Crypto Device Drivers

Release 17.02.1

June 02, 2017

CONTENTS

CHAPTER

ONE

CRYPTO DEVICE SUPPORTED FUNCTIONALITY MATRICES

Supported Feature Flags

Feature Flags	qat	null	aesni_mb	aesni_gcm	snow3g	kasumi	zuc	armv8
RTE_CRYPTODEV_FF_SYMMETRIC_CRYPTO	x	x	x	x	x	x	x	x
RTE_CRYPTODEV_FF_ASYMMETRIC_CRYPTO								
RTE_CRYPTODEV_FF_SYM_OPERATION_CHAINING	x	x	x	x	x	x	x	x
RTE_CRYPTODEV_FF_CPU_SSE			x		x	x		
RTE_CRYPTODEV_FF_CPU_AVX			x		x	x		
RTE_CRYPTODEV_FF_CPU_AVX2			x					
RTE_CRYPTODEV_FF_CPU_AVX512			x					
RTE_CRYPTODEV_FF_CPU_AESNI			x	x				
RTE_CRYPTODEV_FF_HW_ACCELERATED	x							
RTE_CRYPTODEV_FF_CPU_NEON								x
RTE_CRYPTODEV_FF_CPU_ARM_CE								x

Supported Cipher Algorithms

Cipher Algorithms	qat	null	aesni_mb	aesni_gcm	snow3g	kasumi	zuc	armv8
NULL		x						
AES_CBC_128	x		x					x
AES_CBC_192	x		x					
AES_CBC_256	x		x					
AES_CTR_128	x		x					
AES_CTR_192	x		x					
AES_CTR_256	x		x					
DES_CBC	x							
SNOW3G_UEA2	x				x			
KASUMI_F8						x		
ZUC_EEA3							x	

Supported Authentication Algorithms

Cipher Algorithms	qat	null	aesni_mb	aesni_gcm	snow3g	kasumi	zuc	armv8
NONE		x						
MD5								
MD5_HMAC			x					
SHA1								
SHA1_HMAC	x		x					x
SHA224								
SHA224_HMAC			x					
SHA256								
SHA256_HMAC	x		x					x
SHA384								
SHA384_HMAC			x					
SHA512								
SHA512_HMAC	x		x					
AES_XCBC	x		x					
AES_GMAC				x				
SNOW3G_UIA2	x				x			
KASUMI_F9						x		
ZUC_EIA3							x	

Supported AEAD Algorithms

AEAD Algorithms	qat	null	aesni_mb	aesni_gcm	snow3g	kasumi	zuc	armv8
AES_GCM_128	x			x				
AES_GCM_192	x							
AES_GCM_256	x			x				

CHAPTER

TWO

AESN-NI MULTI BUFFER CRYTPO 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

Hash algorithms:

- 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

2.2 Limitations

- Chained mbufs are not supported.
- Only in-place is currently supported (destination address is the same as source address).
- Only supports session-oriented API implementation (session-less APIs 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 v0.44, which can be downloaded in https://github.com/01org/intel-ipsec-mb/archive/v0.44.zip.

```
make
```

2.4 Initialization

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

- Export the environmental variable AESNI_MULTI_BUFFER_LIB_PATH with the path where the library was extracted.
- 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_eal_vdev_init("crypto_aesni_mb") within the application.
- Use –vdev="crypto_aesni_mb" in the EAL options, which will call rte_eal_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 -c 40 -n 4 --vdev="crypto_aesni_mb,socket_id=1,max_nb_sessions=128"
```

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 ISA-L crypto library, which provides operation acceleration through the AES-NI instruction sets for AES-GCM authenticated cipher algorithm.

3.1 Features

AESNI GCM PMD has support for:

Cipher algorithms:

- RTE_CRYPTO_CIPHER_AES_GCM

Authentication algorithms:

- RTE_CRYPTO_AUTH_AES_GCM
- RTE_CRYPTO_AUTH_AES_GMAC

3.2 Installation

To build DPDK with the AESNI_GCM_PMD the user is required to install the `libisal_crypto` library in the build environment. For download and more details please visit https://github.com/01org/isa-l_crypto.

3.3 Initialization

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

- Install the ISA-L crypto 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_eal_vdev_init("crypto_aesni_gcm") within the application.
- Use –vdev="crypto_aesni_gcm" in the EAL options, which will call rte_eal_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 -c 40 -n 4 --vdev="crypto_aesni_gcm,socket_id=1,max_nb_sessions=128"
```

3.4 Limitations

- Chained mbufs are supported but only out-of-place (destination mbuf must be contiguous).
- Hash only is not supported.
- Cipher only is not supported.

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-linuxapp-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. For performance test cryptodev_sw_armv8_perftest can be used.

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

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.

