DPDK

DATA PLANE DEVELOPMENT KIT

# Crypto Device Drivers

***Release 19.11.3***

**Jun 18, 2020**

CHAPTER

ONE

# CRYPTO DEVICE SUPPORTED FUNCTIONALITY MATRICES

## 1.1 Supported Feature Flags

Table 1.1: Features availability in crypto drivers

| Feature | aesni_gcm | aesni_mb | armv8 | caam_jr | ccp | dpaa2_sec | dpaa_sec | kasumi | mvsam | nitrox | null | octeontx | octeontx2 | openssl | qat | snow3g | virtio | zuc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Symmetric crypto | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Asymmetric crypto | | | | | | | | | | | | Y | Y | Y | | | | |
| Sym operation chaining | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| HW Accelerated | | | | Y | Y | Y | Y | | Y | Y | | Y | Y | | Y | | | |
| Protocol offload | | | | Y | | Y | Y | | | | | | | | | | | |
| CPU SSE | Y | Y | | | | | | | | | | | | | | | | |
| CPU AVX | Y | Y | | | | | | | | | | | | | | | | |
| CPU AVX2 | Y | Y | | | | | | | | | | | | | | | | |
| CPU | Y | Y | | | | | | | | | | | | | | | | |

**Note:**

- "In Place SGL" feature flag stands for "In place Scatter-gather list", which means that an input buffer can consist of multiple segments, being the operation in-place (input address = output address).
- "OOP SGL In SGL Out" feature flag stands for "Out-of-place Scatter-gather list Input, Scatter-gather list Output", which means pmd supports different scatter-gather styled input and output buffers (i.e. both can consists of multiple segments).
- "OOP SGL In LB Out" feature flag stands for "Out-of-place Scatter-gather list Input, Linear Buffers Output", which means PMD supports input from scatter-gathered styled buffers, outputting linear buffers (i.e. single segment).
- "OOP LB In SGL Out" feature flag stands for "Out-of-place Linear Buffers Input, Scatter-gather list Output", which means PMD supports input from linear buffer, outputting scatter-gathered styled buffers.
- "OOP LB In LB Out" feature flag stands for "Out-of-place Linear Buffers Input, Linear Buffers Output", which means that Out-of-place operation is supported, with linear input and output buffers.
- "RSA PRIV OP KEY EXP" feature flag means PMD support RSA private key operation (Sign and Decrypt) using exponent key type only.
- "RSA PRIV OP KEY QT" feature flag means PMD support RSA private key operation (Sign and Decrypt) using quintuple (crt) type key only.
- "Digest encrypted" feature flag means PMD support hash-cipher cases, where generated digest is appended to and encrypted with the data.

## 1.2 Supported Cipher Algorithms

Table 1.2: Cipher algorithms in crypto drivers

| Cipher algorithm | aesni_gcm | aesni_mb | armv8 | caam_jr | ccp | dpaa2_sec | dpaa_sec | kasumi | mvsam | nitrox | null | octeontx | octeontx2 | openssl | qat | snow3g | virtio | zuc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NULL | | | | | | | | | Y | | Y | Y | Y | | Y | | | |
| AES CBC (128) | | Y | Y | Y | Y | Y | Y | | Y | Y | | Y | Y | Y | Y | | Y | |
| AES CBC (192) | | Y | | Y | Y | Y | Y | | Y | Y | | Y | Y | Y | Y | | Y | |
| AES CBC (256) | | Y | | Y | Y | Y | Y | | Y | Y | | Y | Y | Y | Y | | Y | |
| AES ECB (128) | | | | | Y | | | | Y | | | | | | | | | |
| AES ECB (192) | | | | | Y | | | | Y | | | | | | | | | |
| AES ECB (256) | | | | | Y | | | | Y | | | | | | | | | |
| AES CTR (128) | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES CTR (192) | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES CTR (256) | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES XTS (128) | | | | | | | | | | | | Y | Y | | Y | | | |
| AES XTS (192) | | | | | | | | | | | | | | | | | | |
| AES XTS (256) | | | | | | | | | | | | Y | Y | | Y | | | |
| AES DOCSIS BPI | | Y | | | | | | | | | | | | | Y | | | |
| 3DES CBC | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |

## 1.3 Supported Authentication Algorithms

Table 1.3: Authentication algorithms in crypto drivers

| Authentication algorithm | aesni_gcm | aesni_mb | armv8 | caam_jr | ccp | dpaa2_sec | dpaa_sec | kasumi | mvsam | nitrox | null | octeontx | octeontx2 | openssl | qat | snow3g | virtio | zuc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NULL | | | | | | | | | Y | | Y | Y | Y | | Y | | | |
| MD5 | | | | | | | | | Y | | | Y | Y | Y | | | | |
| MD5 HMAC | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| SHA1 | | Y | | | Y | | | | Y | | | Y | Y | Y | Y | | | |
| SHA1 HMAC | | Y | Y | Y | Y | Y | Y | | Y | Y | | Y | Y | Y | Y | | Y | |
| SHA224 | | Y | | | Y | | | | Y | | | Y | Y | Y | Y | | | |
| SHA224 HMAC | | Y | | Y | Y | Y | Y | | Y | Y | | Y | Y | Y | Y | | | |
| SHA256 | | Y | | | Y | | | | Y | | | Y | Y | Y | Y | | | |
| SHA256 HMAC | | Y | Y | Y | Y | Y | Y | | Y | Y | | Y | Y | Y | Y | | | |
| SHA384 | | Y | | | Y | | | | Y | | | Y | Y | Y | Y | | | |
| SHA384 HMAC | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| SHA512 | | Y | | | Y | | | | Y | | | Y | Y | Y | Y | | | |
| SHA512 HMAC | | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES XCBC MAC | | Y | | | | | | | | | | | | | Y | | | |
| AES GMAC | Y | Y | | | | | | | Y | | | Y | Y | Y | Y | | | |
| SNOW3G UIA2 | | | | | | Y | Y | | | | | Y | Y | | Y | Y | | |
| KASUMI F9 | | | | | | | | Y | | | | Y | Y | | Y | | | |
| ZUC EIA3 | | | | | | Y | Y | | | | | Y | Y | | Y | | | Y |
| AES CMAC (128) | | Y | | | Y | | | | | | | | | | Y | | | |
| AES CMAC (192) | | | | | Y | | | | | | | | | | | | | |
| AES CMAC (256) | | | | | Y | | | | | | | | | | | | | |
| SHA3_224 | | | | | Y | | | | | | | | | | | | | |
| SHA3_224 | | | | | Y | | | | | | | | | | | | | |

## 1.4 Supported AEAD Algorithms

Table 1.4: AEAD algorithms in crypto drivers

| AEAD algorithm | aesni_gcm | aesni_mb | armv8 | caam_jr | ccp | dpaa2_sec | dpaa_sec | kasumi | mvsam | nitrox | null | octeontx | octeontx2 | openssl | qat | snow3g | virtio | zuc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AES GCM (128) | Y | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES GCM (192) | Y | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES GCM (256) | Y | Y | | Y | Y | Y | Y | | Y | | | Y | Y | Y | Y | | | |
| AES CCM (128) | | Y | | | | | | | | | | | | Y | Y | | | |
| AES CCM (192) | | | | | | | | | | | | | | Y | Y | | | |
| AES CCM (256) | | | | | | | | | | | | | | Y | Y | | | |

## 1.5 Supported Asymmetric Algorithms

Table 1.5: Asymmetric algorithms in crypto drivers

| Asymmetric algorithm | aesni_gcm | aesni_mb | armv8 | caam_jr | ccp | dpaa2_sec | dpaa_sec | kasumi | mvsam | nitrox | null | octeontx | octeontx2 | openssl | qat | snow3g | virtio | zuc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RSA | | | | | | | | | | | | Y | Y | Y | Y | | | |
| DSA | | | | | | | | | | | | | | Y | | | | |
| Modular Exponentiation | | | | | | | | | | | | Y | Y | Y | Y | | | |
| Modular Inversion | | | | | | | | | | | | | | Y | Y | | | |
| Diffie-hellman | | | | | | | | | | | | | | Y | | | | |

CHAPTER

TWO

# AESN-NI MULTI BUFFER CRYPTO POLL MODE DRIVER

The AESNI MB PMD (**librte_pmd_aesni_mb**) provides poll mode crypto driver support for utilizing Intel multi buffer library, see the white paper Fast Multi-buffer IPsec Implementations on Intel® Architecture Processors.

The AES-NI MB PMD has current only been tested on Fedora 21 64-bit with gcc.

## 2.1 Features

AESNI MB PMD has support for:

Cipher algorithms:

- RTE_CRYPTO_CIPHER_AES128_CBC
- RTE_CRYPTO_CIPHER_AES192_CBC
- RTE_CRYPTO_CIPHER_AES256_CBC
- RTE_CRYPTO_CIPHER_AES128_CTR
- RTE_CRYPTO_CIPHER_AES192_CTR
- RTE_CRYPTO_CIPHER_AES256_CTR
- RTE_CRYPTO_CIPHER_AES_DOCSISBPI
- RTE_CRYPTO_CIPHER_DES_CBC
- RTE_CRYPTO_CIPHER_3DES_CBC
- RTE_CRYPTO_CIPHER_DES_DOCSISBPI

Hash algorithms:

- RTE_CRYPTO_HASH_MD5_HMAC
- RTE_CRYPTO_HASH_SHA1_HMAC
- RTE_CRYPTO_HASH_SHA224_HMAC
- RTE_CRYPTO_HASH_SHA256_HMAC
- RTE_CRYPTO_HASH_SHA384_HMAC
- RTE_CRYPTO_HASH_SHA512_HMAC
- RTE_CRYPTO_HASH_AES_XCBC_HMAC

- RTE_CRYPTO_HASH_AES_CMAC
- RTE_CRYPTO_HASH_AES_GMAC
- RTE_CRYPTO_HASH_SHA1
- RTE_CRYPTO_HASH_SHA224
- RTE_CRYPTO_HASH_SHA256
- RTE_CRYPTO_HASH_SHA384
- RTE_CRYPTO_HASH_SHA512

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_CCM
- RTE_CRYPTO_AEAD_AES_GCM

## 2.2 Limitations

- Chained mbufs are not supported.

## 2.3 Installation

To build DPDK with the AESNI_MB_PMD the user is required to download the multi-buffer library from here and compile it on their user system before building DPDK. The latest version of the library supported by this PMD is v0.53, which can be downloaded from https://github.com/01org/intel-ipsec-mb/archive/v0.53.zip.

```
make
make install
```

The library requires NASM to be built. Depending on the library version, it might require a minimum NASM version (e.g. v0.53 requires at least NASM 2.13.03).

NASM is packaged for different OS. However, on some OS the version is too old, so a manual installation is required. In that case, NASM can be downloaded from NASM website. Once it is downloaded, extract it and follow these steps:

```
./configure
make
make install
```

As a reference, the following table shows a mapping between the past DPDK versions and the Multi-Buffer library version supported by them:

Table 2.1: DPDK and Multi-Buffer library version compatibility

| DPDK version | Multi-buffer library version |
|---|---|
| 2.2 - 16.11 | 0.43 - 0.44 |
| 17.02 | 0.44 |
| 17.05 - 17.08 | 0.45 - 0.48 |
| 17.11 | 0.47 - 0.48 |
| 18.02 | 0.48 |
| 18.05 - 19.02 | 0.49 - 0.52 |
| 19.05 - 19.08 | 0.52 |
| 19.11+ | 0.52 - 0.53 |

## 2.4 Initialization

In order to enable this virtual crypto PMD, user must:

- Build the multi buffer library (explained in Installation section).
- Set CONFIG_RTE_LIBRTE_PMD_AESNI_MB=y in config/common_base.

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_aesni_mb") within the application.
- Use –vdev="crypto_aesni_mb" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device (8 by default).
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).

