

SAI MACsec API Proposal

1.4

|  |  |
| --- | --- |
| **Title** | **SAI MACsec API Proposal** |
| **Authors** | **Arista Networks, Inc.** |
| **Status** | **Draft** |
| **Type** |  |
| **Created** | **02/05/2020** |
| **Version** | **1.4** |

**Table of Contents**

[**License**](#_heading=h.1fob9te) **4**

[**Scope & Overview**](#_heading=h.g99jx3ncxpjv) **5**

[Block Diagram](#_heading=h.t2p14zw6bbb) **5**

[**Object Relationships**](#_heading=h.4d34og8) **6**

[**Function call sequences**](#_heading=h.2s8eyo1) **9**

[Enable MACsec on a port](#_heading=h.nufqtif52c5r) 9

[Create ingress/egress SA](#_heading=h.mddh2zpidngg) 11

[Add a SC with 1 SA for a port](#_heading=h.77xp2sx5l9mm) 12

[Remove an SA](#_heading=h.vbeg2ot4bd9p) 13

[Replace egress SA](#_heading=h.devy8otm4skw) 13

[Replace ingress SA](#_heading=h.35pcg5uxmut6) 14

[Remove a SC](#_heading=h.rbrglmh8d3ke) 14

[Disable MACsec on a port](#_heading=h.alnmk1hfn10s) 15

[Periodic activities](#_heading=h.p1nlr9yy22y5) 15

[**Notes**](#_heading=h.12r7gxk51c67) **16**

[Unsupported Features](#_heading=h.erj6mj3r76mo) 16

[Hidden Complexities](#_heading=h.2ag6v5yq9l4e) 16

# License

Contributions to this Specification are made under the terms and conditions set forth in Open Web Foundation Contributor License Agreement (“OWF CLA 1.0”) (“Contribution License”) by:

**Arista Networks, Inc**

 Usage of this Specification is governed by the terms and conditions set forth in the Open Web Foundation Final Specification Agreement (“OWFa 1.0”).

**Note**: The following clarifications, which distinguish technology licensed in the Contribution License and/or Specification License from those technologies merely referenced (but not licensed), were accepted by the Incubation Committee of the OCP:

**None**

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Scope & Overview

Scope:

This document defines the technical specifications for the API used to support 802.1ae MAC Security (MACsec) in Open Compute Project Switch Abstraction Interface (SAI). MAC Security Key Agreement Entity (802.1ae KaY), responsible for obtaining AES key and other parameters, is outside the scope of this document and the MACsec API.

Overview:

The switching hardware consists of network interfaces connected to a forwarding element, such as a switching ASIC. Some switching hardware also include Phy ASIC(s) that interconnect network interfaces and forwarding element interfaces. Each Phy ASIC supports one or more network interfaces. SAI is used for each such Phy ASIC (doc/macsec-gearbox/SAI\_Gearbox\_API\_Proposal-v1.0.docx). A sai\_switch object is used to interface to either a forwarding element or a Phy.

This SAI MACsec API provides a software interface for 802.1ae MACSec Entities (SecY) associated with some or all ports of a sai\_switch object.

MACsec provides data security (confidentiality, integrity, authenticity) to Ethernet port(s). MACsec uses 2 distinct datapath modules:

1. The egress module adds SecTag, encrypts packets and adds a trailer containing ICV (Integrity Check Value).
2. The ingress module decrypts the packet, checks the authentication and integrity (using ICV) and removes the SecTag.

# Block Diagram

MACsec functionality may be integrated in either the Phy chip or the Forwarding Element.

Each Phy chip in the diagram contains MACsec functionality that services a set of network ports (a.k.a. Phy line-side ports). Some resources (e.g. encryption keys) are not shared between network ports. Some other resources are shared between multiple network ports e.g. function like parsing, match-criteria for control-packets, etc.

The Forwarding Element in the diagram is shown to contain similar functionality for another (distinct) set of ports.



# Object Relationships

The SAI MACsec API provides an interface to programming the MACsec functions:

1. Two sai\_macsec objects (one for ingress and one for egress) are bound to a sai\_switch object.
2. Two sai\_macsec\_port objects (one for ingress and one for egress) are bound to each MACsec enabled sai\_port object. In case of a Phy, the binding should be to a line-side port (SAI\_PORT\_CONNECTOR\_ATTR\_LINE\_SIDE\_PORT\_ID).
3. Two acl\_table objects (one for ingress and one for egress) are bound to the same sai\_port object.



