NAME

crypto, **cryptodev** - user-mode access to hardware-accelerated cryptography

SYNOPSIS

device crypto
device cryptodev

#include <sys/ioctl.h>
#include <sys/time.h>
#include <crypto/cryptodev.h>

DESCRIPTION

The **crypto** driver gives user-mode applications access to hardware-accelerated cryptographic transforms as implemented by the crypto(9) in-kernel interface.

The */dev/crypto* special device provides an ioctl(2) based interface. User-mode applications open the special device and then issue ioctl(2) calls on the descriptor. User-mode access to */dev/crypto* is controlled by the **kern.cryptodevallowsoft** sysctl(8) variable. If this variable is zero, then user-mode sessions are only permitted to use cryptography coprocessors.

THEORY OF OPERATION

Use of the device requires a basic series of steps:

1. Open the */dev/crypto* device.

2. Create a session with CIOCGSESSION or CIOCGSESSION2. Applications will require at least one symmetric session. Since cipher and MAC keys are tied to sessions, many applications will require more.

3. Submit requests, synchronously with CIOCCRYPT or CIOCCRYPTAEAD.

4. Optionally destroy a session with CIOCFSESSION.

5. Close the */dev/crypto* device. This will automatically close any remaining sessions associated with the file desriptor.

SYMMETRIC-KEY OPERATION

cryptodev provides a context-based API to traditional symmetric-key encryption (or privacy) algorithms, keyed and unkeyed one-way hash (HMAC and MAC) algorithms, encrypt-then-authenticate (ETA) fused operations, and authenticated encryption with additional data (AEAD) operations. For ETA

operations, drivers perform both a privacy algorithm and an integrity-check algorithm in a single pass over the data: either a fused encrypt/HMAC-generate operation, or a fused HMAC-verify/decrypt operation. Similarly, for AEAD operations, drivers perform either an encrypt/MAC-generate operation or a MAC-verify/decrypt operation.

The algorithm(s) and key(s) to use are specified when a session is created. Individual requests are able to specify per-request initialization vectors or nonces.

Algorithms

For a list of supported algorithms, see crypto(7).

IOCTL Request Descriptions

CIOCFINDDEV *struct crypt_find_op *fop*

```
struct crypt_find_op {
    int     crid;       /* driver id + flags */
    char    name[32];   /* device/driver name */
};
```

If *crid* is -1, then find the driver named *name* and return the id in *crid*. If *crid* is not -1, return the name of the driver with *crid* in *name*. In either case, if the driver is not found, ENOENT is returned.

CIOCGSESSION *struct session_op *sessp*

```
struct session_op {
    uint32_t cipher;  /* e.g. CRYPTO_AES_CBC */
    uint32_t mac;     /* e.g. CRYPTO_SHA2_256_HMAC */

    uint32_t keylen;  /* cipher key */
    const void *key;
    int mackeylen;    /* mac key */
    const void *mackey;

    uint32_t ses;     /* returns: ses # */
};
```

Create a new cryptographic session on a file descriptor for the device; that is, a persistent object specific to the chosen privacy algorithm, integrity algorithm, and keys specified in *sessp*. The special value 0 for either privacy or integrity is reserved

to indicate that the indicated operation (privacy or integrity) is not desired for this session. ETA sessions specify both privacy and integrity algorithms. AEAD sessions specify only a privacy algorithm.

Multiple sessions may be bound to a single file descriptor. The session ID returned in *sessp->ses* is supplied as a required field in the operation structure *crypt_op* for future encryption or hashing requests.

For non-zero privacy algorithms, the privacy algorithm must be specified in *sessp->cipher*, the key length in *sessp->keylen*, and the key value in the octets addressed by *sessp->key*.

For keyed one-way hash algorithms, the one-way hash must be specified in *sessp->mac*, the key length in *sessp->mackey*, and the key value in the octets addressed by *sessp->mackeylen*.

Support for a specific combination of fused privacy and integrity-check algorithms depends on whether the underlying hardware supports that combination. Not all combinations are supported by all hardware, even if the hardware supports each operation as a stand-alone non-fused operation.

CIOCGSESSION2 *struct session2_op *sessp*

```
struct session2_op {
    uint32_t cipher;  /* e.g. CRYPTO_AES_CBC */
    uint32_t mac;     /* e.g. CRYPTO_SHA2_256_HMAC */

    uint32_t keylen;  /* cipher key */
    const void *key;
    int mackeylen;    /* mac key */
    const void *mackey;

    uint32_t ses;     /* returns: ses # */
    int      crid;              /* driver id + flags (rw) */
    int ivlen;                  /* length of nonce/IV */
    int maclen;                 /* length of MAC/tag */
    int      pad[2];            /* for future expansion */
};
```

This request is similar to CIOGSESSION but adds additional fields.

sessp->crid requests either a specific crypto device or a class of devices (software vs hardware).

sessp->ivlen specifies the length of the IV or nonce supplied with each request. If this field is set to zero, the default IV or nonce length is used.

sessp->maclen specifies the length of the MAC or authentication tag supplied or computed by each request. If this field is set to zero, the full MAC is used.

The *sessp->pad* field must be initialized to zero.

CIOCCRYPT *struct crypt_op *cr_op*

```
struct crypt_op {
    uint32_t ses;
    uint16_t op;        /* e.g. COP_ENCRYPT */
    uint16_t flags;
    u_int len;
    const void *src;
    void *dst;
    void *mac;                  /* must be large enough for result */
    const void *iv;
};
```

Request an encryption/decryption (or hash) operation. To encrypt, set *cr_op->op* to COP_ENCRYPT. To decrypt, set *cr_op->op* to COP_DECRYPT. The field *cr_op->len* supplies the length of the input buffer; the fields *cr_op->src*, *cr_op->dst*, *cr_op->mac*, *cr_op->iv* supply the addresses of the input buffer, output buffer, one-way hash, and initialization vector, respectively.

If a session is using either fused encrypt-then-authenticate or an AEAD algorithm, decryption operations require the associated hash as an input. If the hash is incorrect, the operation will fail with EBADMSG and the output buffer will remain unchanged.

CIOCCRYPTAEAD *struct crypt_aead *cr_aead*

```
struct crypt_aead {
    uint32_t ses;
    uint16_t op;        /* e.g. COP_ENCRYPT */
    uint16_t flags;
```

```
    u_int len;
    u_int aadlen;
    u_int ivlen;
    const void *src;
    void *dst;
    const void *aad;  /* additional authenticated data */
    void *tag;                  /* must fit for chosen TAG length */
    const void *iv;
};
```

The CIOCCRYPTAEAD is similar to the CIOCCRYPT but provides additional data in *cr_aead->aad* to include in the authentication mode.

CIOCFSESSION *u_int32_t ses_id*
Destroys the session identified by *ses_id*.

SEE ALSO

aesni(4), hifn(4), ipsec(4), padlock(4), safe(4), crypto(7), geli(8), crypto(9)

HISTORY

The **crypto** driver first appeared in OpenBSD 3.0. The **crypto** driver was imported to FreeBSD 5.0.

BUGS

Error checking and reporting is weak.

The values specified for symmetric-key key sizes to CIOCGSESSION must exactly match the values expected by opencrypto(9). The output buffer and MAC buffers supplied to CIOCCRYPT must follow whether privacy or integrity algorithms were specified for session: if you request a non-NULL algorithm, you must supply a suitably-sized buffer.