5.1 Features

KASUMI PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_CIPHER_KASUMI_F8

Authentication algorithm:

- RTE_CRYPTO_AUTH_KASUMI_F9

5.2 Limitations

- Chained mbufs are not supported.
- KASUMI(F9) supported only if hash offset 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).

5.3 Installation

To build DPDK with the KASUMI_PMD the user is required to download the export controlled `libsso_kasumi` library, by requesting it from https://networkbuilders.intel.com/network-technologies/dpdk. Once approval has been granted, the user needs to log in https://networkbuilders.intel.com/dpdklogin and click on “Kasumi Bit Stream crypto library” link, to download the library. After downloading the library, the user needs to unpack and compile it on their system before building DPDK:

```
make
```

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

```
make KASUMI_CFLAGS=-DKASUMI_C
```

5.4 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_eal_vdev_init("crypto_kasumi") within the application.
- Use –vdev="crypto_kasumi" in the EAL options, which will call rte_eal_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 -c 40 -n 4 --vdev="crypto_kasumi,socket_id=1,max_nb_sessions=128"
```

CHAPTER

SIX

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/

6.1 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_AES_GCM`

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`

6.2 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-linuxapp-gcc target)

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

6.3 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 performance test cryptodev_openssl_perftest can be used.

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

```
sudo ./build/l2fwd-crypto -c 0x3 -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
:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11
```

6.4 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

SEVEN

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.

7.1 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

7.2 Limitations

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

7.3 Installation

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

7.4 Initialization

To use the PMD in an application, user must:

- Call rte_eal_vdev_init("crypto_null") within the application.
- Use –vdev="crypto_null" in the EAL options, which will call rte_eal_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 -c 40 -n 4 --vdev="crypto_null,socket_id=1,max_nb_sessions=128"
```

CHAPTER

EIGHT

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.

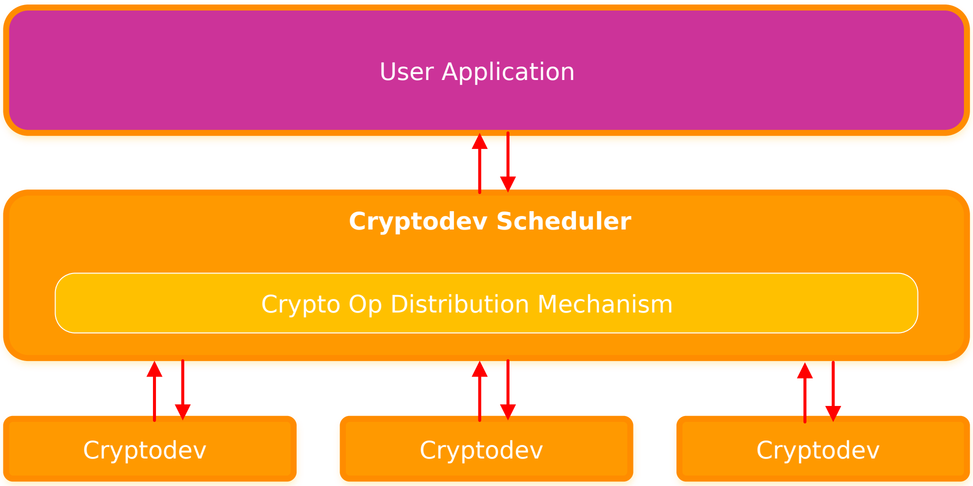

Fig. 8.1: Cryptodev Scheduler Overview

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.

8.1 Limitations

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

8.2 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

8.3 Initialization

To use the PMD in an application, user must:

- Call rte_eal_vdev_init("crpyto_scheduler") within the application.
- Use –vdev="crpyto_scheduler" in the EAL options, which will call rte_eal_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.

Example:

```
... --vdev "crypto_aesni_mb_pmd,name=aesni_mb_1" --vdev "crypto_aesni_mb_pmd,name=aesni_mb_2" -
```

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.

8.4 Cryptodev Scheduler Modes Overview

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

- **CDEV_SCHED_MODE_ROUNDROBIN:**

  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.

CHAPTER

NINE

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.

9.1 Features

SNOW 3G PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_CIPHER_SNOW3G_UEA2

Authentication algorithm:

- RTE_CRYPTO_AUTH_SNOW3G_UIA2

9.2 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).

