

**NAME**

**crypto\_driver** - interface for symmetric cryptographic drivers

**SYNOPSIS**

```
#include <opencrypto/cryptodev.h>
```

*void*

```
crypto_copyback(struct cryptop *crp, int off, int size, const void *src);
```

*void*

```
crypto_copydata(struct cryptop *crp, int off, int size, void *dst);
```

*void*

```
crypto_done(struct cryptop *crp);
```

*int32\_t*

```
crypto_get_driverid(device_t dev, size_t session_size, int flags);
```

*void \**

```
crypto_get_driver_session(crypto_session_t crypto_session);
```

*void*

```
crypto_read_iv(struct cryptop *crp, void *iv);
```

*int*

```
crypto_unblock(uint32_t driverid, int what);
```

*int*

```
crypto_unregister_all(uint32_t driverid);
```

*int*

```
CRYPTODEV_FREESESSION(device_t dev, crypto_session_t crypto_session);
```

*int*

```
CRYPTODEV_NEWSESSION(device_t dev, crypto_session_t crypto_session,  
const struct crypto_session_params *csp);
```

*int*

```
CRYPTODEV_PROBESSESSION(device_t dev, const struct crypto_session_params *csp);
```

*int*

**CRYPTODEV\_PROCESS**(*device\_t dev, struct cryptop \*crp, int flags*);

*void*

**hmac\_init\_ipad**(*struct auth\_hash \*axf, const char \*key, int klen, void \*auth\_ctx*);

*void*

**hmac\_init\_opad**(*struct auth\_hash \*axf, const char \*key, int klen, void \*auth\_ctx*);

## DESCRIPTION

Symmetric cryptographic drivers process cryptographic requests submitted to sessions associated with the driver.

Cryptographic drivers call **crypto\_get\_driverid**() to register with the cryptographic framework. *dev* is the device used to service requests. The **CRYPTODEV**() methods are defined in the method table for the device driver attached to *dev*. *session\_size* specifies the size of a driver-specific per-session structure allocated by the cryptographic framework. *flags* is a bitmask of properties about the driver. Exactly one of **CRYPTOCAP\_F\_SOFTWARE** or **CRYPTOCAP\_F\_HARDWARE** must be specified. **CRYPTOCAP\_F\_SOFTWARE** should be used for drivers which process requests using host CPUs. **CRYPTOCAP\_F\_HARDWARE** should be used for drivers which process requests on separate co-processors. **CRYPTOCAP\_F\_SYNC** should be set for drivers which process requests synchronously in **CRYPTODEV\_PROCESS**(). **CRYPTOCAP\_F\_ACCEL\_SOFTWARE** should be set for software drivers which use accelerated CPU instructions. **crypto\_get\_driverid**() returns an opaque driver id.

**crypto\_unregister\_all**() unregisters a driver from the cryptographic framework. If there are any pending operations or open sessions, this function will sleep. *driverid* is the value returned by an earlier call to **crypto\_get\_driverid**().

When a new session is created by **crypto\_newsession**(), **CRYPTODEV\_PROBESESSION**() is invoked by the cryptographic framework on each active driver to determine the best driver to use for the session. This method should inspect the session parameters in *csp*. If a driver does not support requests described by *csp*, this method should return an error value. If the driver does support requests described by *csp*, it should return a negative value. The framework prefers drivers with the largest negative value, similar to **DEVICE\_PROBE**(9). The following values are defined for non-error return values from this method:

**CRYPTODEV\_PROBE\_HARDWARE**      The driver processes requests via a co-processor.

**CRYPTODEV\_PROBE\_ACCEL\_SOFTWARE**      The driver processes requests on the host CPU using optimized instructions such as AES-NI.

**CRYPTODEV\_PROBE\_SOFTWARE**

The driver processes requests on the host CPU.

This method should not sleep.

Once the framework has chosen a driver for a session, the framework invokes the **CRYPTODEV\_NEWSESSION()** method to initialize driver-specific session state. Prior to calling this method, the framework allocates a per-session driver-specific data structure. This structure is initialized with zeroes, and its size is set by the *session\_size* passed to **crypto\_get\_driverid()**. This method can retrieve a pointer to this data structure by passing *crypto\_session* to **crypto\_get\_driver\_session()**. Session parameters are described in *csp*.