There are 2 specific ACL stages (SAI\_ACL\_STAGE\_INGRESS\_MACSEC and SAI\_ACL\_STAGE\_EGRESS\_MACSEC) used for MACsec.

These ACL tables classify packets based on the following fields:

SAI\_ACL\_TABLE\_ATTR\_FIELD\_DST\_MAC (used to identify 802.1x packets),

SAI\_ACL\_TABLE\_ATTR\_FIELD\_ETHER\_TYPE (used to identify EAPOL packets),

SAI\_ACL\_TABLE\_ATTR\_FIELD\_HAS\_VLAN\_TAG (needed only for vlan-based SC),

SAI\_ACL\_TABLE\_ATTR\_FIELD\_OUTER\_VLAN\_ID (needed only for vlan-based SC),

SAI\_ACL\_ENTRY\_ATTR\_FIELD\_MACSEC\_SCI (only for ingress, in some implementations as explained later).[[1]](#footnote-0)

Each acl\_entry in such a table, should be assigned a unique ordinal priority value.

Each acl\_entry in these acl\_tables uses one of the following actions:

1. SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION using one of 2 enums:
   1. SAI\_PACKET\_ACTION\_DROP to drop packets
   2. SAI\_PACKET\_ACTION\_FORWARD to allow packets without MACsec processing. Packets forwarded without MACsec processing should include MACsec control packets (e.g. EAPOL packets).
2. SAI\_ACL\_ENTRY\_ATTR\_ACTION\_MACSEC\_FLOW for MACsec processing. For egress, a macsec\_flow is always associated with one IEEE 802.1ae SecY. For ingress, one IEEE 802.1ae SecY can be associated with one or more SC (Secure Channel). Depending on hardware implementation, an ingress macsec\_flow object can be associated with:
   1. One IEEE 802.1ae SecY, when SAI\_ACL\_ENTRY\_ATTR\_FIELD\_MACSEC\_SCI is not used
   2. One IEEE 802.1ae SC, when SAI\_ACL\_ENTRY\_ATTR\_FIELD\_MACSEC\_SCI is used



The ACL tables for a port are minimally as follows:

1. For a MACsec-disabled port, there is no ACL table - all packets are forwarded.
2. Port-level MACsec needs an ACL table with at least 2 entries to differentiate control packets (with packet\_action=forward) from data packets (action=macsec\_flow). At least SAI\_ACL\_TABLE\_ATTR\_FIELD\_DST\_MAC and SAI\_ACL\_TABLE\_ATTR\_FIELD\_ETHER\_TYPE are needed to identify 8021.x EAPOL packets. In this case:
   1. For control packet high priority entry, the action is SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION with SAI\_PACKET\_ACTION\_FORWARD.
   2. For other packets (default low-priority entry), the action is:
      1. SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION with SAI\_PACKET\_ACTION\_DROP before SC and SA is established
      2. SAI\_ACL\_ENTRY\_ATTR\_ACTION\_MACSEC\_FLOW after SC and SA is established



Each SC has one or more associated SAs. Each SA has configuration parameters for its cipher suite (encryption key, etc). Each encrypted packet has a packet number (32-bit PN or 64-bit XPN). For each SA, the egress uses a counter to increment the packet number. During the lifetime of a SA, the packet number value used by the egress is periodically sent to the receiver(s) by the key agreement protocol. This ingress logic uses this value as the minimum acceptable value of packet number. The SA is periodically replaced before it exhausts all its packet numbers. The SC can operate uninterrupted for a long time, using rekeying i.e. using a series of overlapping SAs.

1. For egress, only one SA is active at a time. Rekeying involves switching the active SA and changing the SecTAG.AN field in the packet header.
2. For an ingress SC, multiple SAs can be simultaneously active, selected for each packet by SecTAG.AN field from the packet header.

For ingress, the SecTAG.SCI field from the packet header has to be matched to a programmed value. This API supports 2 implementation models:

1. If hardware matches SCI in ACL, the macsec\_flow maps to an IEEE 802.1ae SecY object. Multiple SCs (e.g. SC-21 and SC-22 in Figure 4) can be associated with such a macsec\_flow (e.g. Secure flow-2 in Figure 4). Then a specific value of SCI from the SecTAG in the packet is used to identify a specific SC for that macsec\_flow.
2. If hardware does not match SCI in ACL, there is 1:1 association between acl\_entry, macsec\_flow and macsec\_sc. In that case, for each IEEE 802.1ae SC associated with an IEEE 802.1ae SecY has its own macsec\_flow object.