9.3 Installation

To build DPDK with the SNOW3G_PMD the user is required to download the export controlled `libsso_snow3g` library, by requesting it from https://networkbuilders.intel.com/network-technologies/dpdk. Once approval has been granted, the user needs to log in https://networkbuilders.intel.com/dpdklogin and click on "Snow3G Bit Stream crypto library" link, to download the library. After downloading the library, the user needs to unpack and compile it on their system before building DPDK:

```
make snow3G
```

9.4 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_eal_vdev_init("crypto_snow3g") within the application.
- Use –vdev="crypto_snow3g" in the EAL options, which will call rte_eal_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 -c 40 -n 4 --vdev="crypto_snow3g,socket_id=1,max_nb_sessions=128"
```

CHAPTER

TEN

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

The QAT PMD provides poll mode crypto driver support for **Intel QuickAssist Technology DH895xxC**, **Intel QuickAssist Technology C62x** and **Intel QuickAssist Technology C3xxx** hardware accelerator.

10.1 Features

The QAT 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_SNOW3G_UEA2`
- `RTE_CRYPTO_CIPHER_AES_GCM`
- `RTE_CRYPTO_CIPHER_NULL`
- `RTE_CRYPTO_CIPHER_KASUMI_F8`
- `RTE_CRYPTO_CIPHER_DES_CBC`

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_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`

10.2 Limitations

- Hash only is not supported except SNOW 3G UIA2 and KASUMI F9.
- Only supports the session-oriented API implementation (session-less APIs are not supported).
- SNOW 3G (UEA2) and KASUMI (F8) supported only if cipher length, cipher offset fields are byte-aligned.
- SNOW 3G (UIA2) and KASUMI (F9) supported only if hash length, hash offset fields are byte-aligned.
- No BSD support as BSD QAT kernel driver not available.

10.3 Installation

To use the DPDK QAT PMD an SRIOV-enabled QAT kernel driver is required. The VF devices exposed by this driver will be used by QAT PMD.

To enable QAT in DPDK, follow the instructions mentioned in http://dpdk.org/doc/guides/linux_gsg/build_dpdk.html

Quick instructions as follows:

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

If you are running on kernel 4.4 or greater, see instructions for *Installation using kernel.org driver* below. If you are on a kernel earlier than 4.4, see *Installation using 01.org QAT driver*.

For **Intel QuickAssist Technology C62x** and **Intel QuickAssist Technology C3xxx** device, kernel 4.5 or greater is needed. See instructions for *Installation using kernel.org driver* below.

10.4 Installation using 01.org QAT driver

NOTE: There is no driver available for **Intel QuickAssist Technology C62x** and **Intel QuickAssist Technology C3xxx** devices on 01.org.

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 `DH895xCC` device.
- Using package `qatmux.l.2.3.0-34.tgz`.
- On Fedora21 kernel `3.17.4-301.fc21.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.
- or `rmmod qat_dh895xcc; rmmod intel_qat`.

Build and install the SRIOV-enabled QAT driver:

```
mkdir /QAT
cd /QAT
# copy qatmux.l.2.3.0-34.tgz to this location
tar zxof qatmux.l.2.3.0-34.tgz

export ICP_WITHOUT_IOMMU=1
./installer.sh install QAT1.6 host
```

You can use `cat /proc/icp_dh895xcc_dev0/version` to confirm the driver is correctly installed. You can use `lspci -d:443` to confirm the bdf of the 32 VF devices are available per `DH895xCC` device.

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
```

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`

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``

10.5 Installation using kernel.org driver

For **Intel QuickAssist Technology DH895xxC**:

Assuming you are running on at least a 4.4 kernel, you can use the stock kernel.org QAT driver to start the QAT hardware.

The steps below assume you are:

- Running DPDK on a platform with one `DH895xCC` device.
- On a kernel at least version 4.4.

In BIOS ensure that SRIOV is enabled and either a) disable VT-d or b) enable VT-d and set `"intel_iommu=on iommu=pt"` in the grub file.

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

```
lsmod | grep qat
```

You should see the following output:

```
qat_dh895xcc            5626  0
intel_qat              82336  1 qat_dh895xcc
```

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

First find the bdf of the physical function (PF) of the DH895xCC device:

```
lspci -d : 435
```

You should see output similar to:

```
03:00.0 Co-processor: Intel Corporation Coleto Creek PCIe Endpoint
```

Using the sysfs, enable the VFs:

```
echo 32 > /sys/bus/pci/drivers/dh895xcc/0000\:03\:00.0/sriov_numvfs
```

If you get an error, it's likely you're using a QAT kernel driver earlier than kernel 4.4.

To verify that the VFs are available for use - use `lspci -d:443` to confirm the bdf of the 32 VF devices are available per `DH895xCC` device.

To complete the installation - follow 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` this may be as a result of not using a distribution, but just updating the kernel directly.

Download firmware from the kernel firmware repo at: http://git.kernel.org/cgit/linux/kernel/git/firmware/linux-firmware.git/tree/

Copy qat binaries to /lib/firmware: * `cp qat_895xcc.bin /lib/firmware` * `cp qat_895xcc_mmp.bin /lib/firmware` cd 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:The following warning in /var/log/messages can be ignored: `IOMMU should be enabled for SR-IOV to work correctly`

For **Intel QuickAssist Technology C62x**: Assuming you are running on at least a 4.5 kernel, you can use the stock kernel.org QAT driver to start the QAT hardware.

