# Recent advances with DPDK IPsec

DECLAN DOHERTY
FAN ROY ZHANG
KONSTANTIN ANANYEV
VLADIMIR MEDVEDKIN
INTEL

# Legal disclaimer

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Configurations: Estimates are based on internal Intel analysis using at least Data Plane Development Kit IpSec sample application on Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz

Performance results are based on testing as of 12/09/2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product or component can be absolutely Secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Other names and brands may be claimed as the property of others.



Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks Test and System Configurations

No computer system can be totally secure

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

# librte_ipsec: current status

- Introduced in 19.02
- Works with all different flavours of DPDK crypto devices (rte_cryptodev, rte_security)
- Supported features:
  - ESP protocol tunnel mode both IPv4/IPv6
  - ESP protocol transport mode both IPv4/IPv6
  - ESN and replay window
  - Supported algorithms:

    3DES-CBC, AES-CBC, AES-CTR, AES-GCM, HMAC-SHA1, NULL
  - Multi-segment packets
    - Support for fragment/reassemble packets in examples/ipsec-secgw

# 19.11 proposed features

- Security Association Database (SAD)

  http://patches.dpdk.org/cover/58536/
- New synchronous rte_security API for SW based crypto devices

  http://patches.dpdk.org/cover/58862/
- One SA over multiple crypto devices
  - Fall-back sessions for inline-crypto processing

    http://patches.dpdk.org/cover/58567/

# Security Association Database (SAD)

- Requirements
  - Scale up-to several millions of sessions
  - 2. Support fast lookup rate
  - 3. Support incremental updates
  - 4. RFC compliant

*RFC 4301 4.4.2.  Security Association Database (SAD):*
*In each IPsec implementation, there is a nominal Security Association Database (SAD), in which each entry defines the parameters associated with one SA.  Each SA has an entry in the SAD.*

*RFC 4301 4.1 Definition and Scope:*
*Each entry in the SA Database (SAD) (Section 4.4.2) must indicate whether the SA lookup makes use of the destination IP address, or the destination and source IP addresses, in addition to the SPI. For each inbound, IPsec-protected packet, an implementation must conduct its search of the SAD such that it finds the entry that matches the "longest" SA identifier.  In this context, if two or more SAD entries match based on the SPI value, then the entry that also matches based on destination address, or destination and source address (as indicated in the SAD entry) is the "longest" match.*

# SAD: API

```
struct rte_ipsec_sad *rte_ipsec_sad_create(const char *name, const struct rte_ipsec_sad_conf *conf);
void rte_ipsec_sad_free(struct rte_ipsec_sad *sad);

/** key to search for */
union rte_ipsec_sad_key {
        struct { uint32_t spi; uint32_t dip; uint32_t sip; } v4;
        struct {uint32_t spi; uint8_t dip[16]; uint8_t sip[16]; } v6;
};
/** type of key */
enum {
        RTE_IPSEC_SAD_SPI_ONLY, RTE_IPSEC_SAD_SPI_DIP, RTE_IPSEC_SAD_SPI_DIP_SIP
 };

int rte_ipsec_sad_add(struct rte_ipsec_sad *sad, union rte_ipsec_sad_key *key, int key_type, void *sa);
int rte_ipsec_sad_del(struct rte_ipsec_sad *sad, union rte_ipsec_sad_key *key, int key_type);
int rte_ipsec_sad_lookup(const struct rte_ipsec_sad *sad, const union rte_ipsec_sad_key *keys[], uint32_t n,
                                        void *sa[]);
```

# SAD: internals

```
struct rte_ipsec_sad {
        …
        struct rte_hash *hash[RTE_IPSEC_SAD_KEY_TYPE_MASK];
        __extension__ struct hash_cnt cnt_arr[];
};
```

- 3 hash tables (rte_hash).
- Each table keeps entries for a specific key type (*SPI_ONLY* or *SPI_DIP* or *SPI_DIP_SIP*)
- The value in *SPI_ONLY* uses 2 lsb's to indicate presence of more specific keys in *SPI_DIP* and *SPI_DIP_SIP* tables for a given SPI.
- Lookup always starts with *SPI_ONLY* talble and progress to other two based on the
- values of presence bits.
- *cnt_arr[]* entries contain counters for more specific keys in *SPI_DIP* and *SPI_DIP_SIP* tables for given SPI and are used only by add/delete.

