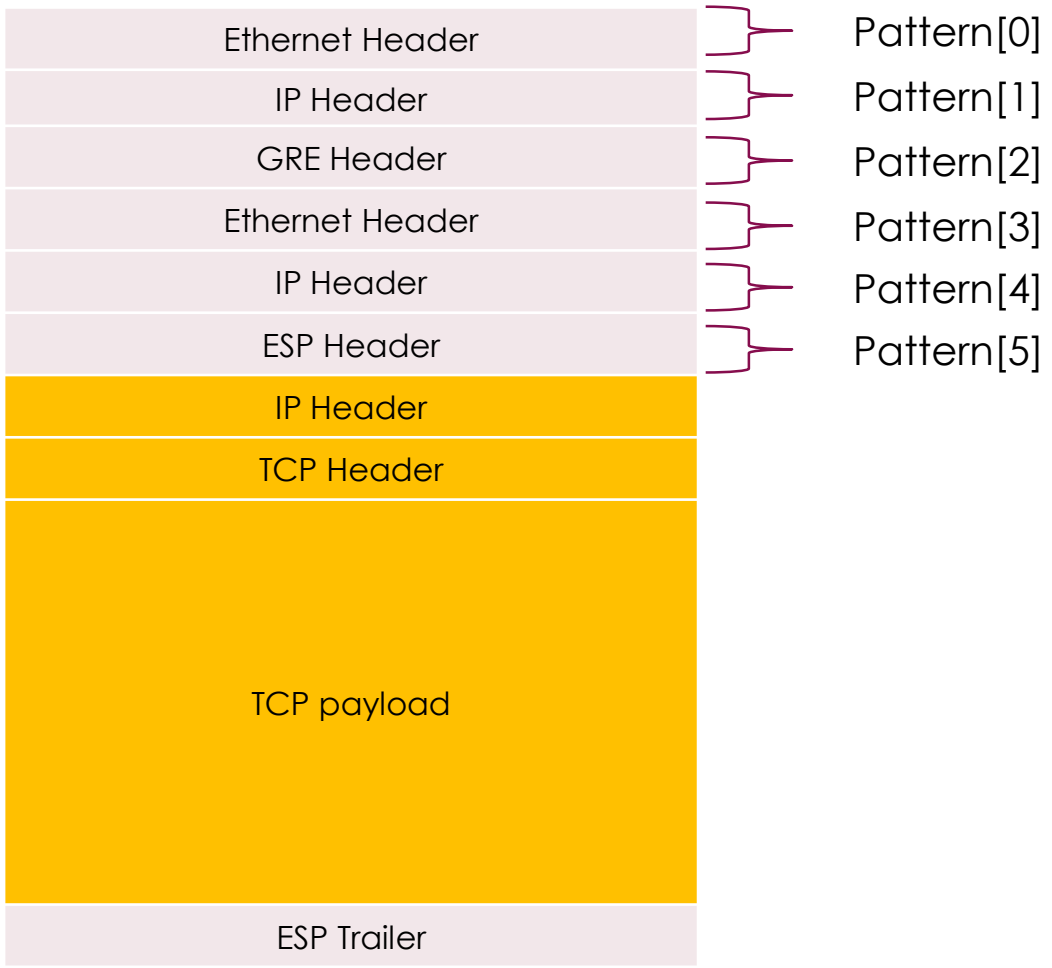
- NVGRE | ESP | TCP using rte_flow

```
/** flow parameters */
attr->ingress = 1; /** attr->egress = 1 */

pattern[0].type = RTE_FLOW_ITEM_TYPE_ETH;
pattern[1].type = RTE_FLOW_ITEM_TYPE_IPV4;
pattern[2].type = RTE_FLOW_ITEM_TYPE_NVGRE;
pattern[3].type = RTE_FLOW_ITEM_TYPE_ETH;
pattern[4].type = RTE_FLOW_ITEM_TYPE_IPV4;
pattern[5].type = RTE_FLOW_ITEM_TYPE_ESP;
pattern[6].type = RTE_FLOW_ITEM_TYPE_END;

action[0].type = RTE_FLOW_ACTION_TYPE_SECURITY;
action[0].conf = sa->sec_session;
action[1].type = RTE_FLOW_ACTION_TYPE_END;
```

NVGRE | ESP | TCP example:



Discussion

Inline Protocol Offload – Control Path

- INLINE_PROTOCOL might be the first encap/decap action in rte_flow.
- How do we describe these with rte_flow?
- Do we need to define an order between actions and patterns?
  - Linking actions to patterns
- How to express the ARP entries for destination IPs in case of encap/decap.