This method should not sleep.

**CRYPTODEV\_FREESSESSION()** is invoked to release any driver-specific state when a session is destroyed. The per-session driver-specific data structure is explicitly zeroed and freed by the framework after this method returns. If a driver requires no additional tear-down steps, it can leave this method undefined.

This method should not sleep.

**CRYPTODEV\_PROCESS()** is invoked for each request submitted to an active session. This method can either complete a request synchronously or schedule it to be completed asynchronously, but it must not sleep.

If this method is not able to complete a request due to insufficient resources such as a full command queue, it can defer the request by returning **ERESTART**. The request will be queued by the framework and retried once the driver releases pending requests via **crypto\_unblock()**. Any requests submitted to sessions belonging to the driver will also be queued until **crypto\_unblock()** is called.

If a driver encounters errors while processing a request, it should report them via the *crp\_etype* field of *crp* rather than returning an error directly.

*flags* may be set to **CRYPTO\_HINT\_MORE** if there are additional requests queued for this driver. The driver can use this as a hint to batch completion interrupts. Note that these additional requests may be from different sessions.

**crypto\_get\_driver\_session()** returns a pointer to the driver-specific per-session data structure for the session *crypto\_session*. This function can be used in the **CRYPTODEV\_NEWSESSION()**, **CRYPTODEV\_PROCESS()**, and **CRYPTODEV\_FREESSESSION()** callbacks.

**crypto\_copydata()** copies *size* bytes out of the input buffer for *crp* into a local buffer pointed to by *dst*. The bytes are read starting at an offset of *off* bytes in the request's input buffer.

**crypto\_copyback()** copies *size* bytes from the local buffer pointed to by *src* into the output buffer for *crp*. The bytes are written starting at an offset of *off* bytes in the request's output buffer.

**crypto\_read\_iv()** copies the IV or nonce for *crp* into the local buffer pointed to by *iv*.

A driver calls **crypto\_done()** to mark the request *crp* as completed. Any errors should be set in *crp\_etype* prior to calling this function.

If a driver defers a request by returning ERESTART from CRYPTO\_PROCESS, the framework will queue all requests for the driver until the driver calls **crypto\_unblock()** to indicate that the temporary resource shortage has been relieved. For example, if a driver returns ERESTART due to a full command ring, it would invoke **crypto\_unblock()** from a command completion interrupt that makes a command ring entry available. *driverid* is the value returned by **crypto\_get\_driverid()**. *what* indicates which types of requests the driver is able to handle again:

CRYPTO\_SYMQ indicates that the driver is able to handle symmetric requests passed to **CRYPTODEV\_PROCESS()**.

**hmac\_init\_ipad()** prepares an authentication context to generate the inner hash of an HMAC. *axf* is a software implementation of an authentication algorithm such as the value returned by **crypto\_auth\_hash()**. *key* is a pointer to a HMAC key of *klen* bytes. *auth\_ctx* points to a valid authentication context for the desired algorithm. The function initializes the context with the supplied key.

**hmac\_init\_opad()** is similar to **hmac\_init\_ipad()** except that it prepares an authentication context to generate the outer hash of an HMAC.

## RETURN VALUES

**crypto\_apply()** returns the return value from the caller-supplied callback function.

**crypto\_contiguous\_subsegment()** returns a pointer to a contiguous segment or NULL.

**crypto\_get\_driverid()** returns a driver identifier on success or -1 on error.

**crypto\_unblock()**, **crypto\_unregister\_all()**, **CRYPTODEV\_FREESESSION()**, **CRYPTODEV\_NEWSESSION()**, and **CRYPTODEV\_PROCESS()** return zero on success or an error on failure.

**CRYPTODEV\_PROBESESSION()** returns a negative value on success or an error on failure.

**SEE ALSO**

crypto(7), crypto(9), crypto\_buffer(9), crypto\_request(9), crypto\_session(9)