# SAD: example

*SPI hash*

| key | value |
|---|---|
| 200 | (NIL \| 0x2) |
| 500 | V1 |
| … | … |
| 100 | (V2 \| 0x3) |

*cnt_arr[]*

| dip | dip+sip |
|---|---|
| 0 | 2 |
| 0 | 0 |
| … | … |
| 1 | 1 |

*SPI+DIP hash*

| key | value |
|---|---|
| … | … |
| 100, 192.0.2.1 | V3 |
| … | … |
| … | … |

*SPI+DIP+SIP hash*

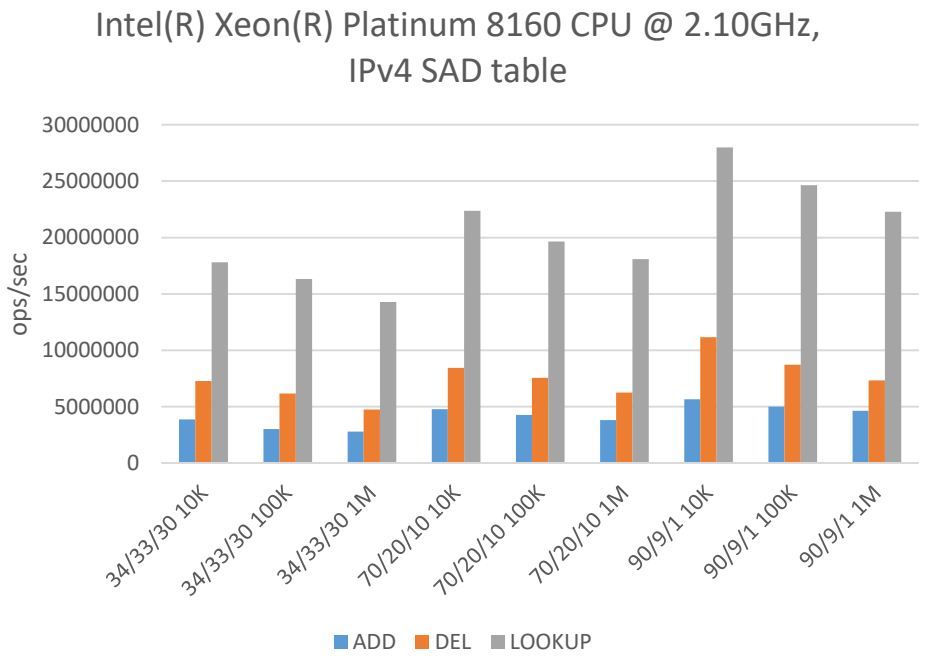
| key | value |
|---|---|
| 100, 192.0.2.1, 203.0.113.1 | V4 |
| … | … |
| 200, 192.0.2.1, 203.0.113.2 | V6 |
| 200, 192.0.2.1, 203.0.113.1 | V5 |

