DPDK

DATA PLANE DEVELOPMENT KIT

Crypto Device Drivers

Release 16.04.0

April 12, 2016

CHAPTER

ONE

CRYPTO DEVICE SUPPORTED FUNCTIONALITY MATRICES

Supported Feature Flags

Feature Flags	qat	null	aesni_mb	aesni_gcm	snow3g
RTE_CRYPTODEV_FF_SYMMETRIC_CRYPTO	x	x			
RTE_CRYPTODEV_FF_ASYMMETRIC_CRYPTO					
RTE_CRYPTODEV_FF_SYM_OPERATION_CHAINING	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	x	
RTE_CRYPTODEV_FF_CPU_AESNI			x	x	
RTE_CRYPTODEV_FF_HW_ACCELERATED	x				

Supported Cipher Algorithms

Cipher Algorithms	qat	null	aesni_mb	aesni_gcm	snow3g
NULL		x			
AES_CBC_128	x		x		
AES_CBC_192	x		x		
AES_CBC_256	x		x		
AES_CTR_128					
AES_CTR_192					
AES_CTR_256					
SNOW3G_UEA2	x				x

Supported Authentication Algorithms

Cipher Algorithms	qat	null	aesni_mb	aesni_gcm	snow3g
NONE		x			
MD5					
MD5_HMAC			x		
SHA1					
SHA1_HMAC	x		x		
SHA224					
SHA224_HMAC			x		
SHA256					
SHA256_HMAC	x		x		
SHA384					
SHA384_HMAC			x		
SHA512					
SHA512_HMAC	x		x		
AES_XCBC	x		x		
SNOW3G_UIA2	x				x

Supported AEAD Algorithms

AEAD Algorithms	qat	null	aesni_mb	aesni_gcm	snow3g
AES_GCM_128	x		x		
AES_GCM_192	x				
AES_GCM_256	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_SYM_CIPHER_AES128_CBC
- RTE_CRYPTO_SYM_CIPHER_AES256_CBC
- RTE_CRYPTO_SYM_CIPHER_AES512_CBC

Hash algorithms:

- RTE_CRYPTO_SYM_HASH_SHA1_HMAC
- RTE_CRYPTO_SYM_HASH_SHA256_HMAC
- RTE_CRYPTO_SYM_HASH_SHA512_HMAC

2.2 Limitations

- Chained mbufs are not supported.
- Hash only is not supported.
- Cipher only is 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).
- Not performance tuned.

2.3 Installation

To build DPDK with the AESNI_MB_PMD the user is required to download the mult- buffer library from here and compile it on their user system before building DPDK. When building the multi-buffer library it is necessary to have YASM package installed and also requires the overriding of YASM path when building, as a path is hard coded in the Makefile of the release package.

```
make YASM=/usr/bin/yasm
```

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("cryptodev_aesni_mb_pmd") within the application.
- Use –vdev="cryptodev_aesni_mb_pmd" 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="cryptodev_aesni_mb_pmd,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 multi buffer library (see AES-NI Multi-buffer PMD documentation to learn more about it, including installation).

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

3.1 Features

AESNI GCM PMD has support for:

Cipher algorithms:

- RTE_CRYPTO_CIPHER_AES_GCM

Authentication algorithms:

- RTE_CRYPTO_AUTH_AES_GCM

3.2 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 (go to Installation section in AES-NI MB PMD documentation).
- 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("cryptodev_aesni_gcm_pmd") within the application.
- Use –vdev="cryptodev_aesni_gcm_pmd" 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="cryptodev_aesni_gcm_pmd,socket_id=1,max_nb_sessions=128"
```

3.3 Limitations

- Chained mbufs are not supported.
- Hash only is not supported.
- Cipher only is 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).
- Not performance tuned.

CHAPTER

FOUR

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.

4.1 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

4.2 Limitations

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

4.3 Installation

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

4.4 Initialization

To use the PMD in an application, user must:

- Call rte_eal_vdev_init("cryptodev_null_pmd") within the application.
- Use –vdev="cryptodev_null_pmd" 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="cryptodev_null_pmd,socket_id=1,max_nb_sessions=128"
```

CHAPTER

FIVE

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.