# Function call sequences

The following sequences assume that a sai\_macsec object has already been created by a create\_macsec function call.

## Enable MACsec on a port

The steps are identical for ingress and egress. The following example is for ingress.

// Create ingress macsec\_port for port\_id1

sai\_object\_id\_t ingr\_macsec\_port\_id1 = 0ULL;

macsec\_attr\_list[0].id = SAI\_MACSEC\_ATTR\_MACSEC\_DIRECTION;

macsec\_attr\_list[0].value.s32 = SAI\_MACSEC\_DIRECTION\_INGRESS;

macsec\_attr\_list[1].id = SAI\_MACSEC\_ATTR\_PORT\_ID;

macsec\_attr\_list[1].value.oid = port\_id1;

saistatus = sai\_macsec\_api->create\_macsec\_port(&ingr\_macsec\_port\_id1, 2,

macsec\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Create ingress MACsec ACL table for port\_id1

sai\_object\_id\_t acl\_table\_id1 = 0ULL;

acl\_attr\_list[0].id = SAI\_ACL\_TABLE\_ATTR\_ACL\_STAGE;

acl\_attr\_list[0].value.s32 = SAI\_ACL\_STAGE\_INGRESS\_MACSEC;

acl\_attr\_list[1].id = SAI\_ACL\_TABLE\_ATTR\_ACL\_BIND\_POINT\_TYPE\_LIST;

acl\_attr\_list[1].value.objlist.count = 1;

acl\_attr\_list[1].value.objlist.list[0] = SAI\_ACL\_BIND\_POINT\_TYPE\_PORT;

acl\_attr\_list[2].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_DST\_MAC;

acl\_attr\_list[2].value.booldata = True;

acl\_attr\_list[3].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_ETHER\_TYPE;

acl\_attr\_list[3].value.booldata = True;

acl\_attr\_list[4].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_NO\_VLAN\_TAG;

acl\_attr\_list[4].value.booldata = True;

acl\_attr\_list[5].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_HAS\_VLAN\_TAG;

acl\_attr\_list[5].value.booldata = True;

acl\_attr\_list[6].id = SAI\_ACL\_TABLE\_ATTR\_FIELD\_OUTER\_VLAN\_ID;

acl\_attr\_list[6].value.booldata = True;

saistatus = sai\_acl\_api->create\_acl\_table(&acl\_table\_id1, 7, acl\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Create an ACL entry to permit EAPOL packets to 802.1x destination MAC address

const unsigned char MacMask[6] = {(char)(0xff), (char)(0xff),

(char)(0xff), (char)(0xff), (char)(0xff), (char)(0xff)};

sai\_object\_id\_t acl\_entry\_id1 = 0ULL;

acl\_attr\_list[0].id = SAI\_ACL\_ENTRY\_ATTR\_TABLE\_ID;

acl\_attr\_list[0].value.oid = acl\_table\_id1;

acl\_attr\_list[1].id = SAI\_ACL\_ENTRY\_ATTR\_PRIORITY;

acl\_attr\_list[1].value.u32 = 100;

acl\_attr\_list[2].id = SAI\_ACL\_ENTRY\_ATTR\_ADMIN\_STATE;

acl\_attr\_list[2].value.booldata = true;

acl\_attr\_list[3].id = SAI\_ACL\_ENTRY\_ATTR\_FIELD\_DST\_MAC;

acl\_attr\_list[3].value.aclfield.data.mac = dot1xDstMac;

acl\_attr\_list[3].value.aclfield.mask.mac = MacMask;

acl\_attr\_list[3].value.aclfield.enable = true;

acl\_attr\_list[4].id = SAI\_ACL\_ENTRY\_ATTR\_FIELD\_ETHER\_TYPE;

acl\_attr\_list[4].value.aclfield.data.u16 = EAPOL\_ETHER\_TYPE;

acl\_attr\_list[4].value.aclfield.mask.u16 = 0xFFFF;

acl\_attr\_list[4].value.aclfield.enable = true;

acl\_attr\_list[5].id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION;

acl\_attr\_list[5].value.aclaction.data.s32 = SAI\_PACKET\_ACTION\_FORWARD;

acl\_attr\_list[5].value.aclaction.enable = true;

saistatus = sai\_acl\_api->create\_acl\_entry(&acl\_entry\_id1, 6, acl\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Create an ACL entry to drop packets (to be later used for macsec\_flow)