Example:

```
./l2fwd-crypto -l 1 -n 4 --vdev="crypto_aesni_mb,socket_id=0,max_nb_sessions=128" \
-- -p 1 --cdev SW --chain CIPHER_HASH --cipher_algo "aes-cbc" --auth_algo "sha1-hmac"
```

## 2.5 Extra notes

For AES Counter mode (AES-CTR), the library supports two different sizes for Initialization Vector (IV):

- 12 bytes: used mainly for IPsec, as it requires 12 bytes from the user, which internally are appended the counter block (4 bytes), which is set to 1 for the first block (no padding required from the user)
- 16 bytes: when passing 16 bytes, the library will take them and use the last 4 bytes as the initial counter block for the first block.

CHAPTER

THREE

# AES-NI GCM CRYPTO POLL MODE DRIVER

The AES-NI GCM PMD (**librte_pmd_aesni_gcm**) provides poll mode crypto driver support for utilizing Intel multi buffer library (see AES-NI Multi-buffer PMD documentation to learn more about it, including installation).

## 3.1 Features

AESNI GCM PMD has support for:

Authentication algorithms:

- RTE_CRYPTO_AUTH_AES_GMAC

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

## 3.2 Limitations

- In out-of-place operations, chained destination mbufs are not supported.
- Chained mbufs are only supported by RTE_CRYPTO_AEAD_AES_GCM algorithm, not RTE_CRYPTO_AUTH_AES_GMAC.
- Cipher only is not supported.

## 3.3 Installation

To build DPDK with the AESNI_GCM_PMD the user is required to download the multi-buffer library from here and compile it on their user system before building DPDK. The latest version of the library supported by this PMD is v0.53, which can be downloaded in https://github.com/01org/intel-ipsec-mb/archive/v0.53.zip.

```
make
make install
```

The library requires NASM to be built. Depending on the library version, it might require a minimum NASM version (e.g. v0.53 requires at least NASM 2.13.03).

NASM is packaged for different OS. However, on some OS the version is too old, so a manual installation is required. In that case, NASM can be downloaded from NASM website. Once it is downloaded, extract it and follow these steps:

```
./configure
make
make install
```

As a reference, the following table shows a mapping between the past DPDK versions and the external crypto libraries supported by them:

Table 3.1: DPDK and external crypto library version compatibility

| DPDK version | Crypto library version |
|---|---|
| 16.04 - 16.11 | Multi-buffer library 0.43 - 0.44 |
| 17.02 - 17.05 | ISA-L Crypto v2.18 |
| 17.08 - 18.02 | Multi-buffer library 0.46 - 0.48 |
| 18.05 - 19.02 | Multi-buffer library 0.49 - 0.52 |
| 19.05+ | Multi-buffer library 0.52 - 0.53 |

## 3.4 Initialization

In order to enable this virtual crypto PMD, user must:

- Build the multi buffer library (explained in Installation section).
- Set CONFIG_RTE_LIBRTE_PMD_AESNI_GCM=y in config/common_base.

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_aesni_gcm") within the application.
- Use –vdev="crypto_aesni_gcm" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device (8 by default).
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).

Example:

```
./l2fwd-crypto -l 1 -n 4 --vdev="crypto_aesni_gcm,socket_id=0,max_nb_sessions=128" \
-- -p 1 --cdev SW --chain AEAD --aead_algo "aes-gcm"
```

CHAPTER

FOUR

# ARMV8 CRYPTO POLL MODE DRIVER

This code provides the initial implementation of the ARMv8 crypto PMD. The driver uses ARMv8 cryptographic extensions to process chained crypto operations in an optimized way. The core functionality is provided by a low-level library, written in the assembly code.

## 4.1 Features

ARMv8 Crypto PMD has support for the following algorithm pairs:

Supported cipher algorithms:

- `RTE_CRYPTO_CIPHER_AES_CBC`

Supported authentication algorithms:

- `RTE_CRYPTO_AUTH_SHA1_HMAC`
- `RTE_CRYPTO_AUTH_SHA256_HMAC`

## 4.2 Installation

In order to enable this virtual crypto PMD, user must:

- Download ARMv8 crypto library source code from here
- Export the environmental variable ARMV8_CRYPTO_LIB_PATH with the path where the `armv8_crypto` library was downloaded or cloned.
- Build the library by invoking:

```
make -C $ARMV8_CRYPTO_LIB_PATH/
```

- Set CONFIG_RTE_LIBRTE_PMD_ARMV8_CRYPTO=y in config/defconfig_arm64-armv8a-linux-gcc

The corresponding device can be created only if the following features are supported by the CPU:

- `RTE_CPUFLAG_AES`
- `RTE_CPUFLAG_SHA1`
- `RTE_CPUFLAG_SHA2`
- `RTE_CPUFLAG_NEON`

## 4.3 Initialization

User can use app/test application to check how to use this PMD and to verify crypto processing.

Test name is cryptodev_sw_armv8_autotest.

## 4.4 Limitations

- Maximum number of sessions is 2048.
- Only chained operations are supported.
- AES-128-CBC is the only supported cipher variant.
- Cipher input data has to be a multiple of 16 bytes.
- Digest input data has to be a multiple of 8 bytes.

CHAPTER

FIVE

# NXP CAAM JOB RING (CAAM_JR)

The caam_jr PMD provides poll mode crypto driver support for NXP SEC 4.x+ (CAAM) hardware accelerator. More information is available at:

NXP Cryptographic Acceleration Technology.

## 5.1 Architecture

SEC is the SOC's security engine, which serves as NXP's latest cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It also implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, run-time integrity checking, and a hardware random number generator. SEC performs higher-level cryptographic operations than previous NXP cryptographic accelerators. This provides significant improvement to system level performance.

SEC HW accelerator above 4.x+ version are also known as CAAM.

caam_jr PMD is one of DPAA drivers which uses uio interface to interact with Linux kernel for configure and destroy the device instance (ring).

## 5.2 Implementation

SEC provides platform assurance by working with SecMon, which is a companion logic block that tracks the security state of the SOC. SEC is programmed by means of descriptors (not to be confused with frame descriptors (FDs)) that indicate the operations to be performed and link to the message and associated data. SEC incorporates two DMA engines to fetch the descriptors, read the message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that SEC can read and write data scattered in memory. SEC may be configured by means of software for dynamic changes in byte ordering. The default configuration for this version of SEC is little-endian mode.

Note that one physical Job Ring represent one caam_jr device.

## 5.3 Features

The CAAM_JR PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_3DES_CBC`

- RTE_CRYPTO_CIPHER_AES128_CBC
- RTE_CRYPTO_CIPHER_AES192_CBC
- RTE_CRYPTO_CIPHER_AES256_CBC
- RTE_CRYPTO_CIPHER_AES128_CTR
- RTE_CRYPTO_CIPHER_AES192_CTR
- RTE_CRYPTO_CIPHER_AES256_CTR

Hash algorithms:

- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_MD5_HMAC

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

## 5.4 Supported DPAA SoCs

- LS1046A/LS1026A
- LS1043A/LS1023A
- LS1028A
- LS1012A

## 5.5 Limitations

- Hash followed by Cipher mode is not supported
- Only supports the session-oriented API implementation (session-less APIs are not supported).

## 5.6 Prerequisites

caam_jr driver has following dependencies are not part of DPDK and must be installed separately:

- **NXP Linux SDK**

  NXP Linux software development kit (SDK) includes support for the family of QorIQ® ARM-Architecture-based system on chip (SoC) processors and corresponding boards.

  It includes the Linux board support packages (BSPs) for NXP SoCs, a fully operational tool chain, kernel and board specific modules.

SDK and related information can be obtained from: NXP QorIQ SDK.

Currently supported by DPDK:

- NXP SDK **18.09+**.
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

## 5.7 Pre-Installation Configuration

### 5.7.1 Config File Options

The following options can be modified in the `config` file to enable caam_jr PMD.

Please note that enabling debugging options may affect system performance.

- `CONFIG_RTE_LIBRTE_PMD_CAAM_JR` (default `n`) By default it is only enabled in common_linux config. Toggle compilation of the `librte_pmd_caam_jr` driver.
- `CONFIG_RTE_LIBRTE_PMD_CAAM_JR_BE` (default `n`) By default it is disabled. It can be used when the underlying hardware supports the CAAM in BE mode. LS1043A, LS1046A and LS1012A support CAAM in BE mode. LS1028A supports CAAM in LE mode.

## 5.8 Installations

To compile the caam_jr PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-armv8a-linux-gcc install
```

## 5.9 Enabling logs

For enabling logs, use the following EAL parameter:

```
./your_crypto_application <EAL args> --log-level=pmd.crypto.caam,<level>
```

CHAPTER

SIX

# AMD CCP POLL MODE DRIVER

This code provides the initial implementation of the ccp poll mode driver. The CCP poll mode driver library (librte_pmd_ccp) implements support for AMD's cryptographic co-processor (CCP). The CCP PMD is a virtual crypto poll mode driver which schedules crypto operations to one or more available CCP hardware engines on the platform. The CCP PMD provides poll mode crypto driver support for the following hardware accelerator devices:

```
AMD Cryptographic Co-processor (0x1456)
AMD Cryptographic Co-processor (0x1468)
```

## 6.1 Features

CCP crypto PMD has support for:

Cipher algorithms:

- RTE_CRYPTO_CIPHER_AES_CBC
- RTE_CRYPTO_CIPHER_AES_ECB
- RTE_CRYPTO_CIPHER_AES_CTR
- RTE_CRYPTO_CIPHER_3DES_CBC

Hash algorithms:

- RTE_CRYPTO_AUTH_SHA1
- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_MD5_HMAC
- RTE_CRYPTO_AUTH_AES_CMAC

- RTE_CRYPTO_AUTH_SHA3_224
- RTE_CRYPTO_AUTH_SHA3_224_HMAC
- RTE_CRYPTO_AUTH_SHA3_256
- RTE_CRYPTO_AUTH_SHA3_256_HMAC
- RTE_CRYPTO_AUTH_SHA3_384
- RTE_CRYPTO_AUTH_SHA3_384_HMAC
- RTE_CRYPTO_AUTH_SHA3_512
- RTE_CRYPTO_AUTH_SHA3_512_HMAC

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

## 6.2 Installation

To compile ccp PMD, it has to be enabled in the config/common_base file and openssl packages have to be installed in the build environment.

- CONFIG_RTE_LIBRTE_PMD_CCP=y

For Ubuntu 16.04 LTS use below to install openssl in the build system:

```
sudo apt-get install openssl
```

This code was verified on Ubuntu 16.04.

## 6.3 Initialization

Bind the CCP devices to DPDK UIO driver module before running the CCP PMD stack. e.g. for the 0x1456 device:

```
cd to the top-level DPDK directory
modprobe uio
insmod ./build/kmod/igb_uio.ko
echo "1022 1456" > /sys/bus/pci/drivers/igb_uio/new_id
```

Another way to bind the CCP devices to DPDK UIO driver is by using the `dpdk-devbind.py` script. The following command assumes `BFD` as `0000:09:00.2`:

```
cd to the top-level DPDK directory
./usertools/dpdk-devbind.py -b igb_uio 0000:09:00.2
```

In order to enable the ccp crypto PMD, user must set CONFIG_RTE_LIBRTE_PMD_CCP=y in config/common_base.

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_ccp") within the application.
- Use –vdev="crypto_ccp" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated. (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device.
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).
- ccp_auth_opt: Specify authentication operations to perform on CPU using openssl APIs.

To validate ccp pmd, l2fwd-crypto example can be used with following command:

```
sudo ./build/l2fwd-crypto -l 1 -n 4 --vdev "crypto_ccp" -- -p 0x1
--chain CIPHER_HASH --cipher_op ENCRYPT --cipher_algo aes-cbc
--cipher_key 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f
--cipher_iv 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:ff
--auth_op GENERATE --auth_algo sha1-hmac
--auth_key 11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
```

The CCP PMD also supports computing authentication over CPU with cipher offloaded to CCP. To enable this feature, pass an additional argument as ccp_auth_opt=1 to –vdev parameters as following:

```
sudo ./build/l2fwd-crypto -l 1 -n 4 --vdev "crypto_ccp,ccp_auth_opt=1" -- -p 0x1
--chain CIPHER_HASH --cipher_op ENCRYPT --cipher_algo aes-cbc
--cipher_key 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f
--cipher_iv 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:ff
--auth_op GENERATE --auth_algo sha1-hmac
--auth_key 11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
```

## 6.4 Limitations

- Chained mbufs are not supported.
- MD5_HMAC is supported only for CPU based authentication.

CHAPTER

SEVEN

# NXP DPAA2 CAAM (DPAA2_SEC)

The DPAA2_SEC PMD provides poll mode crypto driver support for NXP DPAA2 CAAM hardware accelerator.

## 7.1 Architecture

SEC is the SOC's security engine, which serves as NXP's latest cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It also implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, run-time integrity checking, and a hardware random number generator. SEC performs higher-level cryptographic operations than previous NXP cryptographic accelerators. This provides significant improvement to system level performance.

DPAA2_SEC is one of the hardware resource in DPAA2 Architecture. More information on DPAA2 Architecture is described in dpaa2_overview.

DPAA2_SEC PMD is one of DPAA2 drivers which interacts with Management Complex (MC) portal to access the hardware object - DPSECI. The MC provides access to create, discover, connect, configure and destroy dpseci objects in DPAA2_SEC PMD.

DPAA2_SEC PMD also uses some of the other hardware resources like buffer pools, queues, queue portals to store and to enqueue/dequeue data to the hardware SEC.

DPSECI objects are detected by PMD using a resource container called DPRC (like in dpaa2_overview).

For example:

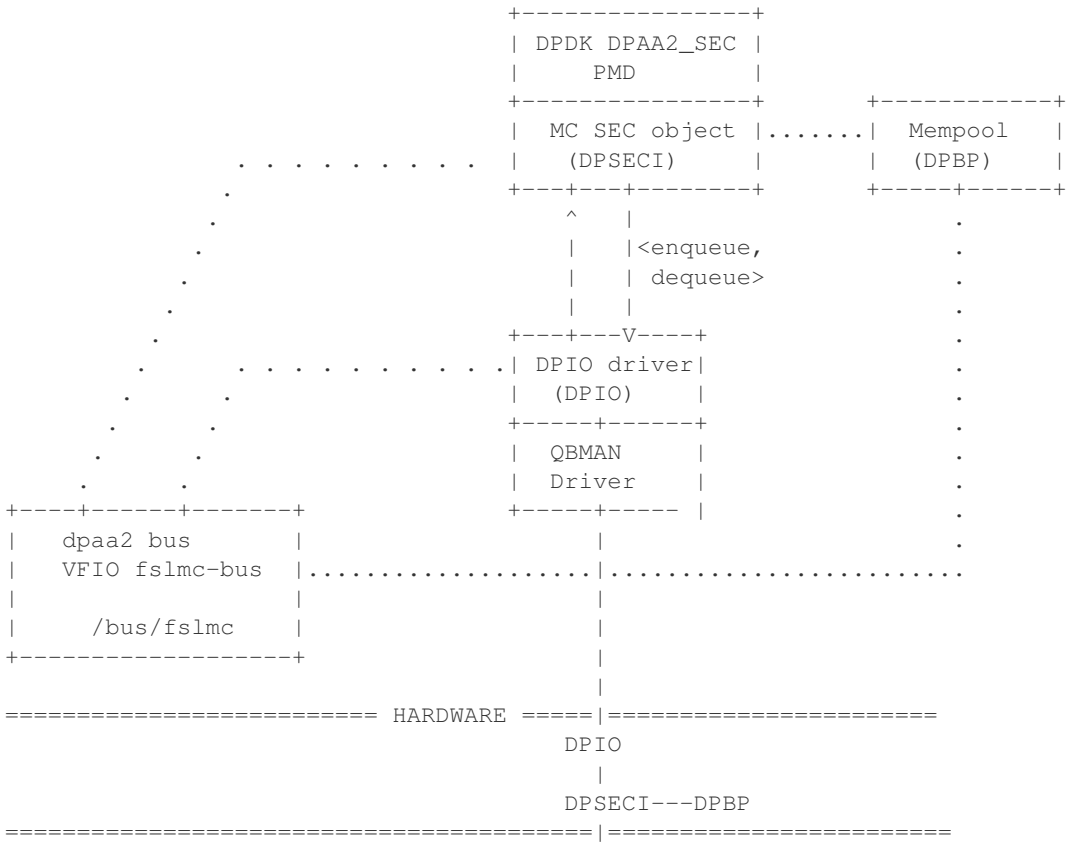
```
DPRC.1 (bus)
  |
  +--+--------+-------+-------+-------+---------+
     |        |       |       |       |         |
   DPMCP.1  DPIO.1  DPBP.1  DPNI.1  DPMAC.1  DPSECI.1
   DPMCP.2  DPIO.2          DPNI.2  DPMAC.2  DPSECI.2
   DPMCP.3
```

## 7.2 Implementation

SEC provides platform assurance by working with SecMon, which is a companion logic block that tracks the security state of the SOC. SEC is programmed by means of descriptors (not to be confused with frame descriptors (FDs)) that indicate the operations to be performed and link to the message and associated data. SEC incorporates two DMA engines to fetch the descriptors, read the message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that SEC

can read and write data scattered in memory. SEC may be configured by means of software for dynamic changes in byte ordering. The default configuration for this version of SEC is little-endian mode.

A block diagram similar to dpaa2 NIC is shown below to show where DPAA2_SEC fits in the DPAA2 Bus model




## 7.3 Features

The DPAA2_SEC PMD has support for:

Cipher algorithms:

- RTE_CRYPTO_CIPHER_3DES_CBC
- RTE_CRYPTO_CIPHER_AES128_CBC
- RTE_CRYPTO_CIPHER_AES192_CBC
- RTE_CRYPTO_CIPHER_AES256_CBC
- RTE_CRYPTO_CIPHER_AES128_CTR
- RTE_CRYPTO_CIPHER_AES192_CTR
- RTE_CRYPTO_CIPHER_AES256_CTR

Hash algorithms:

- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224_HMAC

- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_MD5_HMAC

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

## 7.4 Supported DPAA2 SoCs

- LS2160A
- LS2084A/LS2044A
- LS2088A/LS2048A
- LS1088A/LS1048A

## 7.5 Whitelisting & Blacklisting

For blacklisting a DPAA2 SEC device, following commands can be used.

```
<dpdk app> <EAL args> -b "fslmc:dpseci.x" -- ...
```

Where x is the device object id as configured in resource container.

## 7.6 Limitations

- Hash followed by Cipher mode is not supported
- Only supports the session-oriented API implementation (session-less APIs are not supported).

## 7.7 Prerequisites

DPAA2_SEC driver has similar pre-requisites as described in dpaa2_overview. The following dependencies are not part of DPDK and must be installed separately:

See ../platform/dpaa2 for setup information

Currently supported by DPDK:

- NXP SDK **19.09+**.
- MC Firmware version **10.18.0** and higher.
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

## 7.8 Pre-Installation Configuration

### 7.8.1 Config File Options

Basic DPAA2 config file options are described in dpaa2_overview. In addition to those, the following options can be modified in the `config` file to enable DPAA2_SEC PMD.

Please note that enabling debugging options may affect system performance.

- `CONFIG_RTE_LIBRTE_PMD_DPAA2_SEC` (default `n`) By default it is only enabled in defconfig_arm64-dpaa-* config. Toggle compilation of the `librte_pmd_dpaa2_sec` driver.

## 7.9 Installations

To compile the DPAA2_SEC PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-dpaa-linux-gcc install
```

## 7.10 Enabling logs

For enabling logs, use the following EAL parameter:

```
./your_crypto_application <EAL args> --log-level=pmd.crypto.dpaa2:<level>
```

Using `crypto.dpaa2` as log matching criteria, all Crypto PMD logs can be enabled which are lower than logging `level`.

CHAPTER

EIGHT

# NXP DPAA CAAM (DPAA_SEC)

The DPAA_SEC PMD provides poll mode crypto driver support for NXP DPAA CAAM hardware accelerator.

## 8.1 Architecture

SEC is the SOC's security engine, which serves as NXP's latest cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It also implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, run-time integrity checking, and a hardware random number generator. SEC performs higher-level cryptographic operations than previous NXP cryptographic accelerators. This provides significant improvement to system level performance.

DPAA_SEC is one of the hardware resource in DPAA Architecture. More information on DPAA Architecture is described in dpaa_overview.

DPAA_SEC PMD is one of DPAA drivers which interacts with QBMAN to create, configure and destroy the device instance using queue pair with CAAM portal.

DPAA_SEC PMD also uses some of the other hardware resources like buffer pools, queues, queue portals to store and to enqueue/dequeue data to the hardware SEC.

## 8.2 Implementation

SEC provides platform assurance by working with SecMon, which is a companion logic block that tracks the security state of the SOC. SEC is programmed by means of descriptors (not to be confused with frame descriptors (FDs)) that indicate the operations to be performed and link to the message and associated data. SEC incorporates two DMA engines to fetch the descriptors, read the message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that SEC can read and write data scattered in memory. SEC may be configured by means of software for dynamic changes in byte ordering. The default configuration for this version of SEC is little-endian mode.

## 8.3 Features

The DPAA PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_3DES_CBC`

- RTE_CRYPTO_CIPHER_AES128_CBC
- RTE_CRYPTO_CIPHER_AES192_CBC
- RTE_CRYPTO_CIPHER_AES256_CBC
- RTE_CRYPTO_CIPHER_AES128_CTR
- RTE_CRYPTO_CIPHER_AES192_CTR
- RTE_CRYPTO_CIPHER_AES256_CTR
- RTE_CRYPTO_CIPHER_SNOW3G_UEA2
- RTE_CRYPTO_CIPHER_ZUC_EEA3

Hash algorithms:

- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_SNOW3G_UIA2
- RTE_CRYPTO_AUTH_MD5_HMAC
- RTE_CRYPTO_AUTH_ZUC_EIA3

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

## 8.4 Supported DPAA SoCs

- LS1046A/LS1026A
- LS1043A/LS1023A

## 8.5 Whitelisting & Blacklisting

For blacklisting a DPAA device, following commands can be used.

```
<dpdk app> <EAL args> -b "dpaa:dpaa_sec-X" -- ...
e.g. "dpaa:dpaa_sec-1"

or to disable all 4 SEC devices
-b "dpaa:dpaa_sec-1"  -b "dpaa:dpaa_sec-2" -b "dpaa:dpaa_sec-3" -b "dpaa:dpaa_sec-4"
```

## 8.6 Limitations

- Hash followed by Cipher mode is not supported
- Only supports the session-oriented API implementation (session-less APIs are not supported).

## 8.7 Prerequisites

DPAA_SEC driver has similar pre-requisites as described in dpaa_overview.

See ../platform/dpaa for setup information

- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

## 8.8 Pre-Installation Configuration

### 8.8.1 Config File Options

Basic DPAA config file options are described in dpaa_overview. In addition to those, the following options can be modified in the `config` file to enable DPAA_SEC PMD.

Please note that enabling debugging options may affect system performance.

- `CONFIG_RTE_LIBRTE_PMD_DPAA_SEC` (default `n`) By default it is only enabled in defconfig_arm64-dpaa-* config. Toggle compilation of the `librte_pmd_dpaa_sec` driver.

