NAME

crypto_request - symmetric cryptographic operations

SYNOPSIS

#include <opencrypto/cryptodev.h>

*struct cryptop **
crypto_clonereq(*crypto_session_t cses, struct cryptop *crp, int how*);

int
crypto_dispatch(*struct cryptop *crp*);

int
crypto_dispatch_async(*struct cryptop *crp, int flags*);

void
crypto_dispatch_batch(*struct cryptopq *crpq, int flags*);

void
crypto_destroyreq(*struct cryptop *crp*);

void
crypto_freereq(*struct cryptop *crp*);

*struct cryptop **
crypto_getreq(*crypto_session_t cses, int how*);

void
crypto_initreq(*crypto_session_t cses, int how*);

void
crypto_use_buf(*struct cryptop *crp, void *buf, int len*);

void
crypto_use_mbuf(*struct cryptop *crp, struct mbuf *m*);

void
crypto_use_uio(*struct cryptop *crp, struct uio *uio*);

void

crypto_use_vmpage(*struct cryptop *crp, vm_page_t *pages, int len, int offset*);

void
crypto_use_output_buf(*struct cryptop *crp, void *buf, int len*);

void
crypto_use_output_mbuf(*struct cryptop *crp, struct mbuf *m*);

void
crypto_use_output_uio(*struct cryptop *crp, struct uio *uio*);

void
crypto_use_output_vmpage(*struct cryptop *crp, vm_page_t *pages, int len, int offset*);

DESCRIPTION

Each symmetric cryptographic operation in the kernel is described by an instance of *struct cryptop* and is associated with an active session.

Requests can either be allocated dynamically or use caller-supplied storage. Dynamically allocated requests should be allocated by either **crypto_getreq**() or **crypto_clonereq**(), and freed by **crypto_freereq**() once the request has completed. Requests using caller-supplied storage should be initialized by **crypto_initreq**() at the start of each operation and destroyed by **crypto_destroyreq**() once the request has completed.

For **crypto_clonereq**(), **crypto_getreq**(), and **crypto_initreq**(), *cses* is a reference to an active session. For **crypto_clonereq**() and **crypto_getreq**(), *how* is passed to malloc(9) and should be set to either M_NOWAIT or M_WAITOK.

crypto_clonereq() allocates a new request that inherits request inputs such as request buffers from the original *crp* request. However, the new request is associated with the *cses* session rather than inheriting the session from *crp*. *crp* must not be a completed request.

Once a request has been initialized, the caller should set fields in the structure to describe request-specific parameters. Unused fields should be left as-is.

The **crypto_dispatch**(), **crypto_dispatch_async**(), and **crypto_dispatch_batch**() functions pass one or more crypto requests to the driver attached to the request's session. If there are errors in the request's fields, these functions may return an error to the caller. If errors are encountered while servicing the request, they will instead be reported to the request's callback function (*crp_callback*) via *crp_etype*.

Note that a request's callback function may be invoked before **crypto_dispatch**() returns.

Once a request has signaled completion by invoking its callback function, it should be freed via **crypto_destroyreq**() or **crypto_freereq**().

Cryptographic operations include several fields to describe the request.

Request Buffers

Requests can either specify a single data buffer that is modified in place (*crp_buf*) or separate input (*crp_buf*) and output (*crp_obuf*) buffers. Note that separate input and output buffers are not supported for compression mode requests.

All requests must have a valid *crp_buf* initialized by one of the following functions:

crypto_use_buf() Uses an array of *len* bytes pointed to by *buf* as the data buffer.

crypto_use_mbuf() Uses the network memory buffer *m* as the data buffer.

crypto_use_uio() Uses the scatter/gather list *uio* as the data buffer.

crypto_use_vmpage() Uses the array of *vm_page_t* structures as the data buffer.

One of the following functions should be used to initialize *crp_obuf* for requests that use separate input and output buffers:

crypto_use_output_buf() Uses an array of *len* bytes pointed to by *buf* as the output buffer.

crypto_use_output_mbuf() Uses the network memory buffer *m* as the output buffer.

crypto_use_output_uio() Uses the scatter/gather list *uio* as the output buffer.

crypto_use_output_vmpage() Uses the array of *vm_page_t* structures as the output buffer.

Request Regions

Each request describes one or more regions in the data buffers. Each region is described by an offset relative to the start of a data buffer and a length. The length of some regions is the same for all requests belonging to a session. Those lengths are set in the session parameters of the associated session. All requests must define a payload region. Other regions are only required for specific session modes.