sai\_object\_id\_t acl\_entry\_id2 = 0ULL;

acl\_attr\_list[0].id = SAI\_ACL\_ENTRY\_ATTR\_TABLE\_ID;

acl\_attr\_list[0].value.oid = acl\_table\_id2;

acl\_attr\_list[1].id = SAI\_ACL\_ENTRY\_ATTR\_PRIORITY;

acl\_attr\_list[1].value.u32 = 1;

acl\_attr\_list[2].id = SAI\_ACL\_ENTRY\_ATTR\_ADMIN\_STATE;

acl\_attr\_list[2].value.booldata = true;

acl\_attr\_list[3].id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION;

acl\_attr\_list[3].value.aclaction.data.s32 = SAI\_PACKET\_ACTION\_DROP;

acl\_attr\_list[3].value.aclaction.enable = true;

saistatus = sai\_acl\_api->create\_acl\_entry(&acl\_entry\_id2, 4, acl\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Bind this ACL table to port\_id1

port\_attr.id = SAI\_PORT\_ATTR\_INGRESS\_MACSEC\_ACL;

port\_attr.value.oid = acl\_table\_id1;

sai\_port\_api->set\_port\_attribute(port\_id1, port\_attr);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

## Create ingress/egress SA

Creating SA for ingress and egress, shown below, are very similar. Only the direction and XPN attributes are different.

// Create ingress macsec\_sa

sai\_object\_id\_t ingr\_macsec\_sa\_id11 = 0ULL;

macsec\_attr\_list[0].id = SAI\_MACSEC\_SC\_ATTR\_MACSEC\_DIRECTION;

macsec\_attr\_list[0].value.s32 = SAI\_MACSEC\_DIRECTION\_INGRESS;

macsec\_attr\_list[1].id = SAI\_MACSEC\_ATTR\_SC\_ID;

macsec\_attr\_list[1].value.macsecsc = ingr\_macsec\_sc\_id1;

macsec\_attr\_list[2].id = SAI\_MACSEC\_ATTR\_AN;

macsec\_attr\_list[2].value.macsecan = ingr\_macsec\_an11;

macsec\_attr\_list[3].id = SAI\_MACSEC\_ATTR\_MACSEC\_SAK;

macsec\_attr\_list[3].value.macsecsak = macsec\_sak1;

macsec\_attr\_list[4].id = SAI\_MACSEC\_ATTR\_MACSEC\_SALT;

macsec\_attr\_list[4].value.macsecsak = macsec\_salt1;

macsec\_attr\_list[5].id = SAI\_MACSEC\_ATTR\_MACSEC\_AUTH\_KEY;

macsec\_attr\_list[5].value.macsecauthkey = macsec\_auth\_key1;

saistatus = sai\_macsec\_api->create\_macsec\_sa(&ingr\_macsec\_sa\_id11, 6,

macsec\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Create egress macsec\_sa (only DIRECTION and XPN attributes different)

sai\_object\_id\_t egr\_macsec\_sa\_id11 = 0ULL;

macsec\_attr\_list[0].id = SAI\_MACSEC\_SC\_ATTR\_MACSEC\_DIRECTION;

macsec\_attr\_list[0].value.s32 = SAI\_MACSEC\_DIRECTION\_EGRESS;

macsec\_attr\_list[1].id = SAI\_MACSEC\_ATTR\_SC\_ID;

macsec\_attr\_list[1].value.macsecsc = egr\_macsec\_sc\_id1;

macsec\_attr\_list[2].id = SAI\_MACSEC\_ATTR\_AN;

macsec\_attr\_list[2].value.macsecan = egr\_macsec\_an11;

macsec\_attr\_list[3].id = SAI\_MACSEC\_ATTR\_MACSEC\_SAK;

macsec\_attr\_list[3].value.macsecsak = macsec\_sak1;

macsec\_attr\_list[4].id = SAI\_MACSEC\_ATTR\_MACSEC\_SALT;

macsec\_attr\_list[4].value.macsecsak = macsec\_salt1;

macsec\_attr\_list[5].id = SAI\_MACSEC\_ATTR\_MACSEC\_AUTH\_KEY;

macsec\_attr\_list[5].value.macsecauthkey = macsec\_auth\_key1;

saistatus = sai\_macsec\_api->create\_macsec\_sa(&egr\_macsec\_sa\_id11, 6,

macsec\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

## Add a SC with 1 SA for a port

For a single SC on a port, the steps are identical for ingress and egress. The following example is for ingress.

