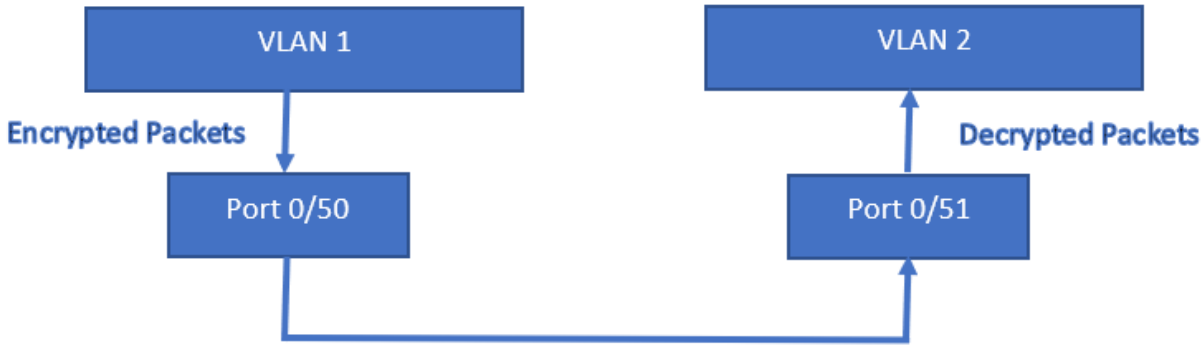
Topology:



Setup:

Port 0/50 is VLAN 1 untagged member
Port 0/51 is VLAN 2 untagged member
MACsec is enabled on Port 0/50 egress side
MACsec is enabled on Port 0/51 ingress side

Procedure:

Send plain-text packet from CPU to Port 0/50.
Capture Port 0/50 egress packet and verify if the packets are encrypted.
Capture Port 0/51 ingress packet and verify if the packets are decrypted.

Port VLAN Configuration:

Add port 0/51 to VLAN 2.
Port 0/50 is in VLAN 1 (by default).

```
Console(config)# interface vlan device 0 vid 2
Console(config-vlan)# exit
Console(config)# interface ethernet 0/51
Console(config-if)# switchport allowed vlan add 2 untagged
Console(config-if)# switchport pvid 2
Console(config-if)# end
Console# show vlan device 0

VLAN                          Ports                              Tag      MAC-Learning   FDB-mode
-----  ------------------------------------------------------  --------  -------------- ---------

1       0/0-53                                                 untagged    Control        FID
2       0/51                                                   untagged    Control        FID
```

PCL Configuration for Packet captured:

Configure a PCL rule to mirror packets to CPU.
Attach the PCL to port 0/50 and 0/51.

```
Console# config
Console(config)# access-list device 0 ingress pcl-ID 0
Console(config-acl)# rule-id 0 action mirror-to-cpu
Console(config-acl)# exit
Console(config)#
Console(config)# interface range ethernet 0/50,51
Console(config-if)# service-acl pcl-ID 0 lookup 0
Console(config-if)# end
```

Make sure the packets sent on port 0/50 are captured as expected (with CPU tag)

```
Console# traffic
Console(traffic)# cpu rx dump
Console(traffic)# send port 0/50 data 0x00112233445500112233466
dxChNetIfRxPacketGet: device[0] queue[0] [80] bytes (include DSA bytes !)
0x0000 : 00 11 22 33 44 55 00 11 22 33 44 66 10 9f 30 02
0x0010 : 80 03 c4 c0 80 00 01 98 00 00 00 00 00 00 00 00
0x0020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0040 : 00 00 00 00 00 00 00 00 00 00 00 00 55 55 55 55

Console(traffic)# do show interfaces mac counters ethernet 0/50,51

Interface      UC Received          MC Received          BC Received       Octets Received
---------  -------------------- -------------------- -------------------- -------------------
  0/50              0                    0                    0                    0
  0/51              1                    0                    0                   64

Interface        UC Sent              MC Sent             BRDC Sent           Octets Sent
---------  -------------------- -------------------- -------------------- --------------------
  0/50              1                    0                    0                   64
  0/51              0                    0                    0                    0
```