For requests with separate input and output data buffers, the AAD, IV, and payload regions are always

defined as regions in the input buffer, and a separate payload output region is defined to hold the output of encryption or decryption in the output buffer. The digest region describes a region in the input data buffer for requests that verify an existing digest. For requests that compute a digest, the digest region describes a region in the output data buffer. Note that the only data written to the output buffer is the encryption or decryption result and any computed digest. AAD and IV regions are not copied from the input buffer into the output buffer but are only used as inputs.

The following regions are defined:

Region	**Buffer**	**Description**
AAD	Input	Embedded Additional Authenticated Data
IV	Input	Embedded IV or nonce
Payload	Input	Data to encrypt, decrypt, compress, or decompress
Payload Output	Output	Encrypted or decrypted data
Digest	Input/Output	Authentication digest, hash, or tag

Region	**Start**	**Length**
AAD	*crp_aad_start*	*crp_aad_length*
IV	*crp_iv_start*	*csp_ivlen*
Payload	*crp_payload_start*	*crp_payload_length*
Payload Output	*crp_payload_output_start*	*crp_payload_length*
Digest	*crp_digest_start*	*csp_auth_mlen*

Requests are permitted to operate on only a subset of the data buffer. For example, requests from IPsec operate on network packets that include headers not used as either additional authentication data (AAD) or payload data.

Request Operations

All requests must specify the type of operation to perform in *crp_op*. Available operations depend on the session's mode.

Compression requests support the following operations:

CRYPTO_OP_COMPRESS Compress the data in the payload region of the data buffer.

CRYPTO_OP_DECOMPRESS Decompress the data in the payload region of the data buffer.

Cipher requests support the following operations:

CRYPTO_OP_ENCRYPT Encrypt the data in the payload region of the data buffer.

CRYPTO_OP_DECRYPT Decrypt the data in the payload region of the data buffer.

Digest requests support the following operations:

CRYPTO_OP_COMPUTE_DIGEST Calculate a digest over the payload region of the data buffer and store the result in the digest region.

CRYPTO_OP_VERIFY_DIGEST Calculate a digest over the payload region of the data buffer. Compare the calculated digest to the existing digest from the digest region. If the digests match, complete the request successfully. If the digests do not match, fail the request with EBADMSG.

AEAD and Encrypt-then-Authenticate requests support the following operations:

CRYPTO_OP_ENCRYPT | CRYPTO_OP_COMPUTE_DIGEST
Encrypt the data in the payload region of the data buffer. Calculate a digest over the AAD and payload regions and store the result in the data buffer.

CRYPTO_OP_DECRYPT | CRYPTO_OP_VERIFY_DIGEST
Calculate a digest over the AAD and payload regions of the data buffer. Compare the calculated digest to the existing digest from the digest region. If the digests match, decrypt the payload region. If the digests do not match, fail the request with EBADMSG.

Request AAD

AEAD and Encrypt-then-Authenticate requests may optionally include Additional Authenticated Data. AAD may either be supplied in the AAD region of the input buffer or as a single buffer pointed to by *crp_aad*. In either case, *crp_aad_length* always indicates the amount of AAD in bytes.

Request ESN

IPsec requests may optionally include Extended Sequence Numbers (ESN). ESN may either be supplied in *crp_esn* or as part of the AAD pointed to by *crp_aad*.

If the ESN is stored in *crp_esn*, CSP_F_ESN should be set in *csp_flags*. This use case is dedicated for encrypt and authenticate mode, since the high-order 32 bits of the sequence number are appended after the Next Header (RFC 4303).

AEAD modes supply the ESN in a separate AAD buffer (see e.g. RFC 4106, Chapter 5 AAD Construction).

Request IV and/or Nonce

Some cryptographic operations require an IV or nonce as an input. An IV may be stored either in the IV region of the data buffer or in *crp_iv*. By default, the IV is assumed to be stored in the IV region. If the IV is stored in *crp_iv*, CRYPTO_F_IV_SEPARATE should be set in *crp_flags* and *crp_iv_start* should be left as zero.

Requests that store part, but not all, of the IV in the data buffer should store the partial IV in the data buffer and pass the full IV separately in *crp_iv*.