// Create ingress macsec\_flow

sai\_object\_id\_t ingr\_macsec\_flow\_id1 = 0ULL;

macsec\_attr\_list[0].id = SAI\_MACSEC\_FLOW\_ATTR\_MACSEC\_DIRECTION;

macsec\_attr\_list[0].value.s32 = SAI\_MACSEC\_DIRECTION\_INGRESS;

saistatus = sai\_macsec\_api->create\_macsec\_flow(&ingr\_macsec\_flow\_id1, 1,

macsec\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Create ingress macsec\_sc for that flow

sai\_object\_id\_t ingr\_macsec\_sc\_id1 = 0ULL;

macsec\_attr\_list[0].id = SAI\_MACSEC\_SC\_ATTR\_MACSEC\_DIRECTION;

macsec\_attr\_list[0].value.s32 = SAI\_MACSEC\_DIRECTION\_INGRESS;

macsec\_attr\_list[1].id = SAI\_MACSEC\_ATTR\_FLOW\_ID;

macsec\_attr\_list[1].value.oid = ingr\_macsec\_flow\_id1;

macsec\_attr\_list[2].id = SAI\_MACSEC\_ATTR\_MACSEC\_SCI;

macsec\_attr\_list[2].value.macsecsci = macsec\_sci1;

macsec\_attr\_list[3].id = SAI\_MACSEC\_ATTR\_MACSEC\_SSCI;

macsec\_attr\_list[3].value.macsecssci = macsec\_ssci1;

macsec\_attr\_list[4].id = SAI\_MACSEC\_SC\_ATTR\_MACSEC\_XPN64\_ENABLE;

macsec\_attr\_list[4].value.booldata = true;

saistatus = sai\_macsec\_api->create\_macsec\_sc(&ingr\_macsec\_sc\_id1, 5,

macsec\_attr\_list);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

Then, as described in earlier section, create ingress macsec\_sa. And finally:

// Change action from packet-action to macsec\_flow for an already created ACL entry

acl\_attr.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_MACSEC\_FLOW;

acl\_attr.value.aclaction.parameter.oid = ingr\_macsec\_flow\_id1;

acl\_attr.value.aclaction.enable = true;

sai\_acl\_api->set\_acl\_entry\_attribute(acl\_entry\_id2, acl\_attr);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

acl\_attr.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION;

acl\_attr.value.aclaction.enable = false;

sai\_acl\_api->set\_acl\_entry\_attribute(acl\_entry\_id2, acl\_attr);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

## Remove an SA

The steps are identical for ingress and egress. The following example is for egress.

// Collect last statistics for SA

saistatus = sai\_macsec\_api->get\_macsec\_sa\_stats\_ext(&egr\_macsec\_sa\_id11,

SA\_STATS\_COUNT, &macsec\_counter\_list, SAI\_MACSEC\_ATTR\_STATS\_MODE);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

UPDATE\_EGRESS\_SA\_STATISTICS(macsec\_counter\_list);

// Remove old SA

saistatus = sai\_macsec\_api->remove\_macsec\_sa(&egr\_macsec\_sa\_id11);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

## Replace egress SA

For egr\_macsec\_sc\_id1, replace egr\_macsec\_sa\_id11 with egr\_macsec\_sa\_id12. When egr\_macsec\_sa\_id12 is created (as described earlier), it automatically replaces egr\_macsec\_sa\_id11 as active SA.

Then, as described earlier, collect statistics for old SA and remove old SA.

## Replace ingress SA

The goal is, for ingr\_macsec\_sc\_id1, replace ingr\_macsec\_sa\_id11 with ingr\_macsec\_sa\_id12. The steps are:

1. As described earlier, create a new SA (ingr\_macsec\_sa\_id12). This automatically allows the new ingress SA to receive packet.
2. Then, 0.5 second after actually receiving packets with the new SA, collect statistics for old SA and remove old SA, as described earlier.

## Remove a SC

The steps are identical for ingress and egress. The following example is for ingress.

In this case, ingr\_macsec\_flow\_id1 is associated with acl\_entry\_id2 and ingr\_macsec\_sc\_id1. And ingr\_macsec\_sc\_id1 uses two SAs: ingr\_macsec\_sa\_id11 and ingr\_macsec\_sc\_id12. Removing this SC involves the following steps.