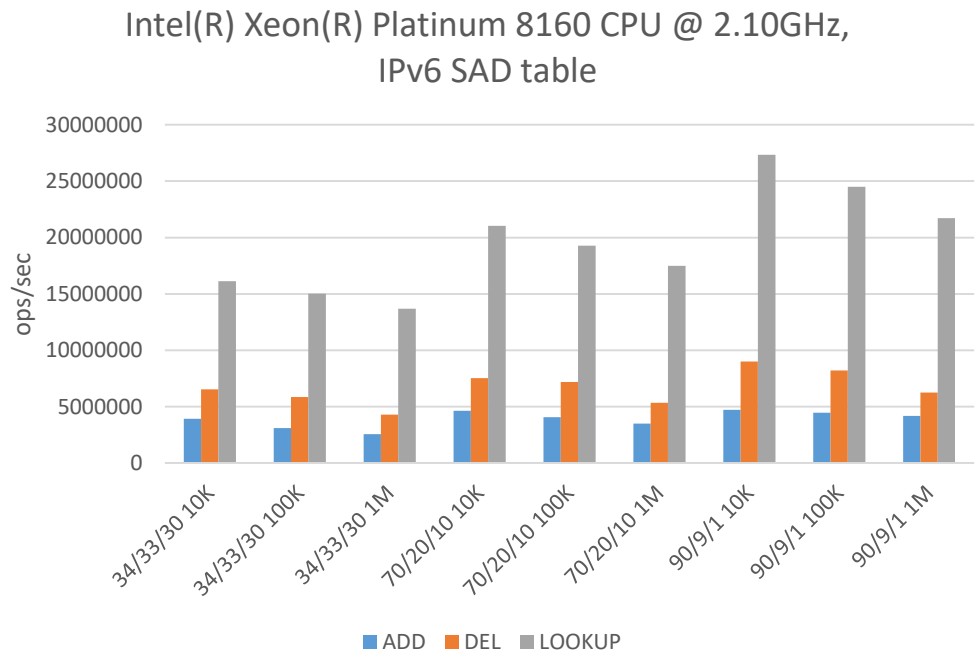
```
rte_ipsec_sad_add(sad, &{.spi=500}, RTE_IPSEC_SAD_SPI_ONLY, V1);
rte_ipsec_sad_add(sad, &{.spi=100}, RTE_IPSEC_SAD_SPI_ONLY, V2);
rte_ipsec_sad_add(sad, &{.spi=100, .dip=192.0.2.1}, RTE_IPSEC_SAD_SPI_DIP, V3);
rte_ipsec_sad_add(sad, &{.spi=100, .dip=192.0.2.1, .sip=203.0.113.1}, RTE_IPSEC_SAD_SPI_DIP_SIP, V4);
rte_ipsec_sad_add(sad, &{.spi=200, .dip=192.0.2.1, .sip=203.0.113.1}, RTE_IPSEC_SAD_SPI_DIP_SIP, V5);
rte_ipsec_sad_add(sad, &{.spi=200, .dip=192.0.2.1, .sip=203.0.113.2}, RTE_IPSEC_SAD_SPI_DIP_SIP, V6);
```

```
rte_ipsec_sad_lookup(sad, &{spi=100, dip=192.0.2.1, sip=198.151.100.100}) ➔ V3
rte_ipsec_sad_lookup(sad, &{spi=100, dip=192.0.2.200, sip=203.0.113.1}) ➔ V2
rte_ipsec_sad_lookup(sad, &{spi=100, dip=192.0.2.1, sip=203.0.113.1}) ➔ V4
```

# SAD: performance

*./testsad <eal opts> -- -n <10K/100K/1M> -l 50M -d [34/33/33, 70/20/10, 90/9/1]*
*(-d : ratio for SA key types  SPI / SPI+DIP / SPI+DIP+SIP)*




*Disclaimer: For more complete information about performance and benchmark results, visit www.intel.com/benchmarks*

# RTE_SECYRITY CPU_CRYPTO

DPDK
DATA PLANE DEVELOPMENT KIT

- Problem:
  - DPDK crypto-dev API is comprehensive, generic and asynchronous (HW oriented)
  - High and unnecessary overhead for SW backed PMDs (AESNI-MB, AESNI-GCM, etc.)
    - allocate/free rte_crypto_op
    - fill/read a rte_crypto_op (costs 3 cache lines)
    - enqueue/dequeue per burst to simulate asynchronous mode
    - dequeue: extra cache line access (check status, retrieve mbuf pointer)
    - based on rte_mbuf (extra layer to de-reference for data buffer address)
  - For bigger packets and slow SW implementation such overhead is less significant.
  - BUT for small packets and/or faster SW implementation the overhead takes larger percentage and will grow even further.
- Propose new API that:
  - works in synchronous mode (function call, all input/output data passed as function parameters)
  - bursts on a per session basis
  - accepts raw data buffers