5.1 Features

SNOW 3G PMD has support for:

Cipher algorithm:

- RTE_CRYPTO_SYM_CIPHER_SNOW3G_UEA2

Authentication algorithm:

- RTE_CRYPTO_SYM_AUTH_SNOW3G_UIA2

5.2 Limitations

- Chained mbufs are not supported.
- Snow3g(UEA2) supported only if cipher length, cipher offset fields are byte-aligned.
- Snow3g(UIA2) supported only if hash length, hash offset fields are byte-aligned.

5.3 Installation

To build DPDK with the SNOW3G_PMD the user is required to download the export controlled `libsso` library, by requesting it from https://networkbuilders.intel.com/network-technologies/dpdk, and compiling it on their system before building DPDK:

```
make -f Makefile_snow3g
```

Note: If using a gcc version higher than 5.0, and compilation fails, apply the following patch:

```
/libsso/src/snow3g/sso_snow3g.c

static inline void ClockFSM_4(sso_snow3gKeyState4_t *pCtx, __m128i *data)
{
    __m128i F, R;
-    uint32_t K, L;
+    uint32_t K;
+    /* Declare unused if SNOW3G_WSM/SNB are defined */
```

```
+    uint32_t L __attribute__ ((unused)) = 0;

     F = _mm_add_epi32(pCtx->LFSR_X[15], pCtx->FSM_X[0]);
     R = _mm_xor_si128(pCtx->LFSR_X[5], pCtx->FSM_X[2]);

/libsso/include/sso_snow3g_internal.h

-inline void ClockFSM_1(sso_snow3gKeyState1_t *pCtx, uint32_t *data);
-inline void ClockLFSR_1(sso_snow3gKeyState1_t *pCtx);
-inline void sso_snow3gStateInitialize_1(sso_snow3gKeyState1_t * pCtx, sso_snow3g_key_schedule_
+void ClockFSM_1(sso_snow3gKeyState1_t *pCtx, uint32_t *data);
+void ClockLFSR_1(sso_snow3gKeyState1_t *pCtx);
+void sso_snow3gStateInitialize_1(sso_snow3gKeyState1_t * pCtx, sso_snow3g_key_schedule_t *pKey
```

5.4 Initialization

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

- Export the environmental variable LIBSSO_PATH with the path where the library was extracted.
- Build the LIBSSO 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("cryptodev_snow3g_pmd") within the application.
- Use –vdev="cryptodev_snow3g_pmd" 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="cryptodev_snow3g_pmd,socket_id=1,max_nb_sessions=128"
```

CHAPTER

SIX

QUICK ASSIST CRYPTO POLL MODE DRIVER

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

6.1 Features

The QAT PMD has support for:

Cipher algorithms:

- `RTE_CRYPTO_SYM_CIPHER_AES128_CBC`
- `RTE_CRYPTO_SYM_CIPHER_AES192_CBC`
- `RTE_CRYPTO_SYM_CIPHER_AES256_CBC`
- `RTE_CRYPTO_SYM_CIPHER_SNOW3G_UEA2`
- `RTE_CRYPTO_CIPHER_AES_GCM`

Hash algorithms:

- `RTE_CRYPTO_AUTH_SHA1_HMAC`
- `RTE_CRYPTO_AUTH_SHA256_HMAC`
- `RTE_CRYPTO_AUTH_SHA512_HMAC`
- `RTE_CRYPTO_AUTH_AES_XCBC_MAC`
- `RTE_CRYPTO_AUTH_SNOW3G_UIA2`

6.2 Limitations

- Chained mbufs are not supported.
- Hash only is not supported except Snow3G UIA2.
- Cipher only is not supported except Snow3G UEA2.
- Only supports the session-oriented API implementation (session-less APIs are not supported).
- Not performance tuned.
- Snow3g(UEA2) supported only if cipher length, cipher offset fields are byte-aligned.

- Snow3g(UIA2) supported only if hash length, hash offset fields are byte-aligned.
- No BSD support as BSD QAT kernel driver not available.

6.3 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.

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*.

6.4 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 `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`

6.5 Installation using kernel.org driver

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 VT-d is disabled.

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 VFs using the sysfs file system.

First find the bdf 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`

6.6 Binding the available VFs to the DPDK UIO driver

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.