MACsec Egress Configuration: Port 0/50

Step-1: Enable MACsec on the device

Initialize MACsec on the device

```
Console# cpss-api call cpssDxChMacSecInit devNum 0 unitBmp 0
result=GT_OK = Operation succeeded
values={ }
```

Step-2: Enable MACsec on Port 0/50 egress side

MACsec Classifier configuration settings per port

```
Console# cpss-api call cpssDxChMacSecPortClassifyConfigSet devNum 0 portNum 50 direction
CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E
portCfg.bypassMacsecDevice(false)>                              <<< Enable MACsec
portCfg.exceptionCfgEnable(false)>
portCfg.forceDrop(false)>
portCfg.defaultVPortValid(false)>
portCfg.defaultVPort(0)>
portCfg.policyMode(CPSS_DXCH_MACSEC_POLICY_MODE_MUST_SECURE_E)> <<< Non MACsec frames not allowed
result=GT_OK = Operation succeeded
values={ }
```

MACsec Transformer configuration settings per port

```
Console# cpss-api call cpssDxChMacSecPortSecyConfigSet devNum 0 portNum 50 direction
CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E
portCfg.bypassMacsecDevice(false)>                              <<< Enable MACsec
portCfg.policyMode(CPSS_DXCH_MACSEC_POLICY_MODE_MUST_SECURE_E)> <<< Non MACsec frames not allowed
portCfg.statCtrl.seqNrThreshold(0)>                             <<< Non-zero value for re-keying proc
portCfg.statCtrl.seqNrThreshold64.l[0](0)>
portCfg.statCtrl.seqNrThreshold64.l[1](0)>
portCfg.pktNumThrStrictCompareModeEnable(false)>
portCfg.ruleSecTag.compEtype(false)>                            <<< MACsec type check
portCfg.ruleSecTag.checkV(false)>                               <<< Version check
portCfg.ruleSecTag.checkKay(false)>                             <<< (C,E bits) check
portCfg.ruleSecTag.checkCe(false)>                              <<< (C=1,E=0) check
portCfg.ruleSecTag.checkSc(false)>                              <<< SC,ES,SCB check
portCfg.ruleSecTag.checkSl(false)>                              <<< SL check
portCfg.ruleSecTag.checkPn(false)>                              <<< PN check
result=GT_OK = Operation succeeded
values={ }
```

Step-3: Install vPort in Classifier Unit

Add a new vPort set for one MACsec classifier

```
Console# cpss-api call cpssDxChMacSecClassifyVportAdd devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E
vPortParams.secTagOffset(0)>12                                  <<< SecTag location from start of frame
vPortParams.pktExpansion(CPSS_DXCH_MACSEC_CLASSIFY_PKT_EXPAND_TYPE_NO_EXPANSION_E)>CPSS_DXCH_MACSEC_CLA
SSIFY_PKT_EXPAND_TYPE_32B_E                                     <<< Expand pkt by 32B
result=GT_OK = Operation succeeded
values={
  vPortHandle= 0xffff618bcdf8
}
```

Obtain vPort index from vPort handler.
This index is used for SA installation.

```
Console# cpss-api call cpssDxChMacSecClassifyVportIndexGet devNum 0 vPortHandle 0xffff618bcdf8
result=GT_OK = Operation succeeded
values={
  vPortIndex=0
}
```