# CPU_CRYPTO: API

- User level

```
/* new session type and xform*/
enum rte_security_session_action_type {…, RTE_SECURITY_ACTION_TYPE_CPU_CRYPTO};
struct rte_security_cpu_crypto_xform {…};
/* synchronous process function */
struct rte_security_vec {struct iovec *vec; uint32_t num;};
rte_security_process_cpu_crypto_bulk(struct rte_security_ctx *instance, struct rte_security_session *sess,
                    struct rte_security_vec buf[], void *iv[], void *aad[], void *digest[], int status[], uint32_t num);
```

- PMD level

```
/* new function in the PMD ops table */
struct rte_security_ops {…; security_process_cpu_crypto_bulk_t process_cpu_crypto_bulk;}
```

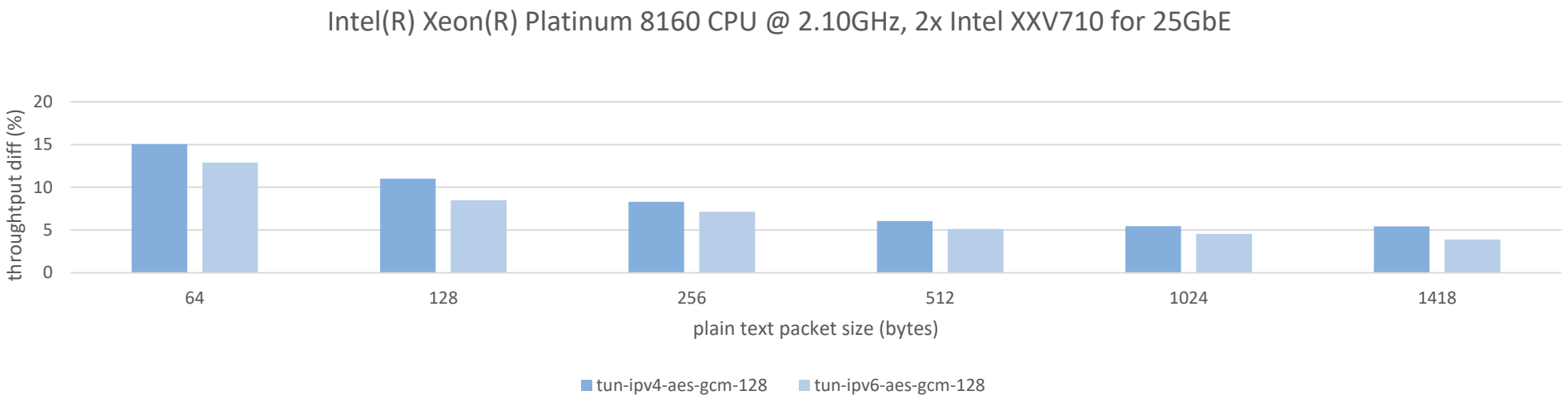
# CPU_CRYPTO: adoption

- Currently supported by: AESNI-GCM, AESNI-MB PMDs
  - Further work for AESNI-MB improvements in plans
- librte_ipsec supports new security type - minimal integration effort for end user
  - examples/ipsec-secgw line changes:
    - 43 for control path
    - 2 for data path

# CPU_CRYPTO: performance

*./ipsec-secgw --lcores=7 -n 4 --vdev="crypto_aesni_gcm0" -w 18:00.0 -w 3b:00.0 -- -p  0x3 -u 1 *