// Change associated ACL entry action from macsec\_flow to packet\_action drop

acl\_attr.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_PACKET\_ACTION;

acl\_attr.value.oid = SAI\_PACKET\_ACTION\_DROP;

acl\_attr.value.aclaction.enable = true;

sai\_acl\_api->set\_acl\_entry\_attribute(acl\_entry\_id2, acl\_attr);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

acl\_attr.id = SAI\_ACL\_ENTRY\_ATTR\_ACTION\_MACSEC\_FLOW;

acl\_attr.value.aclaction.parameter.oid = ingr\_macsec\_flow\_id1;

acl\_attr.value.aclaction.enable = false;

sai\_acl\_api->set\_acl\_entry\_attribute(acl\_entry\_id2, acl\_attr);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

Then, as described earlier, collect last statistics and remove SA, iterating over all SA for this SC (which is ingr\_macsec\_sa\_id11 and ingr\_macsec\_sc\_id12, in this case). And finally:

// Collect last statistics and remove macsec\_sc

saistatus = sai\_macsec\_api->get\_macsec\_sc\_stats\_ext(&ingr\_macsec\_sc\_id1, SC\_STATS\_COUNT,

&macsec\_counter\_list, SAI\_MACSEC\_ATTR\_STATS\_MODE);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

UPDATE\_INGRESS\_SC\_STATISTICS(macsec\_counter\_list);

saistatus = sai\_macsec\_api->remove\_macsec\_sc(&ingr\_macsec\_sc\_id1);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Remove macsec\_flow

saistatus = sai\_macsec\_api->remove\_macsec\_flow(&ingr\_macsec\_flow\_id1);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

## Disable MACsec on a port

The steps are identical for ingress and egress MACsec. The following example is for ingress MACsec.

// Unbind MACsec ingress ACL table from port\_id1

port\_attr.id = SAI\_PORT\_ATTR\_INGRESS\_MACSEC\_ACL;

port\_attr.value.oid = SAI\_NULL\_OBJECT\_ID;;

sai\_port\_api->set\_port\_attribute(port\_id1, port\_attr);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

Then, as described earlier, remove all ingress SC associated with this port.

// Remove ACL table and all its entries

saistatus = sai\_acl\_api->remove\_acl\_entry(&acl\_entry\_id1);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

saistatus = sai\_acl\_api->remove\_acl\_table(&acl\_table\_id1);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

// Remove macsec\_port

saistatus = sai\_macsec\_api->remove\_macsec\_port(&ingr\_macsec\_port\_id1);

if (saistatus != SAI\_STATUS\_SUCCESS) {

return saistatus;

}

## Periodic activities

1. Read statistics
2. Read egress packet number, SAI\_MACSEC\_SECURITY\_ASSOCIATION\_ATTR\_XPN. The Key Agreement Entity sends this value to the MACsec peer.
3. Set minimum acceptable packet number, SAI\_MACSEC\_SECURITY\_ASSOCIATION\_ATTR\_MINIMUM\_XPN, for ingress. The Key Agreement Entity sets this to the value provided by MACsec peer.

# Notes

## Unsupported Features

1. Packet parsing beyond MAC header and VLAN tags.
2. The SAI MACsec API does not support any interrupt or event notification to NOS. Possible uses of notification could include:
   1. Egress packet number crossing threshold. The last used egress packet number can be polled or indirectly inferred from periodically collected egress packet statistics.
   2. A new packet SA value observed at ingress. This can be polled to determine when to remove the old SA.
   3. External Dependencies

The MACsec API makes use of AES key and other parameters obtained by MAC Security Key Agreement Entity (802.1ae KaY). It is the responsibility of the application (NOS or Controller) to implement the MAC Security Key Agreement Entity (802.1ae KaY).

## Hidden Complexities

Some implementation complexities are hidden to simplify the SAI MACsec API:

1. The SAI MACsec API exposes MACsec-specific packet-matching logic as as per-port ACL tables. But since those likely use common resources, the resource availability is tracked per sai\_macsec object.
2. The SAI MACsec API reports the availability of general-purpose acl\_entry resources. But that does not include specialized hardware resources used to support control packets and default action for data packets. The driver may use priority, fields and action of acl\_entry to identify acl\_entries which can be mapped to such specialized hardware. For such acl\_entries, this API may not show any change in resource availability, and the control packet and default action for data packets may appear to "come for free".

1. This matches on the SCI field from the 802.1ae SecTag header. [↑](#footnote-ref-0)