The steps below assume you are:

- Running DPDK on a platform with one `C62x` device.
- On a kernel at least version 4.5.

In BIOS ensure that SRIOV is enabled and either a) disable VT-d or b) enable VT-d and set `"intel_iommu=on iommu=pt"` in the grub file.

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

```
lsmod | grep qat
```

You should see the following output:

```
qat_c62x               16384  0
intel_qat             122880  1 qat_c62x
```

Next, you need to expose the VFs using the sysfs file system.

First find the bdf of the C62x device:

```
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
```

For each c62x device there are 3 PFs. Using the sysfs, for each PF, enable the 16 VFs:

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

If you get an error, it's likely you're using a QAT kernel driver earlier than kernel 4.5.

To verify that the VFs are available for use - use `lspci -d:37c9` to confirm the bdf of the 48 VF devices are available per `C62x` device.

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

For **Intel QuickAssist Technology C3xxx**: Assuming you are running on at least a 4.5 kernel, you can use the stock kernel.org QAT driver to start the QAT hardware.

The steps below assume you are:

- Running DPDK on a platform with one `C3xxx` device.
- On a kernel at least version 4.5.

In BIOS ensure that SRIOV is enabled and either a) disable VT-d or b) enable VT-d and set `"intel_iommu=on iommu=pt"` in the grub file.

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

```
lsmod | grep qat
```

You should see the following output:

```
qat_c3xxx               16384  0
intel_qat              122880  1 qat_c3xxx
```

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

First find the bdf of the physical function (PF) of the C3xxx device

lspci -d:19e2

You should see output similar to:

```
01:00.0 Co-processor: Intel Corporation Device 19e2
```

For c3xxx device there is 1 PFs. Using the sysfs, enable the 16 VFs:

```
echo 16 > /sys/bus/pci/drivers/c3xxx/0000\:01\:00.0/sriov_numvfs
```

If you get an error, it's likely you're using a QAT kernel driver earlier than kernel 4.5.

To verify that the VFs are available for use - use `lspci -d:19e3` to confirm the bdf of the 16 VF devices are available per `C3xxx` device. To complete the installation - follow instructions in *Binding the available VFs to the DPDK UIO driver*.

10.6 Binding the available VFs to the DPDK UIO driver

For **Intel(R) QuickAssist Technology DH895xcc** device: The unbind command below assumes `bdfs` of `03:01.00-03:04.07`, if yours are different adjust the unbind command below:

```
cd $RTE_SDK
modprobe uio
insmod ./build/kmod/igb_uio.ko

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

echo "8086 0443" > /sys/bus/pci/drivers/igb_uio/new_id
```

You can use `lspci -vvd:443` to confirm that all devices are now in use by igb_uio kernel driver.

For **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 yours are different adjust the unbind command below:

```
cd $RTE_SDK
modprobe uio
insmod ./build/kmod/igb_uio.ko

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

echo "8086 37c9" > /sys/bus/pci/drivers/igb_uio/new_id
```

You can use `lspci -vvd:37c9` to confirm that all devices are now in use by igb_uio kernel driver.

For **Intel(R) QuickAssist Technology C3xxx** device: The unbind command below assumes `bdfs` of `01:01.00-01:02.07`, if yours are different adjust the unbind command below:

```
cd $RTE_SDK
modprobe uio
insmod ./build/kmod/igb_uio.ko

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

echo "8086 19e3" > /sys/bus/pci/drivers/igb_uio/new_id
```

You can use `lspci -vvd:19e3` to confirm that all devices are now in use by igb_uio kernel driver.

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

```
cd $RTE_SDK
./usertools/dpdk-devbind.py -b igb_uio 0000:03:01.1
```

CHAPTER

ELEVEN

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.

11.1 Features

ZUC PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_CIPHER_ZUC_EEA3

Authentication algorithm:

- RTE_CRYPTO_AUTH_ZUC_EIA3

11.2 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 in the underlying library.

11.3 Installation

To build DPDK with the ZUC_PMD the user is required to download the export controlled `libsso_zuc` library, by requesting it from https://networkbuilders.intel.com/network-technologies/dpdk. Once approval has been granted, the user needs to log in https://networkbuilders.intel.com/dpdklogin and click on "ZUC Library" link, to download the library. After downloading the library, the user needs to unpack and compile it on their system before building DPDK:

```
make
```

11.4 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).
- Build the LIBSSO_ZUC library (explained in Installation section).
- Build DPDK as follows:

```
make config T=x86_64-native-linuxapp-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_eal_vdev_init("crypto_zuc") within the application.
- Use –vdev="crypto_zuc" in the EAL options, which will call rte_eal_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 -c 40 -n 4 --vdev="crypto_zuc,socket_id=1,max_nb_sessions=128"
```