*-P -l --config="(0,0,7),(1,0,7)" …  (1 core, 1 SA, 1 inbound, 1 outbound ports)*




*Disclaimer: For more complete information about performance and benchmark results, visit www.intel.com/benchmarks*

# Multiple sessions per SA

- API brief

```
/* opaque SW representation of the SA (HW neutral) */
struct rte_ipsec_sa;
/* associates SA with particular HW device (session). Same SA can be referred by multiple sessions.*/
struct rte_ipsec_session {
        struct rte_ipsec_sa *sa;
        /** session action type */
        enum rte_security_session_action_type type;
        /** session and related data */
        union {
                struct { struct rte_cryptodev_sym_session *ses; } crypto;
                struct { struct rte_security_session *ses;  … } security;
        };
        …
```

# Multiple sessions per SA: usage scenarios

- single SA / single core / multiple crypto-devs
  - load-balancing (not covered)
  - fall-back sessions
    - inline device does not support processing of IP fragmented IPsec packets
    - add fall-back session on crypto-dev for the same SA
- single SA / multiple cores / multiple crypto-devs
  - fat tunnel (not covered)

# examples/ipsec-secgw: fragmentation & reassembly

Current status:

- Since 19.08 ipsec-secgw has an ability to fragment packets bigger than the MTU, and reassemble fragmented packets.
  - inbound packets: RX => reassemble => IPsec process
  - outbound packets: IPsec process => fragment => TX
- To minimize possible performance effect, reassembly is implemented as RX callback using librte_ip_frag
- To support processing reassembled packets the ipsec-secgw relies on librte_ipsec ability to handle multi-segment packets.
  - Also attached crypto devices have to support 'In Place SGL' offload capability.

# examples/ipsec-secgw: fall-back session

- NIC provides IPsec offload ability … but with some limitations
  - No IP reassemble support in HW
  - Though for many usage scenarios % of fragmented packets is relatively low.
- For INLINE sessions add an ability to have 2 sessions per SA:
  - PRIMARY (INLINE-CRYPTO HW) – used for majority of packets (fast-path).
  - FALL-BACK (CRYPTO-DEV HW/SW) handles packets that can’t be processed by PRIMARY: reassembled packets, etc. (exception-path).
- Works, but few things to be aware about …

# fall-back session: ASYNC vs SYNC

input burst of N IPsec packets (same SA): <*pkt0*, *pkt1*, *pkt2_frag0*, *pkt2_frag1*, *pkt3*, …, *pktN-1*>

after SW reassemble: <*pkt0*, *pkt1*, *pkt2*, *pkt3*, …, *pktN-1*>

## fall-back over ASYNC crypto-device:

```
/* process first two packets */
ipsec_process(pkt0,pkt1);
/*PKT2 enqueued for ASYNC processing and will  be available somewhere in future */
ipsec_prepare(pkt2);
rte_cryptodev_enqueue(pkt2);
/* process rest of the bulk */
ipsec_process(pkt3, …, pktN-1);
/* … sometime later */
ipsec_dequeue(…)=>pkt2; ipsec_process(pkt2);
```

