Virtio-crypto device Spec

Signed-off-by: Gonglei <arei.gonglei@huawei.com>

# Crypto Device

The virtio crypto device is a virtual crypto device (ie. hardware crypto accelerator card). The encryption and decryption requests of are placed in the data queue, and handled by the real hardware crypto accelerators finally. A second queue is the control queue, which is used to create or destroy session for symmetric algorithms, and to control some advanced features.

## Device ID

65535 (experimental)

## Virtqueues

0 dataq

…

N-1 dataq

N controlq

*N=1 if VIRTIO\_CRYPTO\_F\_MQ is not negotiated, otherwise N is set by max\_virtqueues.*

## Feature bits

**VIRTIO\_CRYPTO\_F\_STATUS (0)**

Configuration status field is available.

**VIRTIO\_CRYPTO\_F\_MQ (1)**

 Device supports multiqueue to encrypt and decrypt.

**VIRTIO\_CRYPTO\_F\_ALGS (2)**

Configuration algorithms field is available.

## Device configuration layout

Three driver-read-only configuration fields are currently defined. The status only exists if VIRTIO\_CRYPTO\_F\_STATUS is set. On read-only bit (for the driver) is currently defined for the status field: VIRTIO\_CRYPTO\_S\_HW\_READY.

|  |
| --- |
| #define VIRTIO\_CRYPTO\_S\_HW\_READY 1 |

The following driver-read-only field, max\_virtqueues only exists if VIRTIO\_CRYPTO\_F\_MQ is set. This field specifies the maximum number of data virtqueues (dataq1. . .dataqN) that can be configured once VIRTIO\_CRYPTO\_F\_MQ is negotiated.

|  |
| --- |
| struct virtio\_crypto\_config { le16 status;le16 max\_virtqueues;le32 algorithms;} |

The last driver-read-only field, algorithms only exists if VIRTIO\_CRYPTO\_F\_ALGS is set. This field specifies the algorithms which the device offered once VIRTIO\_CRYPTO\_F\_ALGS is negotiated. Two read-only bits (for the driver) are currently defined for the algorithms field: VIRTIO\_CRYPTO\_ALG\_SYM and VIRTIO\_CRYPTO\_ALG\_ASYM.

|  |
| --- |
| #define VIRTIO\_CRYPTO\_ALG\_SYM (1 << 0)#define VIRTIO\_CRYPTO\_ALG\_SYM (1 << 1) |

### Device Requirements: Device configuration layout

The device MUST set max\_virtqueues to between 1 and 0x8000 inclusive, if it offers VIRTIO\_CRYPTO\_F\_MQ.

If the driver does not negotiate the VIRTIO\_CRYPTO\_F\_STATUS feature, it SHOULD assume the hardware-backed implementation is ready, otherwise it SHOULD read the ready status from the bottom bit of status.

If the driver does not negotiate the VIRTIO\_CRYPTO\_F\_ALGS feature, it SHOULD assume the

device support all algorithms.

## Device Initialization

A driver would perform a typical initialization routine like so:

1. Identify and initialize data virtqueue, up to N if VIRTIO\_CRYPTO\_F\_MQ feature bit is negotiated, N=max\_virtqueues, otherwise identify N=1.

2. Identify the control virtqueue.

3. If the VIRTIO\_CRYPTO\_F\_STATUS feature bit is negotiated, the ready status of hardware-backend comes from the bottom bit of status. Otherwise, the driver assumes it’s active.

4. If the VIRTIO\_CRYPTO\_F\_ALGS feature bit is negotiated, the driver can read the supported algorithms from bits of algorithms. Otherwise, the driver assumes all algorithms are supported.

## Device Operation

### Session operation

The symmetric algorithms have the concept of sessions. A session is a handle which describes the

cryptographic parameters to be applied to a number of buffers. The data within a session handle includes the following:

1. The operation (cipher, hash or both, and if both, the order in which the algorithms should be applied).

2. The cipher setup data, including the cipher algorithm and mode, the key and its length, and the direction (encrypt or decrypt).

3. The hash setup data, including the hash algorithm, mode (plain, nested or authenticated), and digest result length (to allow for truncation).

*  Authenticated mode can refer to HMAC, which requires that the key and its length are also specified. It is also used for GCM and CCM authenticated encryption, in which case the AAD length is also specified.
*  For nested mode, the inner and outer prefix data and length are specified, as well as the outer hash algorithm.