Request and Callback Scheduling

The crypto framework provides multiple methods of scheduling the dispatch of requests to drivers along with the processing of driver callbacks. The **crypto_dispatch**(), **crypto_dispatch_async**(), and **crypto_dispatch_batch**() functions can be used to request different dispatch scheduling policies.

crypto_dispatch() synchronously passes the request to the driver. The driver itself may process the request synchronously or asynchronously depending on whether the driver is implemented by software or hardware.

crypto_dispatch_async() dispatches the request asynchronously. If the driver is inherently synchronous, the request is queued to a taskqueue backed by a pool of worker threads. This can increase througput by allowing requests from a single producer to be processed in parallel. By default the pool is sized to provide one thread for each CPU. Worker threads dequeue requests and pass them to the driver asynchronously. **crypto_dispatch_async**() additionally takes a *flags* parameter. The CRYPTO_ASYNC_ORDERED flag indicates that completion callbacks for requests must be called in the same order as requests were dispatched. If the driver is asynchronous, the behavior of **crypto_dispatch_async**() is identical to that of **crypto_dispatch**().

crypto_dispatch_batch() allows the caller to collect a batch of requests and submit them to the driver at the same time. This allows hardware drivers to optimize the scheduling of request processing and batch completion interrupts. A batch is submitted to the driver by invoking the driver's process method on each request, specifying CRYPTO_HINT_MORE with each request except for the last. The *flags* parameter to **crypto_dispatch_batch**() is currently ignored.

Callback function scheduling is simpler than request scheduling. Callbacks can either be invoked synchronously from **crypto_done**(), or they can be queued to a pool of worker threads. This pool of worker threads is also sized to provide one worker thread for each CPU by default. Note that a callback function invoked synchronously from **crypto_done**() must follow the same restrictions placed on threaded interrupt handlers.

By default, callbacks are invoked asynchronously by a worker thread. If CRYPTO_F_CBIMM is set,

the callback is always invoked synchronously from **crypto_done**(). If CRYPTO_F_CBIFSYNC is set, the callback is invoked synchronously if the request was processed by a software driver or asynchronously if the request was processed by a hardware driver.

If a request was scheduled to the taskqueue with CRYPTO_ASYNC_ORDERED, callbacks are always invoked asynchronously ignoring CRYPTO_F_CBIMM and CRYPTO_F_CBIFSYNC. This flag is used by IPsec to ensure that decrypted network packets are passed up the network stack in roughly the same order they were received.

Other Request Fields

In addition to the fields and flags enumerated above, *struct cryptop* includes the following:

crp_session A reference to the active session. This is set when the request is created by **crypto_getreq**() and should not be modified. Drivers can use this to fetch driver-specific session state or session parameters.

crp_etype Error status. Either zero on success, or an error if a request fails. Set by drivers prior to completing a request via **crypto_done**().

crp_flags A bitmask of flags. The following flags are available in addition to flags discussed previously:

CRYPTO_F_DONE Set by *crypto_done* before calling *crp_callback*. This flag is not very useful and will likely be removed in the future. It can only be safely checked from the callback routine at which point it is always set.

crp_cipher_key Pointer to a request-specific encryption key. If this value is not set, the request uses the session encryption key.

crp_auth_key Pointer to a request-specific authentication key. If this value is not set, the request uses the session authentication key.

crp_opaque An opaque pointer. This pointer permits users of the cryptographic framework to store information about a request to be used in the callback.

crp_callback Callback function. This must point to a callback function of type *void (*)(struct cryptop *)*. The callback function should inspect *crp_etype* to determine the status of the completed operation. It should also arrange for the request to be freed via **crypto_freereq**().

crp_olen Used with compression and decompression requests to describe the updated length of the payload region in the data buffer.

If a compression request increases the size of the payload, then the data buffer is unmodified, the request completes successfully, and *crp_olen* is set to the size the compressed data would have used. Callers can compare this to the payload region length to determine if the compressed data was discarded.

RETURN VALUES

crypto_dispatch() returns an error if the request contained invalid fields, or zero if the request was valid.

crypto_getreq() returns a pointer to a new request structure on success, or NULL on failure. NULL can only be returned if M_NOWAIT was passed in *how*.

SEE ALSO

ipsec(4), crypto(7), crypto(9), crypto_session(9), mbuf(9), uio(9)

BUGS

Not all drivers properly handle mixing session and per-request keys within a single session. Consumers should either use a single key for a session specified in the session parameters or always use per-request keys.