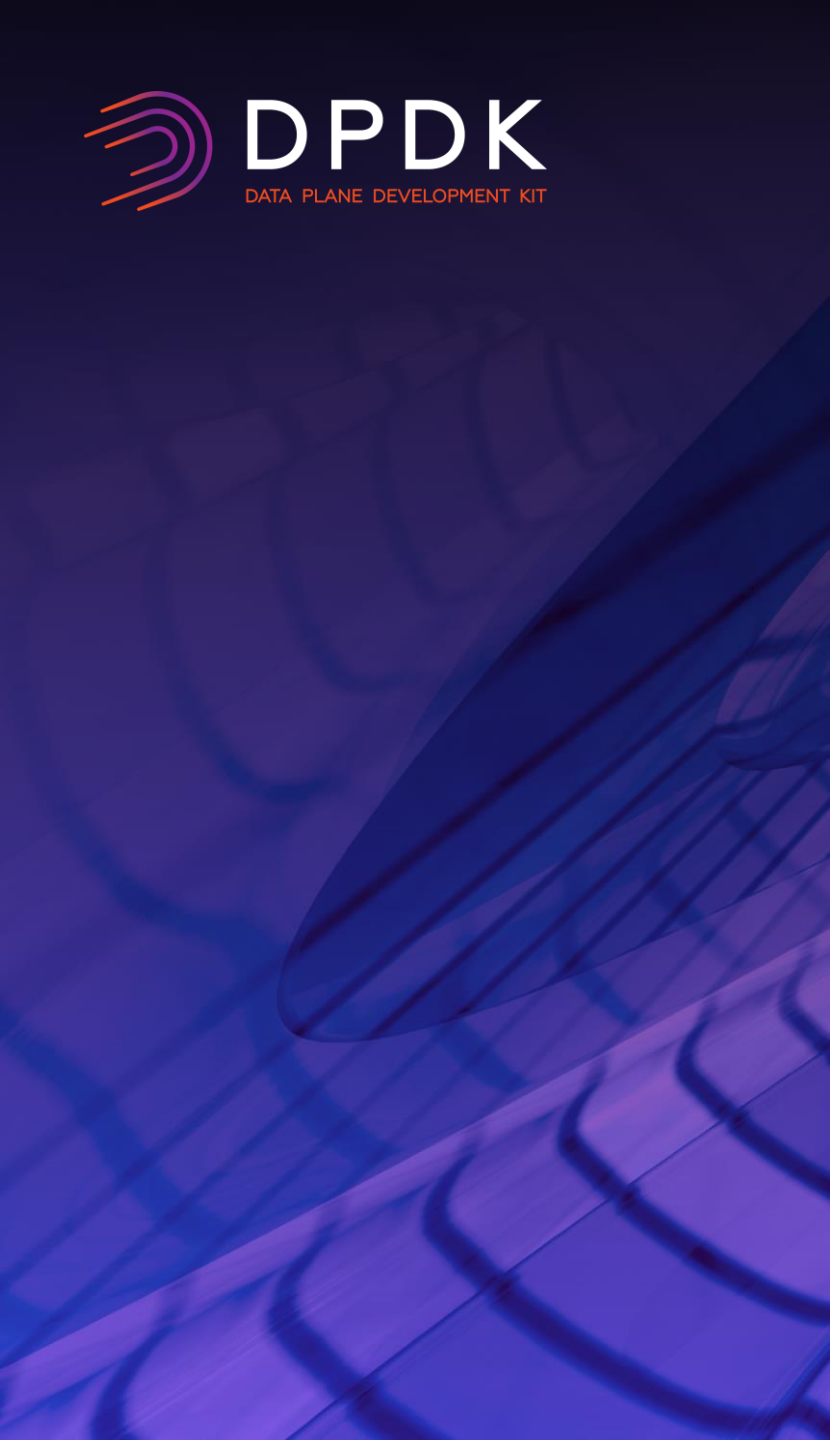
- packet reorder in IPsec processing path:

  <*pkt0*, *pkt1*, *pkt3*, …, *pktN-1*,*pktN*, …, *pktN+M*,*pkt2*>
- replay window size has to be >= N+M *(M value depends on HW/PMD latency)*

## fall-back over SYNC crypto-device:

```
/* process first two packets */
ipsec_process(pkt0,pkt1);
/*SYNC processing for PKT2 */
ipsec_prepare(pkt2);
rte_security_process_cpu_crypto_bulk(pkt2);
ipsec_process(pkt2);
/* process rest of the bulk */
ipsec_process(pkt3, …, pktN-1);
```

- input packet order is preserved
- requires crypto-dev with synchronous API (CPU_CRYPTO)
- Might slowdown fast-path




# Q&A