The controlq virtqueue is used to control session operations, including creation or close. The request is preceded by a header:

|  |
| --- |
| struct virtio\_crypto\_sym\_ctlhdr { /\* control type \*/ u8 type;}; |

Two bits are currently defined for the control header type:

|  |
| --- |
| enum virto\_crypto\_ctl\_type { VIRTIO\_CRYPTO\_CTRL\_CREATE\_SESSION = 1, VIRTIO\_CRYPTO\_CTRL\_CLOSE\_SESSION = 2,}; |

**1.6.1.1 Session creation**

Firstly, the session creation request MUST be set the control type with VIRTIO\_CRYPTO\_CTRL\_CREATE\_SESSION, then the request is preceded by an operation header:

|  |
| --- |
| typedef struct virtio\_crypto\_sym\_session\_op { virtio\_crypto\_sym\_op\_type\_t op\_type; virtio\_crypto\_sym\_cipher\_t cipher\_setup\_data; virtio\_crypto\_sym\_hash\_t hash\_setup\_data; virtio\_crypto\_sym\_alg\_chain\_order\_t alg\_chain\_order; u8 verify\_digest;} virtio\_crypto\_sym\_session\_op\_t; |

And the structures definition details are:

|  |
| --- |
| typedef enum virtio\_crypto\_sym\_op\_type{ VIRTIO\_CRYPTO\_SYM\_OP\_NONE = 0, /\*\*< No operation \*/ VIRTIO\_CRYPTO\_SYM\_OP\_CIPHER, /\*\*< Cipher only operation on the data \*/ VIRTIO\_CRYPTO\_SYM\_OP\_HASH, /\*\*< Hash only operation on the data \*/ VIRTIO\_CRYPTO\_SYM\_OP\_ALGORITHM\_CHAINING /\*\*< Chain any cipher with any hash operation. The order depends on \* the value in the CpaCySymAlgChainOrder enum. \* \* This value is also used for authenticated ciphers (GCM and CCM), in \* which case the cipherAlgorithm should take one of the values @ref \* CPA\_CY\_SYM\_CIPHER\_AES\_CCM or @ref CPA\_CY\_SYM\_CIPHER\_AES\_GCM, while the \* hashAlgorithm should take the corresponding value @ref \* CPA\_CY\_SYM\_HASH\_AES\_CCM or @ref CPA\_CY\_SYM\_HASH\_AES\_GCM. \*/} virtio\_crypto\_sym\_op\_type\_t;typedef enum virtio\_crypto\_sym\_hash\_mode{ VIRTIO\_CRYPTO\_SYM\_HASH\_MODE\_PLAIN = 1, /\*\*< Plain hash. Can be specified for MD5 and the SHA family of \* hash algorithms. \*/ VIRTIO\_CRYPTO\_SYM\_HASH\_MODE\_AUTH, /\*\*< Authenticated hash. This mode may be used in conjunction with the \* MD5 and SHA family of algorithms to specify HMAC. It MUST also be \* specified with all of the remaining algorithms, all of which are in \* fact authentication algorithms. \*/ VIRTIO\_CRYPTO\_SYM\_HASH\_MODE\_NESTED /\*\*< Nested hash. Can be specified for MD5 and the SHA family of \* hash algorithms. \*/} virtio\_crypto\_sym\_hash\_mode\_t;typedef struct virtio\_crypto\_sym\_hash\_auth\_mode { /\* length of authenticated key \*/ le32 auth\_key\_len; /\* The length of the additional authenticated data (AAD) in bytes \*/ le32 aad\_len;} virtio\_crypto\_sym\_hash\_auth\_mode\_t;typedef struct virtio\_crypto\_sym\_cipher { /\* cipher algorithm type (ie. aes-cbc ) \*/ enum virtio\_crypto\_cipher\_alg alg; /\* length of key \*/ le32 keylen; /\* encrypt or decrypt \*/ u8 op;} virtio\_crypto\_sym\_cipher\_t;typedef struct virtio\_crypto\_sym\_hash { /\* hash algorithm type \*/ enum virtio\_crypto\_hash\_alg hash\_alg; /\* mode of hash operation, including authenticated/plain/nested hash \*/ virtio\_crypto\_sym\_hash\_mode\_t hash\_mode; /\* hash result length \*/ le32 hash\_result\_len; virtio\_crypto\_sym\_hash\_auth\_mode\_t auth\_mode\_setup\_data;} virtio\_crypto\_sym\_hash\_t;typedef enum virtio\_crypto\_sym\_alg\_chain\_order { /\* Perform the hash operation followed by the cipher operation \*/ VIRTIO\_CRYPTO\_SYM\_ALG\_CHAIN\_ORDER\_HASH\_THEN\_CIPHER = 1, /\* Perform the cipher operation followed by the hash operation \*/ VIRTIO\_CRYPTO\_SYM\_ALG\_CHAIN\_ORDER\_CIPHER\_THEN\_HASH} virtio\_crypto\_sym\_alg\_chain\_order\_t; |

