SecY

common_port,
controlled_port,
uncontrolled_port

Port   3
<<In the case of the Controlled Port the "Interface Group" will roll up from the general (verification, generation) plus the statistics
accumulated for the channnels plus the statistics for the currently active SAs. Specifiy this.>>

verification

Verification
struct verificationControls{   bool   controlledReceivesAll, replayProtect, validateReceivedFrames;};
struct verificationStatistics{   count   InXcastPktsNoTag, InXcastPktsBadTag, InXcastPktsUnknownSCI,
                                               InXcastPktsNotReceived, InXcastPktsReplayed, InXcastPktsInvalid, InXcastPktsMIsordered,
                                               InXcastPktsUntagged, InXcastPktsUnchecked;};

receive_channels

Receive_channel
const SCI sci;
struct rxChannelStatistics{   count   InXcastPktsNotReceived, InXcastPktsReplayed, InXcastPktsInvalid,
                                            InXcastPktsMIsordered;};

rxa[0-3]

Receive_association   4
Data_key *data_key; PN next_PN; bool inUse;
struct rxSAStatistics{   count   InXcastPktsNotReceived, InXcastPktsReplayed, InXcastPktsInvalid,
                                        InXcastPktsMisordered;};
use_key(Data_key *data_key); stop_using(); start_using();
set_next_PN(); PN get_next_PN(); update_next_PN();

generation

Generation
struct generationControls{   bool   controlledSendsTagged;};
struct generationStatistics{   count   OutPktsPnExhausted, OutPktsTooLong, OutXcastPktsProtected, OutXcastPktsUntagged;};

transmit_channel

Transmit_channel
const SCI sci;
AN encodingSA, encipheringSA;

start_encoding(AN encodingSA); stop_encoding();

txa[0-3]

Transmit_association   4
Data_key *data_key; PN nextPN; bool inUse;
struct txSAStatistics{ count   OutPktsPnExhausted, OutXcastPktsProtected, OutXcastPktsUntagged; };

use_key(Data_key *data_key);
set_next_PN(); PN get_next_PN(); update_next_PN();

current_ciphersuite

Ciphersuite
<<cipher suite identifier>>
Data_key *install_key(SAK sak, bool transmitter, bool receiver, void *manager_ref);

saks

Data_key // KSP instance
const SAK sak
const bool transmitter
const bool receiver
const void * manager_ref

ciphersuites

Ciphersuite
<<cipher suite identifier>>
<<cipher suite capabilities>>

· (|) · · **Uncontrolled Port** · · · (|) · **Controlled Port** · · · ·

ctrl.InXcastPktsUnchecked++

remove_secTAG()

sa->next_PN = rx.pn + 1

ctrl.InXcastPktsUntagged++ ◄ if (untagged(rx))

sa->InXcastPktsOrdered++   sa->InXcastPktsMisordered++

uncntrl.ifInXcastPkts++

if (replayProtect && (rx.pn < sa->next_PN))

if (replayProtect && (rx.pn < sa->next_PN) || (sa->next_PN == 0))

if ((!VALID) &&
(validateReceiveFrames || frame.encrypted))   sa->InXcastPktsInvalid++

remove_secTAG()

Validate

if (replayProtect &&
(rx.pn < sa->next_PN) || (sa->next_PN == 0))   sa->InXcastPktsReplayed++ ◄

if (!sa->inUse)   sa->InXcastPktsNotReceived++

sa = &sc.rxa[rx.an]

if ((sc = find(receive_channels, rx.sci)) == 0)   ctrl.InXcastPktsUnknownSCI++

if (invalid_tag(rx)))   ctrl.InXcastPktsBadTag++ ◄ if (invalid_tag(rx)))

if (untagged(rx))   ctrl.InXcastPktsNoTag++

if (controlledReceivesAll)

if (!controlledReceivesAll)

rx = received frame

· · · · · · · **Common Port** · (|) ·

NOTE-- the notation xxXcastPktsxx++ means increment xxUcastPktsxx, xxMulticastPktsxx, or
xxBroadcastPktsxx as appropriate for the frame (unicast, mutlicast, or broadcast) considered.

( ) · · **Uncontrolled Port** · · · · · **( )** · **Controlled Port** · · · ·

if (controlledSendsTagged)   if (!controlledSendsTagged)

sa = &transmit_channel.txa[encodingSA]

if (sa->next_PN) == 0)   sa->OutPktsPnExhausted++

if (AlwaysIncludeSCI || ((rcv_channel_cnt > 1) || !(validateReceiveFrames || operP2PMAC)))
  add_secTAG(encodingSA, sa->next_PN, sci);
else
  add_secTAG(encodingSA, sa->next_PN);

Protect

if (tx->len > common_port->max_len)   ctrl.OutPktsTooLong++

sa.OutXcastPktsProtected++   sa.OutXcastPktsUntagged++

uncntrl.ifOutXcastPkts++

· · · · · · **Common Port** · **( )** ·