## 8.9 Installations

To compile the DPAA_SEC PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-dpaa-linux-gcc install
```

## 8.10 Enabling logs

For enabling logs, use the following EAL parameter:

```
./your_crypto_application <EAL args> --log-level=pmd.crypto.dpaa:<level>
```

Using `pmd.crypto.dpaa` as log matching criteria, all Crypto PMD logs can be enabled which are lower than logging `level`.

CHAPTER

NINE

# KASUMI CRYPTO POLL MODE DRIVER

The KASUMI PMD (**librte_pmd_kasumi**) provides poll mode crypto driver support for utilizing Intel Libsso library, which implements F8 and F9 functions for KASUMI UEA1 cipher and UIA1 hash algorithms.

## 9.1 Features

KASUMI PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_CIPHER_KASUMI_F8

Authentication algorithm:

- RTE_CRYPTO_AUTH_KASUMI_F9

## 9.2 Limitations

- Chained mbufs are not supported.
- KASUMI(F9) supported only if hash offset and length field is byte-aligned.
- In-place bit-level operations for KASUMI(F8) are not supported (if length and/or offset of data to be ciphered is not byte-aligned).

## 9.3 Installation

To build DPDK with the KASUMI_PMD the user is required to download the export controlled `libsso_kasumi` library, by registering in Intel Resource & Design Center. Once approval has been granted, the user needs to search for *Kasumi F8 F9 3GPP cryptographic algorithms Software Library* to download the library or directly through this link. After downloading the library, the user needs to unpack and compile it on their system before building DPDK:

```
make
```

**Note**: When encrypting with KASUMI F8, by default the library encrypts full blocks of 8 bytes, regardless the number of bytes to be encrypted provided (which leads to a possible buffer overflow). To avoid this situation, it is necessary not to pass 3GPP_SAFE_BUFFERS as a compilation flag. Also, this is required when using chained operations (cipher-then-auth/auth-then-cipher). For this, in the Makefile of the library, make sure that this flag is commented out:

```
#EXTRA_CFLAGS  += -D_3GPP_SAFE_BUFFERS
```

**Note**: To build the PMD as a shared library, the libsso_kasumi library must be built as follows:

```
make KASUMI_CFLAGS=-DKASUMI_C
```

## 9.4 Initialization

In order to enable this virtual crypto PMD, user must:

- Export the environmental variable LIBSSO_KASUMI_PATH with the path where the library was extracted (kasumi folder).
- Build the LIBSSO library (explained in Installation section).
- Set CONFIG_RTE_LIBRTE_PMD_KASUMI=y in config/common_base.

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_kasumi") within the application.
- Use –vdev="crypto_kasumi" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device (8 by default).
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).

Example:

```
./l2fwd-crypto -l 1 -n 4 --vdev="crypto_kasumi,socket_id=0,max_nb_sessions=128" \
-- -p 1 --cdev SW --chain CIPHER_ONLY --cipher_algo "kasumi-f8"
```

## 9.5 Extra notes on KASUMI F9

When using KASUMI F9 authentication algorithm, the input buffer must be constructed according to the 3GPP KASUMI specifications (section 4.4, page 13): http://cryptome.org/3gpp/35201-900.pdf. Input buffer has to have COUNT (4 bytes), FRESH (4 bytes), MESSAGE and DIRECTION (1 bit) concatenated. After the DIRECTION bit, a single '1' bit is appended, followed by between 0 and 7 '0' bits, so that the total length of the buffer is multiple of 8 bits. Note that the actual message can be any length, specified in bits.

Once this buffer is passed this way, when creating the crypto operation, length of data to authenticate (op.sym.auth.data.length) must be the length of all the items described above, including the padding at the end. Also, offset of data to authenticate (op.sym.auth.data.offset) must be such that points at the start of the COUNT bytes.

CHAPTER

TEN

# CAVIUM OCTEON TX CRYPTO POLL MODE DRIVER

The OCTEON TX crypto poll mode driver provides support for offloading cryptographic operations to cryptographic accelerator units on **OCTEON TX** ® family of processors (CN8XXX). The OCTEON TX crypto poll mode driver enqueues the crypto request to this accelerator and dequeues the response once the operation is completed.

## 10.1 Supported Symmetric Crypto Algorithms

### 10.1.1 Cipher Algorithms

- RTE_CRYPTO_CIPHER_NULL
- RTE_CRYPTO_CIPHER_3DES_CBC
- RTE_CRYPTO_CIPHER_3DES_ECB
- RTE_CRYPTO_CIPHER_AES_CBC
- RTE_CRYPTO_CIPHER_AES_CTR
- RTE_CRYPTO_CIPHER_AES_XTS
- RTE_CRYPTO_CIPHER_DES_CBC
- RTE_CRYPTO_CIPHER_KASUMI_F8
- RTE_CRYPTO_CIPHER_SNOW3G_UEA2
- RTE_CRYPTO_CIPHER_ZUC_EEA3

### 10.1.2 Hash Algorithms

- RTE_CRYPTO_AUTH_NULL
- RTE_CRYPTO_AUTH_AES_GMAC
- RTE_CRYPTO_AUTH_KASUMI_F9
- RTE_CRYPTO_AUTH_MD5
- RTE_CRYPTO_AUTH_MD5_HMAC
- RTE_CRYPTO_AUTH_SHA1
- RTE_CRYPTO_AUTH_SHA1_HMAC

- RTE_CRYPTO_AUTH_SHA224
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_SNOW3G_UIA2
- RTE_CRYPTO_AUTH_ZUC_EIA3

### 10.1.3 AEAD Algorithms

- RTE_CRYPTO_AEAD_AES_GCM

## 10.2 Supported Asymmetric Crypto Algorithms

- RTE_CRYPTO_ASYM_XFORM_RSA
- RTE_CRYPTO_ASYM_XFORM_MODEX

## 10.3 Config flags

For compiling the OCTEON TX crypto poll mode driver, please check if the CONFIG_RTE_LIBRTE_PMD_OCTEONTX_CRYPTO setting is set to *y* in config/common_base file.

- CONFIG_RTE_LIBRTE_PMD_OCTEONTX_CRYPTO=y

## 10.4 Compilation

The OCTEON TX crypto poll mode driver can be compiled either natively on **OCTEON TX** ® board or cross-compiled on an x86 based platform.

Refer ../platform/octeontx for details about setting up the platform and building DPDK applications.

---

**Note:** OCTEON TX crypto PF driver needs microcode to be available at */lib/firmware/* directory. Refer SDK documents for further information.

---

SDK and related information can be obtained from: Cavium support site.

## 10.5 Execution

The number of crypto VFs to be enabled can be controlled by setting sysfs entry, *sriov_numvfs*, for the corresponding PF driver.

```
echo <num_vfs> > /sys/bus/pci/devices/<dev_bus_id>/sriov_numvfs
```

The device bus ID, *dev_bus_id*, to be used in the above step can be found out by using dpdk-devbind.py script. The OCTEON TX crypto PF device need to be identified and the corresponding device number can be used to tune various PF properties.

Once the required VFs are enabled, dpdk-devbind.py script can be used to identify the VFs. To be accessible from DPDK, VFs need to be bound to vfio-pci driver:

```
cd <dpdk directory>
./usertools/dpdk-devbind.py -u <vf device no>
./usertools/dpdk-devbind.py -b vfio-pci <vf device no>
```

Appropriate huge page need to be setup in order to run the DPDK example applications.

```
echo 8 > /sys/kernel/mm/hugepages/hugepages-524288kB/nr_hugepages
mkdir /mnt/huge
mount -t hugetlbfs nodev /mnt/huge
```

Example applications can now be executed with crypto operations offloaded to OCTEON TX crypto PMD.

```
./build/ipsec-secgw --log-level=8 -c 0xff -- -P -p 0x3 -u 0x2 --config
"(1,0,0),(0,0,0)" -f ep1.cfg
```

## 10.6 Testing

The symmetric crypto operations on OCTEON TX crypto PMD may be verified by running the test application:

```
./test
RTE>>cryptodev_octeontx_autotest
```

The asymmetric crypto operations on OCTEON TX crypto PMD may be verified by running the test application:

```
./test
RTE>>cryptodev_octeontx_asym_autotest
```

CHAPTER

ELEVEN

# MARVELL OCTEON TX2 CRYPTO POLL MODE DRIVER

The OCTEON TX2 crypto poll mode driver provides support for offloading cryptographic operations to cryptographic accelerator units on the **OCTEON TX2** ® family of processors (CN9XXX).

More information about OCTEON TX2 SoCs may be obtained from https://www.marvell.com

## 11.1 Features

The OCTEON TX2 crypto PMD has support for:

### 11.1.1 Symmetric Crypto Algorithms

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_NULL`
- `RTE_CRYPTO_CIPHER_3DES_CBC`
- `RTE_CRYPTO_CIPHER_3DES_ECB`
- `RTE_CRYPTO_CIPHER_AES_CBC`
- `RTE_CRYPTO_CIPHER_AES_CTR`
- `RTE_CRYPTO_CIPHER_AES_XTS`
- `RTE_CRYPTO_CIPHER_DES_CBC`
- `RTE_CRYPTO_CIPHER_KASUMI_F8`
- `RTE_CRYPTO_CIPHER_SNOW3G_UEA2`
- `RTE_CRYPTO_CIPHER_ZUC_EEA3`

Hash algorithms:

- `RTE_CRYPTO_AUTH_NULL`
- `RTE_CRYPTO_AUTH_AES_GMAC`
- `RTE_CRYPTO_AUTH_KASUMI_F9`
- `RTE_CRYPTO_AUTH_MD5`
- `RTE_CRYPTO_AUTH_MD5_HMAC`
- `RTE_CRYPTO_AUTH_SHA1`

- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_SNOW3G_UIA2
- RTE_CRYPTO_AUTH_ZUC_EIA3

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

### 11.1.2 Asymmetric Crypto Algorithms

- RTE_CRYPTO_ASYM_XFORM_RSA
- RTE_CRYPTO_ASYM_XFORM_MODEX

## 11.2 Installation

The OCTEON TX2 crypto PMD may be compiled natively on an OCTEON TX2 platform or cross-compiled on an x86 platform.

Enable OCTEON TX2 crypto PMD in your config file:

- CONFIG_RTE_LIBRTE_PMD_OCTEONTX2_CRYPTO=y

Refer to ../platform/octeontx2 for instructions to build your DPDK application.

---

**Note:** The OCTEON TX2 crypto PMD uses services from the kernel mode OCTEON TX2 crypto PF driver in linux. This driver is included in the OCTEON TX SDK.

---

## 11.3 Initialization

List the CPT PF devices available on your OCTEON TX2 platform:

```
lspci -d:a0fd
```

`a0fd` is the CPT PF device id. You should see output similar to:

```
0002:10:00.0 Class 1080: Device 177d:a0fd
```

Set `sriov_numvfs` on the CPT PF device, to create a VF:

```
echo 1 > /sys/bus/pci/drivers/octeontx2-cpt/0002:10:00.0/sriov_numvfs
```

Bind the CPT VF device to the vfio_pci driver:

```
echo '177d a0fe' > /sys/bus/pci/drivers/vfio-pci/new_id
echo 0002:10:00.1 > /sys/bus/pci/devices/0002:10:00.1/driver/unbind
echo 0002:10:00.1 > /sys/bus/pci/drivers/vfio-pci/bind
```

Another way to bind the VF would be to use the `dpdk-devbind.py` script:

```
cd <dpdk directory>
./usertools/dpdk-devbind.py -u 0002:10:00.1
./usertools/dpdk-devbind.py -b vfio-pci 0002:10.00.1
```

---

**Note:** Ensure that sufficient huge pages are available for your application:

```
echo 8 > /sys/kernel/mm/hugepages/hugepages-524288kB/nr_hugepages
```

Refer to linux_gsg_hugepages for more details.

---

## 11.4 Debugging Options

Table 11.1: OCTEON TX2 crypto PMD debug options

| # | Component | EAL log command |
|---|---|---|
| 1 | CPT | –log-level='pmd.crypto.octeontx2,8' |

## 11.5 Testing

The symmetric crypto operations on OCTEON TX2 crypto PMD may be verified by running the test application:

```
./test
RTE>>cryptodev_octeontx2_autotest
```

The asymmetric crypto operations on OCTEON TX2 crypto PMD may be verified by running the test application:

```
./test
RTE>>cryptodev_octeontx2_asym_autotest
```

CHAPTER

TWELVE

# OPENSSL CRYPTO POLL MODE DRIVER

This code provides the initial implementation of the openssl poll mode driver. All cryptography operations are using Openssl library crypto API. Each algorithm uses EVP interface from openssl API - which is recommended by Openssl maintainers.

For more details about openssl library please visit openssl webpage: https://www.openssl.org/

## 12.1 Features

OpenSSL PMD has support for:

Supported cipher algorithms:

- RTE_CRYPTO_CIPHER_3DES_CBC
- RTE_CRYPTO_CIPHER_AES_CBC
- RTE_CRYPTO_CIPHER_AES_CTR
- RTE_CRYPTO_CIPHER_3DES_CTR
- RTE_CRYPTO_CIPHER_DES_DOCSISBPI

Supported authentication algorithms:

- RTE_CRYPTO_AUTH_AES_GMAC
- RTE_CRYPTO_AUTH_MD5
- RTE_CRYPTO_AUTH_SHA1
- RTE_CRYPTO_AUTH_SHA224
- RTE_CRYPTO_AUTH_SHA256
- RTE_CRYPTO_AUTH_SHA384
- RTE_CRYPTO_AUTH_SHA512
- RTE_CRYPTO_AUTH_MD5_HMAC
- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384_HMAC

- RTE_CRYPTO_AUTH_SHA512_HMAC

Supported AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM
- RTE_CRYPTO_AEAD_AES_CCM

Supported Asymmetric Crypto algorithms:

- RTE_CRYPTO_ASYM_XFORM_RSA
- RTE_CRYPTO_ASYM_XFORM_DSA
- RTE_CRYPTO_ASYM_XFORM_DH
- RTE_CRYPTO_ASYM_XFORM_MODINV
- RTE_CRYPTO_ASYM_XFORM_MODEX

## 12.2 Installation

To compile openssl PMD, it has to be enabled in the config/common_base file and appropriate openssl packages have to be installed in the build environment.

The newest openssl library version is supported:

- 1.0.2h-fips 3 May 2016.

Older versions that were also verified:

- 1.0.1f 6 Jan 2014
- 1.0.1 14 Mar 2012

For Ubuntu 14.04 LTS these packages have to be installed in the build system:

```
sudo apt-get install openssl
sudo apt-get install libc6-dev-i386 # for i686-native-linux-gcc target
```

This code was also verified on Fedora 24. This code has NOT been verified on FreeBSD yet.

## 12.3 Initialization

User can use app/test application to check how to use this pmd and to verify crypto processing.

Test name is cryptodev_openssl_autotest. For asymmetric crypto operations testing, run cryptodev_openssl_asym_autotest.

To verify real traffic l2fwd-crypto example can be used with this command:

```
sudo ./build/l2fwd-crypto -l 0-1 -n 4 --vdev "crypto_openssl"
--vdev "crypto_openssl"-- -p 0x3 --chain CIPHER_HASH
--cipher_op ENCRYPT --cipher_algo AES_CBC
--cipher_key 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f
--iv 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:ff
--auth_op GENERATE --auth_algo SHA1_HMAC
--auth_key 11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
```

## 12.4 Limitations

- Maximum number of sessions is 2048.
- Chained mbufs are supported only for source mbuf (destination must be contiguous).
- Hash only is not supported for GCM and GMAC.
- Cipher only is not supported for GCM and GMAC.

CHAPTER

THIRTEEN

# MVSAM CRYPTO POLL MODE DRIVER

The MVSAM CRYPTO PMD (**librte_crypto_mvsam_pmd**) provides poll mode crypto driver support by utilizing MUSDK library, which provides cryptographic operations acceleration by using Security Acceleration Engine (EIP197) directly from user-space with minimum overhead and high performance.

Detailed information about SoCs that use MVSAM crypto driver can be obtained here:

- https://www.marvell.com/embedded-processors/armada-70xx/
- https://www.marvell.com/embedded-processors/armada-80xx/
- https://www.marvell.com/embedded-processors/armada-3700/

## 13.1 Features

MVSAM CRYPTO PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_NULL`
- `RTE_CRYPTO_CIPHER_AES_CBC`
- `RTE_CRYPTO_CIPHER_AES_CTR`
- `RTE_CRYPTO_CIPHER_AES_ECB`
- `RTE_CRYPTO_CIPHER_3DES_CBC`
- `RTE_CRYPTO_CIPHER_3DES_CTR`
- `RTE_CRYPTO_CIPHER_3DES_ECB`

Hash algorithms:

- `RTE_CRYPTO_AUTH_NULL`
- `RTE_CRYPTO_AUTH_MD5`
- `RTE_CRYPTO_AUTH_MD5_HMAC`
- `RTE_CRYPTO_AUTH_SHA1`
- `RTE_CRYPTO_AUTH_SHA1_HMAC`
- `RTE_CRYPTO_AUTH_SHA224`
- `RTE_CRYPTO_AUTH_SHA224_HMAC`

- RTE_CRYPTO_AUTH_SHA256
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_AES_GMAC

AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM

For supported feature flags please consult *Crypto Device Supported Functionality Matrices*.

## 13.2 Limitations

- Hardware only supports scenarios where ICV (digest buffer) is placed just after the authenticated data. Other placement will result in error.

## 13.3 Installation

MVSAM CRYPTO PMD driver compilation is disabled by default due to external dependencies. Currently there are two driver specific compilation options in config/common_base available:

- CONFIG_RTE_LIBRTE_PMD_MVSAM_CRYPTO (default: n)

  Toggle compilation of the librte_pmd_mvsam driver.

MVSAM CRYPTO PMD requires MUSDK built with EIP197 support thus following extra option must be passed to the library configuration script:

```
--enable-sam [--enable-sam-statistics] [--enable-sam-debug]
```

For instructions how to build required kernel modules please refer to *doc/musdk_get_started.txt*.

## 13.4 Initialization

After successfully building MVSAM CRYPTO PMD, the following modules need to be loaded:

```
insmod musdk_cma.ko
insmod crypto_safexcel.ko rings=0,0
insmod mv_sam_uio.ko
```

The following parameters (all optional) are exported by the driver:

- max_nb_queue_pairs: maximum number of queue pairs in the device (default: 8 - A8K, 4 - A7K/A3K).
- max_nb_sessions: maximum number of sessions that can be created (default: 2048).
- socket_id: socket on which to allocate the device resources on.

l2fwd-crypto example application can be used to verify MVSAM CRYPTO PMD operation:

```
./l2fwd-crypto --vdev=eth_mvpp2,iface=eth0 --vdev=crypto_mvsam -- \
  --cipher_op ENCRYPT --cipher_algo aes-cbc \
  --cipher_key 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f  \
  --auth_op GENERATE --auth_algo sha1-hmac \
  --auth_key 10:11:12:13:14:15:16:17:18:19:1a:1b:1c:1d:1e:1f
```

CHAPTER

FOURTEEN

# MARVELL NITROX CRYPTO POLL MODE DRIVER

The Nitrox crypto poll mode driver provides support for offloading cryptographic operations to the NITROX V security processor. Detailed information about the NITROX V security processor can be obtained here:

- https://www.marvell.com/security-solutions/nitrox-security-processors/nitrox-v/

## 14.1 Features

Nitrox crypto PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_AES_CBC`

Hash algorithms:

- `RTE_CRYPTO_AUTH_SHA1_HMAC`
- `RTE_CRYPTO_AUTH_SHA224_HMAC`
- `RTE_CRYPTO_AUTH_SHA256_HMAC`

## 14.2 Limitations

- AES_CBC Cipher Only combination is not supported.
- Session-less APIs are not supported.

## 14.3 Installation

For compiling the Nitrox crypto PMD, please check if the CONFIG_RTE_LIBRTE_PMD_NITROX setting is set to *y* in config/common_base file.

- `CONFIG_RTE_LIBRTE_PMD_NITROX=y`

## 14.4 Initialization

Nitrox crypto PMD depend on Nitrox kernel PF driver being installed on the platform. Nitrox PF driver is required to create VF devices which will be used by the PMD. Each VF device can enable one cryptodev PMD.

Nitrox kernel PF driver is available as part of CNN55XX-Driver SDK. The SDK and it's installation instructions can be obtained from: Marvell Technical Documentation Portal.

CHAPTER

FIFTEEN

# NULL CRYPTO POLL MODE DRIVER

The Null Crypto PMD (**librte_pmd_null_crypto**) provides a crypto poll mode driver which provides a minimal implementation for a software crypto device. As a null device it does not modify the data in the mbuf on which the crypto operation is to operate and it only has support for a single cipher and authentication algorithm.

When a burst of mbufs is submitted to a Null Crypto PMD for processing then each mbuf in the burst will be enqueued in an internal buffer for collection on a dequeue call as long as the mbuf has a valid rte_mbuf_offload operation with a valid rte_cryptodev_session or rte_crypto_xform chain of operations.

## 15.1 Features

Modes:

- RTE_CRYPTO_XFORM_CIPHER ONLY
- RTE_CRYPTO_XFORM_AUTH ONLY
- RTE_CRYPTO_XFORM_CIPHER THEN RTE_CRYPTO_XFORM_AUTH
- RTE_CRYPTO_XFORM_AUTH THEN RTE_CRYPTO_XFORM_CIPHER

Cipher algorithms:

- RTE_CRYPTO_CIPHER_NULL

Authentication algorithms:

- RTE_CRYPTO_AUTH_NULL

## 15.2 Limitations

- Only in-place is currently supported (destination address is the same as source address).

## 15.3 Installation

The Null Crypto PMD is enabled and built by default in both the Linux and FreeBSD builds.

## 15.4 Initialization

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_null") within the application.
- Use –vdev="crypto_null" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device (8 by default).
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).

Example:

```
./l2fwd-crypto -l 1 -n 4 --vdev="crypto_null,socket_id=0,max_nb_sessions=128" \
-- -p 1 --cdev SW --chain CIPHER_ONLY --cipher_algo "null"
```

CHAPTER

SIXTEEN

# CRYPTODEV SCHEDULER POLL MODE DRIVER LIBRARY

Scheduler PMD is a software crypto PMD, which has the capabilities of attaching hardware and/or software cryptodevs, and distributes ingress crypto ops among them in a certain manner.

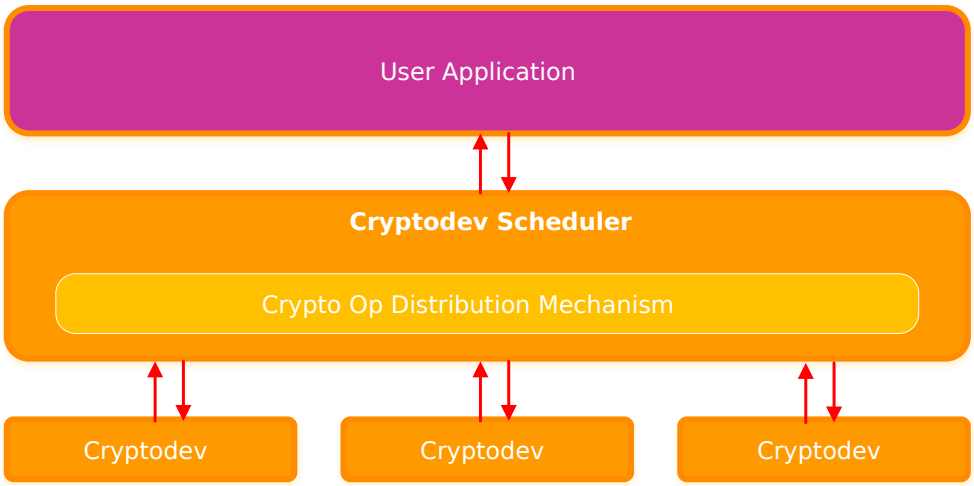



Fig. 16.1: Cryptodev Scheduler Overview

The Cryptodev Scheduler PMD library (**librte_pmd_crypto_scheduler**) acts as a software crypto PMD and shares the same API provided by librte_cryptodev. The PMD supports attaching multiple crypto PMDs, software or hardware, as slaves, and distributes the crypto workload to them with certain behavior. The behaviors are categorizes as different "modes". Basically, a scheduling mode defines certain actions for scheduling crypto ops to its slaves.

The librte_pmd_crypto_scheduler library exports a C API which provides an API for attaching/detaching slaves, set/get scheduling modes, and enable/disable crypto ops reordering.

## 16.1 Limitations

- Sessionless crypto operation is not supported
- OOP crypto operation is not supported when the crypto op reordering feature is enabled.

## 16.2 Installation

To build DPDK with CRYTPO_SCHEDULER_PMD the user is required to set CONFIG_RTE_LIBRTE_PMD_CRYPTO_SCHEDULER=y in config/common_base, and recompile DPDK

## 16.3 Initialization

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_scheduler") within the application.
- Use –vdev="crypto_scheduler" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_sessions: Specify the maximum number of sessions that can be created. This value may be overwritten internally if there are too many devices are attached.
- slave: If a cryptodev has been initialized with specific name, it can be attached to the scheduler using this parameter, simply filling the name here. Multiple cryptodevs can be attached initially by presenting this parameter multiple times.
- mode: Specify the scheduling mode of the PMD. The supported scheduling mode parameter values are specified in the "Cryptodev Scheduler Modes Overview" section.
- mode_param: Specify the mode-specific parameter. Some scheduling modes may be initialized with specific parameters other than the default ones, such as the **threshold** packet size of **packet-size-distr** mode. This parameter fulfills the purpose.
- ordering: Specify the status of the crypto operations ordering feature. The value of this parameter can be "enable" or "disable". This feature is disabled by default.

Example:

```
... --vdev "crypto_aesni_mb0,name=aesni_mb_1" --vdev "crypto_aesni_mb1,name=aesni_mb_2" --vdev
```

---

**Note:**

- The scheduler cryptodev cannot be started unless the scheduling mode is set and at least one slave is attached. Also, to configure the scheduler in the run-time, like attach/detach slave(s), change scheduling mode, or enable/disable crypto op ordering, one should stop the scheduler first, otherwise an error will be returned.
- The crypto op reordering feature requires using the userdata field of every mbuf to be processed to store temporary data. By the end of processing, the field is set to pointing to NULL, any previously stored value of this field will be lost.

---

## 16.4 Cryptodev Scheduler Modes Overview

Currently the Crypto Scheduler PMD library supports following modes of operation:

- **CDEV_SCHED_MODE_ROUNDROBIN:**

  *Initialization mode parameter*: **round-robin**

  Round-robin mode, which distributes the enqueued burst of crypto ops among its slaves in a round-robin manner. This mode may help to fill the throughput gap between the physical core and the existing cryptodevs to increase the overall performance.

- **CDEV_SCHED_MODE_PKT_SIZE_DISTR:**

  *Initialization mode parameter*: **packet-size-distr**

  Packet-size based distribution mode, which works with 2 slaves, the primary slave and the secondary slave, and distributes the enqueued crypto operations to them based on their data lengths. A crypto operation will be distributed to the primary slave if its data length is equal to or bigger than the designated threshold, otherwise it will be handled by the secondary slave.

  A typical usecase in this mode is with the QAT cryptodev as the primary and a software cryptodev as the secondary slave. This may help applications to process additional crypto workload than what the QAT cryptodev can handle on its own, by making use of the available CPU cycles to deal with smaller crypto workloads.

  The threshold is set to 128 bytes by default. It can be updated by calling function **rte_cryptodev_scheduler_option_set**. The parameter of **option_type** must be **CDEV_SCHED_OPTION_THRESHOLD** and **option** should point to a rte_cryptodev_scheduler_threshold_option structure filled with appropriate threshold value. Please NOTE this threshold has be a power-of-2 unsigned integer. It is possible to use **mode_param** initialization parameter to achieve the same purpose. For example:

  ... –vdev “crypto_scheduler,mode=packet-size-distr,mode_param=threshold:512” ...

  The above parameter will overwrite the threshold value to 512.

- **CDEV_SCHED_MODE_FAILOVER:**

  *Initialization mode parameter*: **fail-over**

  Fail-over mode, which works with 2 slaves, the primary slave and the secondary slave. In this mode, the scheduler will enqueue the incoming crypto operation burst to the primary slave. When one or more crypto operations fail to be enqueued, then they will be enqueued to the secondary slave.

- **CDEV_SCHED_MODE_MULTICORE:**

  *Initialization mode parameter*: **multi-core**

  Multi-core mode, which distributes the workload with several (up to eight) worker cores. The enqueued bursts are distributed among the worker cores in a round-robin manner. If scheduler cannot enqueue entire burst to the same worker, it will enqueue the remaining operations to the next available worker. For pure small packet size (64 bytes) traffic however the multi-core mode is not an optimal solution, as it doesn’t give significant per-core performance improvement. For mixed traffic (IMIX) the optimal number of worker cores is around 2-3. For large packets (1.5 kbytes) scheduler shows linear scaling in performance up to eight cores. Each worker uses its own slave cryptodev. Only software cryptodevs are supported. Only the same type of cryptodevs should be used concurrently.

  The multi-core mode uses one extra parameter:

- corelist: Semicolon-separated list of logical cores to be used as workers. The number of worker cores should be equal to the number of slave cryptodevs. These cores should be present in EAL core list parameter and should not be used by the application or any other process.

**Example:** ... –vdev "crypto_aesni_mb1,name=aesni_mb_1" –vdev "crypto_aesni_mb_pmd2,name=aesni_mb_2" –vdev "crypto_scheduler,slave=aesni_mb_1,slave=aesni_mb_2,mode=multi-core,corelist=23;24" ...

CHAPTER

SEVENTEEN

# SNOW 3G CRYPTO POLL MODE DRIVER

The SNOW 3G PMD (**librte_pmd_snow3g**) provides poll mode crypto driver support for utilizing Intel Libsso library, which implements F8 and F9 functions for SNOW 3G UEA2 cipher and UIA2 hash algorithms.

## 17.1 Features

SNOW 3G PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_CIPHER_SNOW3G_UEA2

Authentication algorithm:

- RTE_CRYPTO_AUTH_SNOW3G_UIA2

## 17.2 Limitations

- Chained mbufs are not supported.
- SNOW 3G (UIA2) supported only if hash offset field is byte-aligned.
- In-place bit-level operations for SNOW 3G (UEA2) are not supported (if length and/or offset of data to be ciphered is not byte-aligned).

## 17.3 Installation

To build DPDK with the SNOW3G_PMD the user is required to download the export controlled `libsso_snow3g` library, by registering in Intel Resource & Design Center. Once approval has been granted, the user needs to search for *Snow3G F8 F9 3GPP cryptographic algorithms Software Library* to download the library or directly through this link. After downloading the library, the user needs to unpack and compile it on their system before building DPDK:

```
make snow3G
```

**Note**: When encrypting with SNOW3G UEA2, by default the library encrypts blocks of 4 bytes, regardless the number of bytes to be encrypted provided (which leads to a possible buffer overflow). To avoid this situation, it is necessary not to pass 3GPP_SAFE_BUFFERS as a compilation flag. For this, in the Makefile of the library, make sure that this flag is commented out.:

```
#EXTRA_CFLAGS  += -D_3GPP_SAFE_BUFFERS
```

## 17.4 Initialization

In order to enable this virtual crypto PMD, user must:

- Export the environmental variable LIBSSO_SNOW3G_PATH with the path where the library was extracted (snow3g folder).
- Build the LIBSSO_SNOW3G library (explained in Installation section).
- Set CONFIG_RTE_LIBRTE_PMD_SNOW3G=y in config/common_base.

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_snow3g") within the application.
- Use –vdev="crypto_snow3g" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device (8 by default).
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).

Example:

```
./l2fwd-crypto -l 1 -n 4 --vdev="crypto_snow3g,socket_id=0,max_nb_sessions=128" \
-- -p 1 --cdev SW --chain CIPHER_ONLY --cipher_algo "snow3g-uea2"
```

CHAPTER

EIGHTEEN

# INTEL(R) QUICKASSIST (QAT) CRYPTO POLL MODE DRIVER

QAT documentation consists of three parts:

- Details of the symmetric and asymmetric crypto services below.
- Details of the compression service in the compressdev drivers section.
- Details of building the common QAT infrastructure and the PMDs to support the above services. See *Building PMDs on QAT* below.

## 18.1 Symmetric Crypto Service on QAT

The QAT symmetric crypto PMD (hereafter referred to as *QAT SYM [PMD]*) provides poll mode crypto driver support for the following hardware accelerator devices:

- `Intel QuickAssist Technology DH895xCC`
- `Intel QuickAssist Technology C62x`
- `Intel QuickAssist Technology C3xxx`
- `Intel QuickAssist Technology D15xx`
- `Intel QuickAssist Technology C4xxx`

### 18.1.1 Features

The QAT SYM PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_3DES_CBC`
- `RTE_CRYPTO_CIPHER_3DES_CTR`
- `RTE_CRYPTO_CIPHER_AES128_CBC`
- `RTE_CRYPTO_CIPHER_AES192_CBC`
- `RTE_CRYPTO_CIPHER_AES256_CBC`
- `RTE_CRYPTO_CIPHER_AES128_CTR`
- `RTE_CRYPTO_CIPHER_AES192_CTR`
- `RTE_CRYPTO_CIPHER_AES256_CTR`

- RTE_CRYPTO_CIPHER_AES_XTS
- RTE_CRYPTO_CIPHER_SNOW3G_UEA2
- RTE_CRYPTO_CIPHER_NULL
- RTE_CRYPTO_CIPHER_KASUMI_F8
- RTE_CRYPTO_CIPHER_DES_CBC
- RTE_CRYPTO_CIPHER_AES_DOCSISBPI
- RTE_CRYPTO_CIPHER_DES_DOCSISBPI
- RTE_CRYPTO_CIPHER_ZUC_EEA3

Hash algorithms:

- RTE_CRYPTO_AUTH_SHA1
- RTE_CRYPTO_AUTH_SHA1_HMAC
- RTE_CRYPTO_AUTH_SHA224
- RTE_CRYPTO_AUTH_SHA224_HMAC
- RTE_CRYPTO_AUTH_SHA256
- RTE_CRYPTO_AUTH_SHA256_HMAC
- RTE_CRYPTO_AUTH_SHA384
- RTE_CRYPTO_AUTH_SHA384_HMAC
- RTE_CRYPTO_AUTH_SHA512
- RTE_CRYPTO_AUTH_SHA512_HMAC
- RTE_CRYPTO_AUTH_AES_XCBC_MAC
- RTE_CRYPTO_AUTH_SNOW3G_UIA2
- RTE_CRYPTO_AUTH_MD5_HMAC
- RTE_CRYPTO_AUTH_NULL
- RTE_CRYPTO_AUTH_KASUMI_F9
- RTE_CRYPTO_AUTH_AES_GMAC
- RTE_CRYPTO_AUTH_ZUC_EIA3
- RTE_CRYPTO_AUTH_AES_CMAC

Supported AEAD algorithms:

- RTE_CRYPTO_AEAD_AES_GCM
- RTE_CRYPTO_AEAD_AES_CCM

### 18.1.2 Limitations

- Only supports the session-oriented API implementation (session-less APIs are not supported).
- SNOW 3G (UEA2), KASUMI (F8) and ZUC (EEA3) supported only if cipher length and offset fields are byte-multiple.

- SNOW 3G (UIA2) and ZUC (EIA3) supported only if hash length and offset fields are byte-multiple.
- No BSD support as BSD QAT kernel driver not available.
- ZUC EEA3/EIA3 is not supported by dh895xcc devices
- Maximum additional authenticated data (AAD) for GCM is 240 bytes long and must be passed to the device in a buffer rounded up to the nearest block-size multiple (x16) and padded with zeros.
- Queue pairs are not thread-safe (that is, within a single queue pair, RX and TX from different lcores is not supported).
- A GCM limitation exists, but only in the case where there are multiple generations of QAT devices on a single platform. To optimise performance, the GCM crypto session should be initialised for the device generation to which the ops will be enqueued. Specifically if a GCM session is initialised on a GEN2 device, but then attached to an op enqueued to a GEN3 device, it will work but cannot take advantage of hardware optimisations in the GEN3 device. And if a GCM session is initialised on a GEN3 device, then attached to an op sent to a GEN1/GEN2 device, it will not be enqueued to the device and will be marked as failed. The simplest way to mitigate this is to use the bdf whitelist to avoid mixing devices of different generations in the same process if planning to use for GCM.

### 18.1.3 Extra notes on KASUMI F9

When using KASUMI F9 authentication algorithm, the input buffer must be constructed according to the 3GPP KASUMI specification (section 4.4, page 13). The input buffer has to have COUNT (4 bytes), FRESH (4 bytes), MESSAGE and DIRECTION (1 bit) concatenated. After the DIRECTION bit, a single '1' bit is appended, followed by between 0 and 7 '0' bits, so that the total length of the buffer is multiple of 8 bits. Note that the actual message can be any length, specified in bits.

Once this buffer is passed this way, when creating the crypto operation, length of data to authenticate "op.sym.auth.data.length" must be the length of all the items described above, including the padding at the end. Also, offset of data to authenticate "op.sym.auth.data.offset" must be such that points at the start of the COUNT bytes.

## 18.2 Asymmetric Crypto Service on QAT

The QAT asymmetric crypto PMD (hereafter referred to as *QAT ASYM [PMD]*) provides poll mode crypto driver support for the following hardware accelerator devices:

- `Intel QuickAssist Technology DH895xCC`
- `Intel QuickAssist Technology C62x`
- `Intel QuickAssist Technology C3xxx`
- `Intel QuickAssist Technology D15xx`
- `Intel QuickAssist Technology C4xxx`

The QAT ASYM PMD has support for:

- `RTE_CRYPTO_ASYM_XFORM_MODEX`
- `RTE_CRYPTO_ASYM_XFORM_MODINV`

### 18.2.1 Limitations

- Big integers longer than 4096 bits are not supported.
- Queue pairs are not thread-safe (that is, within a single queue pair, RX and TX from different lcores is not supported).
- RSA-2560, RSA-3584 are not supported

## 18.3 Building PMDs on QAT

A QAT device can host multiple acceleration services:

- symmetric cryptography
- data compression
- asymmetric cryptography

These services are provided to DPDK applications via PMDs which register to implement the corresponding cryptodev and compressdev APIs. The PMDs use common QAT driver code which manages the QAT PCI device. They also depend on a QAT kernel driver being installed on the platform, see *Dependency on the QAT kernel driver* below.

### 18.3.1 Configuring and Building the DPDK QAT PMDs

Further information on configuring, building and installing DPDK is described here.

Quick instructions for QAT cryptodev PMD are as follows:

```
cd to the top-level DPDK directory
make defconfig
sed -i 's,\(CONFIG_RTE_LIBRTE_PMD_QAT_SYM\)=n,\1=y,' build/.config
or/and
sed -i 's,\(CONFIG_RTE_LIBRTE_PMD_QAT_ASYM\)=n,\1=y,' build/.config
make
```

Quick instructions for QAT compressdev PMD are as follows:

```
cd to the top-level DPDK directory
make defconfig
make
```

### 18.3.2 Build Configuration

These are the build configuration options affecting QAT, and their default values:

```
CONFIG_RTE_LIBRTE_PMD_QAT=y
CONFIG_RTE_LIBRTE_PMD_QAT_SYM=n
CONFIG_RTE_LIBRTE_PMD_QAT_ASYM=n
CONFIG_RTE_PMD_QAT_MAX_PCI_DEVICES=48
CONFIG_RTE_PMD_QAT_COMP_IM_BUFFER_SIZE=65536
```

CONFIG_RTE_LIBRTE_PMD_QAT must be enabled for any QAT PMD to be built.

Both QAT SYM PMD and QAT ASYM PMD have an external dependency on libcrypto, so are not built by default. CONFIG_RTE_LIBRTE_PMD_QAT_SYM/ASYM should be enabled to build them.

The QAT compressdev PMD has no external dependencies, so needs no configuration options and is built by default.

The number of VFs per PF varies - see table below. If multiple QAT packages are installed on a platform then CONFIG_RTE_PMD_QAT_MAX_PCI_DEVICES should be adjusted to the number of VFs which the QAT common code will need to handle.

---

**Note:** There are separate config items (not QAT-specific) for max cryptodevs CONFIG_RTE_CRYPTO_MAX_DEVS and max compressdevs CONFIG_RTE_COMPRESS_MAX_DEVS, if necessary these should be adjusted to handle the total of QAT and other devices which the process will use. In particular for crypto, where each QAT VF may expose two crypto devices, sym and asym, it may happen that the number of devices will be bigger than MAX_DEVS and the process will show an error during PMD initialisation. To avoid this problem CONFIG_RTE_CRYPTO_MAX_DEVS may be increased or -w, pci-whitelist domain:bus:devid:func option may be used.

---

QAT compression PMD needs intermediate buffers to support Deflate compression with Dynamic Huffman encoding. CONFIG_RTE_PMD_QAT_COMP_IM_BUFFER_SIZE specifies the size of a single buffer, the PMD will allocate a multiple of these, plus some extra space for associated meta-data. For GEN2 devices, 20 buffers are allocated while for GEN1 devices, 12 buffers are allocated, plus 1472 bytes overhead.

---

**Note:** If the compressed output of a Deflate operation using Dynamic Huffman Encoding is too big to fit in an intermediate buffer, then the operation will fall back to fixed compression rather than failing the operation. To avoid this less performant case, applications should configure the intermediate buffer size to be larger than the expected input data size (compressed output size is usually unknown, so the only option is to make larger than the input size).

---

### 18.3.3 Device and driver naming

- The qat cryptodev symmetric crypto driver name is "crypto_qat".
- The qat cryptodev asymmetric crypto driver name is "crypto_qat_asym".

The "rte_cryptodev_devices_get()" returns the devices exposed by either of these drivers.

- Each qat sym crypto device has a unique name, in format "<pci bdf>_<service>", e.g. "0000:41:01.0_qat_sym".
- Each qat asym crypto device has a unique name, in format "<pci bdf>_<service>", e.g. "0000:41:01.0_qat_asym". This name can be passed to "rte_cryptodev_get_dev_id()" to get the device_id.

---

**Note:** The cryptodev driver name is passed to the dpdk-test-crypto-perf tool in the "-devtype" parameter.

The qat crypto device name is in the format of the slave parameter passed to the crypto scheduler.

---

- The qat compressdev driver name is "compress_qat". The rte_compressdev_devices_get() returns the devices exposed by this driver.

- Each qat compression device has a unique name, in format <pci bdf>_<service>, e.g. “0000:41:01.0_qat_comp”. This name can be passed to rte_compressdev_get_dev_id() to get the device_id.

### 18.3.4 Dependency on the QAT kernel driver

To use QAT an SRIOV-enabled QAT kernel driver is required. The VF devices created and initialised by this driver will be used by the QAT PMDs.

Instructions for installation are below, but first an explanation of the relationships between the PF/VF devices and the PMDs visible to DPDK applications.

Each QuickAssist PF device exposes a number of VF devices. Each VF device can enable one symmetric cryptodev PMD and/or one asymmetric cryptodev PMD and/or one compressdev PMD. These QAT PMDs share the same underlying device and pci-mgmt code, but are enumerated independently on their respective APIs and appear as independent devices to applications.

---

**Note:** Each VF can only be used by one DPDK process. It is not possible to share the same VF across multiple processes, even if these processes are using different acceleration services.

Conversely one DPDK process can use one or more QAT VFs and can expose both cryptodev and compressdev instances on each of those VFs.

---

### 18.3.5 Available kernel drivers

Kernel drivers for each device for each service are listed in the following table. (Scroll right to see the full table)

Table 18.1: QAT device generations, devices and drivers

| S | A | C | Gen | Device | Driver/ver | Kernel Module | Pci Driver | PF Did | #PFs | VF Did | VFs/PF |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Yes | No | No | 1 | DH895xCC | linux/4.4+ | qat_dh895xcc | dh895xcc | 435 | 1 | 443 | 32 |
| Yes | Yes | No | “ | “ | 01.org/4.2.0+ | “ | “ | “ | “ | “ | “ |
| Yes | Yes | Yes | “ | “ | 01.org/4.3.0+ | “ | “ | “ | “ | “ | “ |
| Yes | No | No | 2 | C62x | linux/4.5+ | qat_c62x | c6xx | 37c8 | 3 | 37c9 | 16 |
| Yes | Yes | Yes | “ | “ | 01.org/4.2.0+ | “ | “ | “ | “ | “ | “ |
| Yes | No | No | 2 | C3xxx | linux/4.5+ | qat_c3xxx | c3xxx | 19e2 | 1 | 19e3 | 16 |
| Yes | Yes | Yes | “ | “ | 01.org/4.2.0+ | “ | “ | “ | “ | “ | “ |
| Yes | No | No | 2 | D15xx | p | qat_d15xx | d15xx | 6f54 | 1 | 6f55 | 16 |
| Yes | No | No | 3 | C4xxx | p | qat_c4xxx | c4xxx | 18a0 | 1 | 18a1 | 128 |

The first 3 columns indicate the service:

- S = Symmetric crypto service (via cryptodev API)
- A = Asymmetric crypto service (via cryptodev API)
- C = Compression service (via compressdev API)

The `Driver` column indicates either the Linux kernel version in which support for this device was introduced or a driver available on Intel's 01.org website. There are both linux in-tree and 01.org kernel drivers available for some devices. p = release pending.

If you are running on a kernel which includes a driver for your device, see *Installation using kernel.org driver* below. Otherwise see *Installation using 01.org QAT driver*.

### 18.3.6 Installation using kernel.org driver

The examples below are based on the C62x device, if you have a different device use the corresponding values in the above table.

In BIOS ensure that SRIOV is enabled and either:

- Disable VT-d or
- Enable VT-d and set `"intel_iommu=on iommu=pt"` in the grub file.

Check that the QAT driver is loaded on your system, by executing:

```
lsmod | grep qa
```

You should see the kernel module for your device listed, e.g.:

```
qat_c62x               5626  0
intel_qat              82336  1 qat_c62x
```

Next, you need to expose the Virtual Functions (VFs) using the sysfs file system.

First find the BDFs (Bus-Device-Function) of the physical functions (PFs) of your device, e.g.:

```
lspci -d:37c8
```

You should see output similar to:

```
1a:00.0 Co-processor: Intel Corporation Device 37c8
3d:00.0 Co-processor: Intel Corporation Device 37c8
3f:00.0 Co-processor: Intel Corporation Device 37c8
```

Enable the VFs for each PF by echoing the number of VFs per PF to the pci driver:

```
echo 16 > /sys/bus/pci/drivers/c6xx/0000:1a:00.0/sriov_numvfs
echo 16 > /sys/bus/pci/drivers/c6xx/0000:3d:00.0/sriov_numvfs
echo 16 > /sys/bus/pci/drivers/c6xx/0000:3f:00.0/sriov_numvfs
```

Check that the VFs are available for use. For example `lspci -d:37c9` should list 48 VF devices available for a `C62x` device.

To complete the installation follow the instructions in *Binding the available VFs to the DPDK UIO driver*.

---

**Note:** If the QAT kernel modules are not loaded and you see an error like `Failed to load MMP firmware qat_895xcc_mmp.bin` in kernel logs, this may be as a result of not using a distribution, but just updating the kernel directly.

Download firmware from the kernel firmware repo.

Copy qat binaries to `/lib/firmware`:

```
cp qat_895xcc.bin /lib/firmware
cp qat_895xcc_mmp.bin /lib/firmware
```

Change to your linux source root directory and start the qat kernel modules:

```
insmod ./drivers/crypto/qat/qat_common/intel_qat.ko
insmod ./drivers/crypto/qat/qat_dh895xcc/qat_dh895xcc.ko
```

**Note:** If you see the following warning in `/var/log/messages` it can be ignored: `IOMMU should be enabled for SR-IOV to work correctly`.

### 18.3.7 Installation using 01.org QAT driver

Download the latest QuickAssist Technology Driver from 01.org. Consult the *Getting Started Guide* at the same URL for further information.

The steps below assume you are:

- Building on a platform with one `C62x` device.
- Using package `qat1.7.l.4.2.0-000xx.tar.gz`.
- On Fedora26 kernel `4.11.11-300.fc26.x86_64`.

In the BIOS ensure that SRIOV is enabled and VT-d is disabled.

Uninstall any existing QAT driver, for example by running:

- `./installer.sh uninstall` in the directory where originally installed.

Build and install the SRIOV-enabled QAT driver:

```
mkdir /QAT
cd /QAT

# Copy the package to this location and unpack
tar zxof qat1.7.l.4.2.0-000xx.tar.gz

./configure --enable-icp-sriov=host
make install
```

You can use `cat /sys/kernel/debug/qat<your device type and bdf>/version/fw` to confirm the driver is correctly installed and is using firmware version 4.2.0. You can use `lspci -d:37c9` to confirm the presence of the 16 VF devices available per `C62x` PF.

Confirm the driver is correctly installed and is using firmware version 4.2.0:

```
cat /sys/kernel/debug/qat<your device type and bdf>/version/fw
```

Confirm the presence of 48 VF devices - 16 per PF:

```
lspci -d:37c9
```

To complete the installation - follow instructions in *Binding the available VFs to the DPDK UIO driver*.

**Note:** If using a later kernel and the build fails with an error relating to `strict_stroul` not being available apply the following patch:

```
/QAT/QAT1.6/quickassist/utilities/downloader/Target_CoreLibs/uclo/include/linux/uclo_platform.h
+ #if LINUX_VERSION_CODE >= KERNEL_VERSION(3,18,5)
+ #define STR_TO_64(str, base, num, endPtr) {endPtr=NULL; if (kstrtoul((str), (base), (num))) p
+ #else
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,38)
```

```
#define STR_TO_64(str, base, num, endPtr) {endPtr=NULL; if (strict_strtoull((str), (base), (num
#else
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,25)
#define STR_TO_64(str, base, num, endPtr) {endPtr=NULL; strict_strtoll((str), (base), (num));}
#else
#define STR_TO_64(str, base, num, endPtr)                                   \
     do {                                                                   \
           if (str[0] == '-')                                               \
           {                                                                \
                *(num) = -(simple_strtoull((str+1), &(endPtr), (base)));    \
           }else {                                                          \
                *(num) = simple_strtoull((str), &(endPtr), (base));         \
           }                                                                \
     } while(0)
+ #endif
#endif
#endif
```

**Note:** If the build fails due to missing header files you may need to do following:

```
sudo yum install zlib-devel
sudo yum install openssl-devel
sudo yum install libudev-devel
```

**Note:** If the build or install fails due to mismatching kernel sources you may need to do the following:

```
sudo yum install kernel-headers-`uname -r`
sudo yum install kernel-src-`uname -r`
sudo yum install kernel-devel-`uname -r`
```

### 18.3.8 Binding the available VFs to the DPDK UIO driver

Unbind the VFs from the stock driver so they can be bound to the uio driver.

#### For an Intel(R) QuickAssist Technology DH895xCC device

The unbind command below assumes `BDFs` of `03:01.00-03:04.07`, if your VFs are different adjust the unbind command below:

```
for device in $(seq 1 4); do \
    for fn in $(seq 0 7); do \
        echo -n 0000:03:0${device}.${fn} > \
        /sys/bus/pci/devices/0000\:03\:0${device}.${fn}/driver/unbind; \
    done; \
done
```

#### For an Intel(R) QuickAssist Technology C62x device

The unbind command below assumes `BDFs` of `1a:01.00-1a:02.07`, `3d:01.00-3d:02.07` and `3f:01.00-3f:02.07`, if your VFs are different adjust the unbind command below:

```
for device in $(seq 1 2); do \
    for fn in $(seq 0 7); do \
```

```
        echo -n 0000:1a:0${device}.${fn} > \
        /sys/bus/pci/devices/0000\:1a\:0${device}.${fn}/driver/unbind; \

        echo -n 0000:3d:0${device}.${fn} > \
        /sys/bus/pci/devices/0000\:3d\:0${device}.${fn}/driver/unbind; \

        echo -n 0000:3f:0${device}.${fn} > \
        /sys/bus/pci/devices/0000\:3f\:0${device}.${fn}/driver/unbind; \
    done; \
done
```

#### For Intel(R) QuickAssist Technology C3xxx or D15xx device

The unbind command below assumes `BDFs` of `01:01.00-01:02.07`, if your VFs are different adjust the unbind command below:

```
for device in $(seq 1 2); do \
    for fn in $(seq 0 7); do \
        echo -n 0000:01:0${device}.${fn} > \
        /sys/bus/pci/devices/0000\:01\:0${device}.${fn}/driver/unbind; \
    done; \
done
```

#### Bind to the DPDK uio driver

Install the DPDK igb_uio driver, bind the VF PCI Device id to it and use lspci to confirm the VF devices are now in use by igb_uio kernel driver, e.g. for the C62x device:

```
cd to the top-level DPDK directory
modprobe uio
insmod ./build/kmod/igb_uio.ko
echo "8086 37c9" > /sys/bus/pci/drivers/igb_uio/new_id
lspci -vvd:37c9
```

Another way to bind the VFs to the DPDK UIO driver is by using the `dpdk-devbind.py` script:

```
cd to the top-level DPDK directory
./usertools/dpdk-devbind.py -b igb_uio 0000:03:01.1
```

### 18.3.9 Testing

QAT SYM crypto PMD can be tested by running the test application:

```
make defconfig
make -j
cd ./build/app
./test -l1 -n1 -w <your qat bdf>
RTE>>cryptodev_qat_autotest
```

QAT ASYM crypto PMD can be tested by running the test application:

```
make defconfig
make -j
cd ./build/app
./test -l1 -n1 -w <your qat bdf>
RTE>>cryptodev_qat_asym_autotest
```

QAT compression PMD can be tested by running the test application:

```
make defconfig
sed -i 's,\(CONFIG_RTE_COMPRESSDEV_TEST\)=n,\1=y,' build/.config
make -j
cd ./build/app
./test -l1 -n1 -w <your qat bdf>
RTE>>compressdev_autotest
```

### 18.3.10 Debugging

There are 2 sets of trace available via the dynamic logging feature:

- pmd.qat_dp exposes trace on the data-path.
- pmd.qat_general exposes all other trace.

pmd.qat exposes both sets of traces. They can be enabled using the log-level option (where 8=maximum log level) on the process cmdline, e.g. using any of the following:

```
--log-level="pmd.qat_general,8"
--log-level="pmd.qat_dp,8"
--log-level="pmd.qat,8"
```

---

**Note:** The global RTE_LOG_DP_LEVEL overrides data-path trace so must be set to RTE_LOG_DEBUG to see all the trace. This variable is in config/rte_config.h for meson build and config/common_base for gnu make. Also the dynamic global log level overrides both sets of trace, so e.g. no QAT trace would display in this case:

```
--log-level="7" --log-level="pmd.qat_general,8"
```

---

CHAPTER

NINETEEN

# VIRTIO CRYPTO POLL MODE DRIVER

The virtio crypto PMD provides poll mode driver support for the virtio crypto device.

## 19.1 Features

The virtio crypto PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_CIPHER_AES_CBC`

Hash algorithms:

- `RTE_CRYPTO_AUTH_SHA1_HMAC`

## 19.2 Limitations

- Only supports the session-oriented API implementation (session-less APIs are not supported).
- Only supports modern mode since virtio crypto conforms to virtio-1.0.
- Only has two types of queues: data queue and control queue. These two queues only support indirect buffers to communication with the virtio backend.
- Only supports AES_CBC cipher only algorithm and AES_CBC with HMAC_SHA1 chaining algorithm since the vhost crypto backend only these algorithms are supported.
- Does not support Link State interrupt.
- Does not support runtime configuration.

## 19.3 Virtio crypto PMD Rx/Tx Callbacks

Rx callbacks:

- `virtio_crypto_pkt_rx_burst`

Tx callbacks:

- `virtio_crypto_pkt_tx_burst`

## 19.4 Installation

Quick instructions are as follows:

Firstly run DPDK vhost crypto sample as a server side and build QEMU with vhost crypto enabled. QEMU can then be started using the following parameters:

```
qemu-system-x86_64 \
[...] \
    -chardev socket,id=charcrypto0,path=/path/to/your/socket \
    -object cryptodev-vhost-user,id=cryptodev0,chardev=charcrypto0 \
    -device virtio-crypto-pci,id=crypto0,cryptodev=cryptodev0
[...]
```

Secondly bind the uio_generic driver for the virtio-crypto device. For example, 0000:00:04.0 is the domain, bus, device and function number of the virtio-crypto device:

```
modprobe uio_pci_generic
echo -n 0000:00:04.0 > /sys/bus/pci/drivers/virtio-pci/unbind
echo "1af4 1054" > /sys/bus/pci/drivers/uio_pci_generic/new_id
```

Finally the front-end virtio crypto PMD driver can be installed:

```
cd to the top-level DPDK directory
sed -i 's,\(CONFIG_RTE_LIBRTE_PMD_VIRTIO_CRYPTO\)=n,\1=y,' config/common_base
make config T=x86_64-native-linux-gcc
make install T=x86_64-native-linux-gcc
```

## 19.5 Tests

The unit test cases can be tested as below:

```
reserve enough huge pages
cd to the top-level DPDK directory
export RTE_TARGET=x86_64-native-linux-gcc
export RTE_SDK=`pwd`
cd to app/test
type the command "make" to compile
run the tests with "./test"
type the command "cryptodev_virtio_autotest" to test
```

The performance can be tested as below:

```
reserve enough huge pages
cd to the top-level DPDK directory
export RTE_TARGET=x86_64-native-linux-gcc
export RTE_SDK=`pwd`
cd to app/test-crypto-perf
type the command "make" to compile
run the tests with the following command:

./dpdk-test-crypto-perf -l 0,1 -- --devtype crypto_virtio \
    --ptest throughput --optype cipher-then-auth --cipher-algo aes-cbc \
    --cipher-op encrypt --cipher-key-sz 16 --auth-algo sha1-hmac \
    --auth-op generate --auth-key-sz 64 --digest-sz 12 \
    --total-ops 100000000 --burst-sz 64 --buffer-sz 2048
```

CHAPTER

TWENTY

# ZUC CRYPTO POLL MODE DRIVER

The ZUC PMD (**librte_pmd_zuc**) provides poll mode crypto driver support for utilizing Intel Libsso library, which implements F8 and F9 functions for ZUC EEA3 cipher and EIA3 hash algorithms.

## 20.1 Features

ZUC PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_CIPHER_ZUC_EEA3

Authentication algorithm:

- RTE_CRYPTO_AUTH_ZUC_EIA3

## 20.2 Limitations

- Chained mbufs are not supported.
- ZUC (EIA3) supported only if hash offset field is byte-aligned.
- ZUC (EEA3) supported only if cipher length, cipher offset fields are byte-aligned.
- ZUC PMD cannot be built as a shared library, due to limitations in the underlying library.

## 20.3 Installation

To build DPDK with the ZUC_PMD the user is required to download the export controlled `libsso_zuc` library, by registering in Intel Resource & Design Center. Once approval has been granted, the user needs to search for *ZUC 128-EAA3 and 128-EIA3 3GPP cryptographic algorithms Software Library* to download the library or directly through this link. After downloading the library, the user needs to unpack and compile it on their system before building DPDK:

```
make
```

## 20.4 Initialization

In order to enable this virtual crypto PMD, user must:

- Export the environmental variable LIBSSO_ZUC_PATH with the path where the library was extracted (zuc folder).
- Export the environmental variable LD_LIBRARY_PATH with the path where the built libsso library is (LIBSSO_ZUC_PATH/build).
- Build the LIBSSO_ZUC library (explained in Installation section).
- Build DPDK as follows:

```
make config T=x86_64-native-linux-gcc
sed -i 's,\(CONFIG_RTE_LIBRTE_PMD_ZUC\)=n,\1=y,' build/.config
make
```

To use the PMD in an application, user must:

- Call rte_vdev_init("crypto_zuc") within the application.
- Use –vdev="crypto_zuc" in the EAL options, which will call rte_vdev_init() internally.

The following parameters (all optional) can be provided in the previous two calls:

- socket_id: Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).
- max_nb_queue_pairs: Specify the maximum number of queue pairs in the device (8 by default).
- max_nb_sessions: Specify the maximum number of sessions that can be created (2048 by default).

Example:

```
./l2fwd-crypto -l 1 -n 4 --vdev="crypto_zuc,socket_id=0,max_nb_sessions=128" \
-- -p 1 --cdev SW --chain CIPHER_ONLY --cipher_algo "zuc-eea3"
```