When the device finish the processing of session close, the device MUST return a session identifier to the driver. So the session creation request MUST end by a tailer:

|  |
| --- |
| typedef struct virtio\_crypto\_sym\_session\_op\_inhdr { u8 status; le64 session\_id;} virtio\_crypto\_sym\_session\_op\_inhdr\_t; |

Both status and session\_id are written by the device: either VIRTIO\_CRYPTO\_CTRL\_OK for success, VIRTIO\_CRYPTO\_CTRL\_ERR for creation failed or device error.

|  |
| --- |
| enum { VIRTIO\_CRYPTO\_CTRL\_OK = 0, VIRTIO\_CRYPTO\_CTRL\_ERR = 1,}; |

**1.6.1.2 Session close**

The session close request MUST be set the control type with VIRTIO\_CRYPTO\_CTRL\_CLOSE\_SESSION and pass the session\_id to the device.

### Encryption operation

The encryption and decryption requests and the corresponding results are transmitted by placing them in dataq. The symmetric algorithms requests are preceded by a header:

|  |
| --- |
| typedef struct virtio\_crypto\_sym\_crypt\_op { /\* the backend returned session identifier \*/ le64 session\_id;/\* length of initial vector \*/ le32 iv\_len; /\* iv offset in the whole crypto data memory \*/ le32 iv\_offset; /\* length of additional auth data \*/ le32 auth\_len; /\* additional auth data offset in the whole crypto data memory \*/ le32 additional\_auth\_offset; /\* cipher start source offest \*/ le32 cipher\_start\_src\_offset; le32 len\_to\_cipher; /\* hash start source offest \*/ le32 hash\_start\_src\_offset; le32 len\_to\_hash; /\* length of source data \*/ le32 source\_len;} virtio\_crypto\_sym\_crypt\_op\_t; |

The data requests end by a status byte. The ﬁnal status byte is written by the device: either VIRTIO\_CRYPTO\_S\_OK for success, VIRTIO\_CRYPTO\_S\_ERR for device or driver error, VIRTIO\_CRYPTO\_S\_BADMSG for verification failed when decrypt AEAD algorithms:

|  |
| --- |
| #define VIRTIO\_CRYPTO\_S\_OK 0#define VIRTIO\_CRYPTO\_S\_ERR 1#define VIRTIO\_CRYPTO\_S\_BADMSG 2 |

**1.6.2.1 Steps of encryption Operation**

Both ctrlq and dataq virtqueue are bidirectional.

Step1: Create a session:

* + - 1. The driver fill out the context message, include algorithm name, key, keylen etc;
			2. The driver send a context message to the backend device by controlq;
			3. The device create a session using the message transmitted by controlq;
			4. Return the session id to the driver.

Step 2: Execute the detail encryption operation:

1. The driver fill out the encrypt requests;
2. Put the requests into dataq and kick the virtqueue;
3. The device execute the encryption operation according the requests’ arguments;
4. The device return the encryption result to the driver by dataq;
5. The driver callback handle the result and over.

Note: the driver CAN support both synchronous and asynchronous encryption. Then the performance is poor in synchronous operation because frequent context switching and virtualization overhead. The driver SHOULD by preference use asynchronous encryption.

### Decryption Operation

The decryption process is the same with encryption, except that the device MUST verify and return the verify result to the driver. If the verify result is not correct, VIRTIO\_CRYPTO\_S\_BADMSG (bad message) SHOULD be returned the driver.