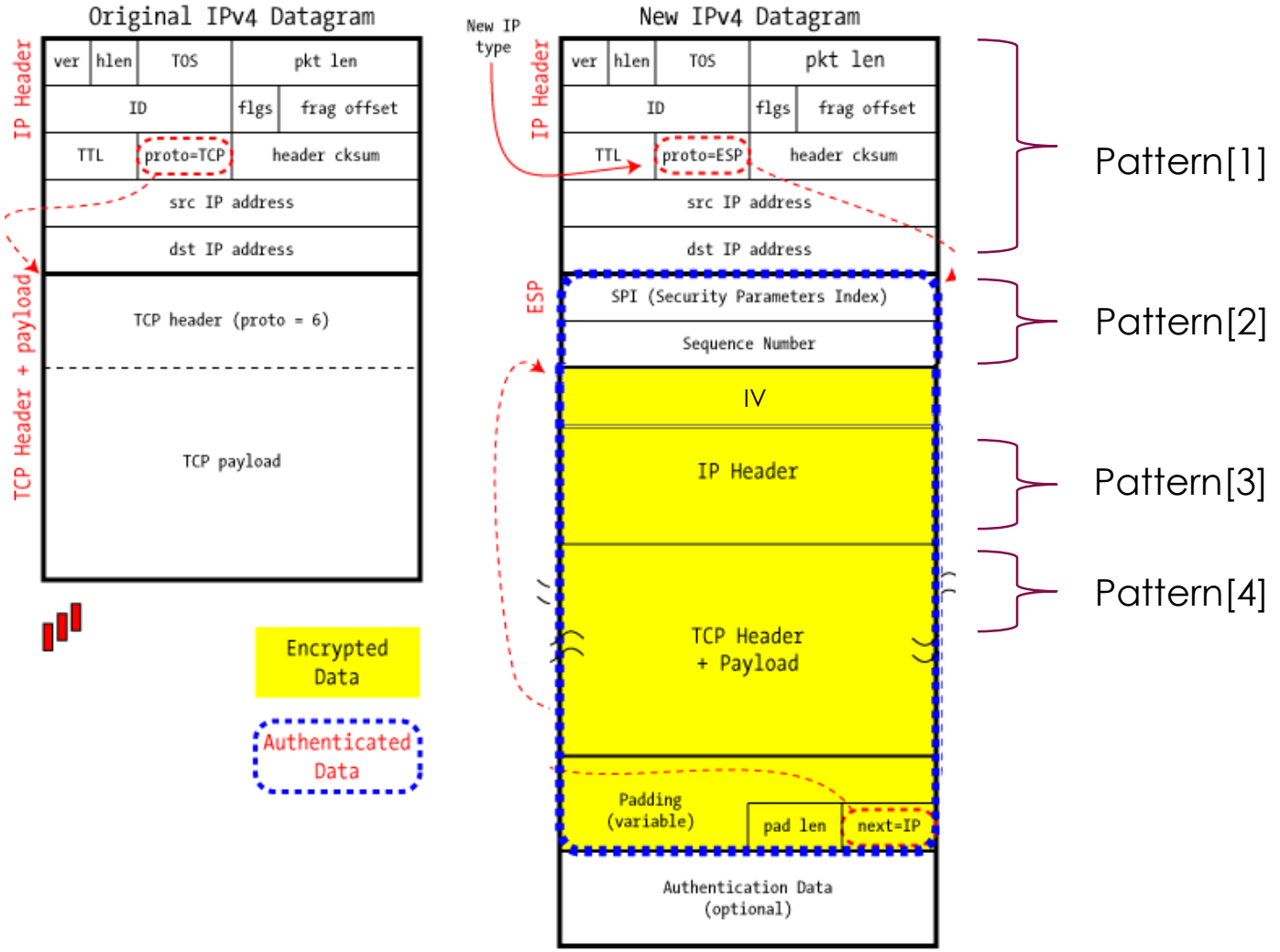
Inline Protocol Offload – Control Path

- Actions referring to patterns

```
/** security session configuration parameters */
struct rte_security_session_conf sess_conf = {
        .action_type = RTE_SECURITY_ACTION_TYPE_INLINE_PROTOCOL,
        .protocol = RTE_SECURITY_PROTOCOL_IPSEC,
        .ipsec = {
                .spi = /**< Security Protocol Index */,
                .salt = /** Salt value */,
                .direction = RTE_SECURITY_IPSEC_SA_DIR_INGRESS,
                .proto = RTE_SECURITY_IPSEC_SA_PROTO_ESP,
                .mode = RTE_SECURITY_IPSEC_SA_MODE_TUNNEL
                .tunnel = /** Tunnel parameters */,
        },
        .crypto_xform = /** crypto transforms*/
};
```

```
/** flow parameters */
pattern[0].type = RTE_FLOW_ITEM_TYPE_ETH; /** encap/decap */
pattern[1].type = RTE_FLOW_ITEM_TYPE_IPV4; /** encap/decap */
pattern[2].type = RTE_FLOW_ITEM_TYPE_ESP; /** encap/decap */
pattern[3].type = RTE_FLOW_ITEM_TYPE_IPV4; /** inner IP */
pattern[4].type = RTE_FLOW_ITEM_TYPE_TCP; /** inner TCP */
pattern[5].type = RTE_FLOW_ITEM_TYPE_END;

action[0].type = RTE_FLOW_ACTION_TYPE_SECURITY;
action[0].conf = sa->sec_session;
action[1].type = RTE_FLOW_ACTION_TYPE_END;
```



Inline Protocol Offload – Data Path

- **Idea:** Offload could be transparent to the data-path after configuration is complete
- Do we need to use any mbuf flags?
- Do we need to provide any metadata with mbufs? SA identifier? Crypto result?
- How are we to handle IP fragments?
- Error handling – failed to decap – Bad IP header, ESP trailer, etc.

Summary

Summary

- rte_security is a representation of a security session
- rte_security can be used with ethdev and cryptodev
- rte_security + rte_flow = powerful control plane
- Redundancy to ease the application migration between security offloads.

Future Work

- Further IPsec enablement
  - Further encapsulations
  - LSO + checksum
  - IPsec inline protocol offload
- Further protocol enablement
  - MACsec, PDCP, DTLS, etc would fit under this model.
- Software equivalent enablement
  - It could be possible to offer software equivalent processing under this API, may or may not be desirable depending on protocol and it’s processing overhead.

Questions?

Boris Pismenny (Mellanox)

Declan Doherty (Intel)

Hemant Agrawal (NXP)