NAME

ieee80211_crypto - 802.11 cryptographic support

SYNOPSIS

#include <net80211/ieee80211_var.h>

void
ieee80211_crypto_register(*const struct ieee80211_cipher **);

void
ieee80211_crypto_unregister(*const struct ieee80211_cipher **);

int
ieee80211_crypto_available(*int cipher*);

void
ieee80211_notify_replay_failure(*struct ieee80211vap *, const struct ieee80211_frame *, const struct ieee80211_key *, uint64_t rsc, int tid*);

void
ieee80211_notify_michael_failure(*struct ieee80211vap *, const struct ieee80211_frame *, u_int keyix*);

int
ieee80211_crypto_newkey(*struct ieee80211vap *, int cipher, int flags, struct ieee80211_key **);

int
ieee80211_crypto_setkey(*struct ieee80211vap *, struct ieee80211_key **);

int
ieee80211_crypto_delkey(*struct ieee80211vap *, struct ieee80211_key **);

void
ieee80211_key_update_begin(*struct ieee80211vap **);

void
ieee80211_key_update_end(*struct ieee80211vap **);

void
ieee80211_crypto_delglobalkeys(*struct ieee80211vap **);

void
ieee80211_crypto_reload_keys(*struct ieee80211com **);

*struct ieee80211_key **
ieee80211_crypto_encap(*struct ieee80211_node *, struct mbuf **);

*struct ieee80211_key **
ieee80211_crypto_decap(*struct ieee80211_node *, struct mbuf *, int flags*);

int
ieee80211_crypto_demic(*struct ieee80211vap *, struct ieee80211_key *, struct mbuf *, int force*);

int
ieee80211_crypto_enmic(*struct ieee80211vap *, struct ieee80211_key *, struct mbuf *, int force*);

DESCRIPTION

The **net80211** layer includes comprehensive cryptographic support for 802.11 protocols. Software implementations of ciphers required by WPA and 802.11i are provided as well as encap/decap processing of 802.11 frames. Software ciphers are written as kernel modules and register with the core crypto support. The cryptographic framework supports hardware acceleration of ciphers by drivers with automatic fall-back to software implementations when a driver is unable to provide necessary hardware services.

CRYPTO CIPHER MODULES

net80211 cipher modules register their services using **ieee80211_crypto_register**() and supply a template that describes their operation. This *ieee80211_cipher* structure defines protocol-related state such as the number of bytes of space in the 802.11 header to reserve/remove during encap/decap and entry points for setting up keys and doing cryptographic operations.

Cipher modules can associate private state to each key through the *wk_private* structure member. If state is setup by the module it will be called before a key is destroyed so it can reclaim resources.

Crypto modules can notify the system of two events. When a packet replay event is recognized **ieee80211_notify_replay_failure**() can be used to signal the event. When a TKIP Michael failure is detected **ieee80211_notify_michael_failure**() can be invoked. Drivers may also use these routines to signal events detected by the hardware.

CRYPTO KEY MANAGEMENT

The **net80211** layer implements a per-vap 4-element "global key table" and a per-station "unicast key" for protocols such as WPA, 802.1x, and 802.11i. The global key table is designed to support legacy

WEP operation and Multicast/Group keys, though some applications also use it to implement WPA in station mode. Keys in the global table are identified by a key index in the range 0-3. Per-station keys are identified by the MAC address of the station and are typically used for unicast PTK bindings.

net80211 provides ioctl(2) operations for managing both global and per-station keys. Drivers typically do not participate in software key management; they are involved only when providing hardware acceleration of cryptographic operations.

ieee80211_crypto_newkey() is used to allocate a new **net80211** key or reconfigure an existing key. The cipher must be specified along with any fixed key index. The **net80211** layer will handle allocating cipher and driver resources to support the key.

Once a key is allocated it's contents can be set using **ieee80211_crypto_setkey**() and deleted with **ieee80211_crypto_delkey**() (with any cipher and driver resources reclaimed).

ieee80211_crypto_delglobalkeys() is used to reclaim all keys in the global key table for a vap; it typically is used only within the **net80211** layer.

ieee80211_crypto_reload_keys() handles hardware key state reloading from software key state, such as required after a suspend/resume cycle.

DRIVER CRYPTO SUPPORT

Drivers identify ciphers they have hardware support for through the *ic_cryptocaps* field of the *ieee80211com* structure. If hardware support is available then a driver should also fill in the iv_key_alloc, iv_key_set, and iv_key_delete methods of each *ieee80211vap* created for use with the device. In addition the methods iv_key_update_begin and iv_key_update_end can be setup to handle synchronization requirements for updating hardware key state.

When **net80211** allocates a software key and the driver can accelerate the cipher operations the iv_key_alloc method will be invoked. Drivers may return a token that is associated with outbound traffic (for use in encrypting frames). Otherwise, e.g. if hardware resources are not available, the driver will not return a token and **net80211** will arrange to do the work in software and pass frames to the driver that are already prepared for transmission.

For receive, drivers mark frames with the M_WEP mbuf flag to indicate the hardware has decrypted the payload. If frames have the IEEE80211_FC1_PROTECTED bit marked in their 802.11 header and are not tagged with M_WEP then decryption is done in software. For more complicated scenarios the software key state is consulted; e.g. to decide if Michael verification needs to be done in software after the hardware has handled TKIP decryption.

Drivers that manage complicated key data structures, e.g. faulting software keys into a hardware key cache, can safely manipulate software key state by bracketing their work with calls to **ieee80211_key_update_begin**() and **ieee80211_key_update_end**(). These calls also synchronize hardware key state update when receive traffic is active.

SEE ALSO

ioctl(2), wlan_ccmp(4), wlan_tkip(4), wlan_wep(4), ieee80211(9)