Step-4: Install an SA record in Transform Unit

Add a new SA set for one MACsec Transformer

```
Console# cpss-api call cpssDxChMacSecSecySaAdd devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E vPortId 0
saParams.params(CPSS_DXCH_MACSEC_SECY_SA_PARAM_UNT)>CPSS_DXCH_MACSEC_SECY_SA_EGR_STC
saParams.params.egress.saInUse(false)>                    << SA not in use. Cannot be transformed
saParams.params.egress.confidentialityOffset(0)>          << Bytes authenticated but not encrypted
saParams.params.egress.protectFrames(false)>true          << Enable frame protection
saParams.params.egress.includeSci(false)>true             << Insert SCI in the packet
saParams.params.egress.useEs(false)>                      << Do not modify ES bit in SECTag
saParams.params.egress.useScb(false)>                     << Do not modify SCB bit in SECTag
saParams.params.egress.confProtect(false)>true            << Enable confidentiality protection
saParams.params.egress.controlledPortEnable(false)>true << Allow non-control packets
saParams.params.egress.preSecTagAuthStart(0)>0  << No of bytes from frame start to bypass encryption
saParams.params.egress.preSecTagAuthLength(0)>12          << no of bytes to be authenticated
saParams.actionType(CPSS_DXCH_MACSEC_SECY_SA_ACTION_BYPASS_E)>CPSS_DXCH_MACSEC_SECY_SA_ACTION_EGRESS_E
saParams.destPort(CPSS_DXCH_MACSEC_SECY_PORT_COMMON_E)>
trRecParams.an(0)>2                                       << Association number
trRecParams.keyArr[0](0)>0xad                             << MACsec key
trRecParams.keyArr[1](0)>0x7a
trRecParams.keyArr[2](0)>0x2b
trRecParams.keyArr[3](0)>0xd0
trRecParams.keyArr[4](0)>0x3e
trRecParams.keyArr[5](0)>0xac
trRecParams.keyArr[6](0)>0x83
trRecParams.keyArr[7](0)>0x5a
trRecParams.keyArr[8](0)>0x6f
trRecParams.keyArr[9](0)>0x62
trRecParams.keyArr[10](0)>0x0f
trRecParams.keyArr[11](0)>0xdc
trRecParams.keyArr[12](0)>0xb5
trRecParams.keyArr[13](0)>0x06
trRecParams.keyArr[14](0)>0xb3
trRecParams.keyArr[15](0)>0x45
trRecParams.keyArr[16](0)>*
trRecParams.keyByteCount(0)>16                            << MACsec key-size
trRecParams.sciArr[0](0)>0x12                             << SCI
trRecParams.sciArr[1](0)>0x15
trRecParams.sciArr[2](0)>0x35
trRecParams.sciArr[3](0)>0x24
trRecParams.sciArr[4](0)>0xc0
trRecParams.sciArr[5](0)>0x89
trRecParams.sciArr[6](0)>0x5e
trRecParams.sciArr[7](0)>0x81
trRecParams.seqTypeExtended(false)>                       << Extended packet numbering- 32bit or 64bit
trRecParams.seqNumLo(0)>                                  << Sequence number low/high
trRecParams.seqNumHi(0)>
trRecParams.ssciArr[0](0)>*                               << Valid if seqTypeExtended is True
trRecParams.saltArr[0](0)>*
trRecParams.seqMask(0)>                 << Replay window size - 0 is enforced for strict ordering
trRecParams.customHkeyEnable(false)>    << Custom Hash Key enable flag. False is automatic generation
trRecParams.customHkeyArr[0](0)>*
result=GT_OK = Operation succeeded
values={
  saHandle= 0xffff618d8260
}
```

Step-5: Install Rules in Classifier Unit

Add a new rule for matching a packet to a vPort in MACsec classifier

```
Console# cpss-api call cpssDxChMacSecClassifyRuleAdd devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E vPortHandle 0xffff618bcdf8
ruleParams.key.packetType(0)>                   << 0 - untagged VLAN; 3 - MACsec (ingress only)
ruleParams.key.numTags(0)>1                     << Number of VLAN tags
ruleParams.key.portNum(0)>50
ruleParams.mask.packetType(0)>0
ruleParams.mask.numTags(0)>0x7f
ruleParams.mask.portNum(0)>0x3f
ruleParams.data[0](0)>0x00112233                << MAC DA
ruleParams.data[1](0)>0x44550000                << MAC DA
ruleParams.data[2](0)>0                         << EtherType, VLAN tag
ruleParams.data[3](0)>0
ruleParams.dataMask[0](0)>0xffffffff
ruleParams.dataMask[1](0)>0xffff0000
ruleParams.dataMask[2](0)>0
ruleParams.dataMask[3](0)>0
```

```
ruleParams.policy.rulePriority(0)>              << Priority value used to resolve multi rule matches
ruleParams.policy.drop(false)>                  << do not drop packet
ruleParams.policy.controlPacket(false)>         << do not mark packets as control packets
ruleParams.preemptionClass(CPSS_PORT_PREEMPTION_CLASS_EXPRESS_E)>
result=GT_OK = Operation succeeded
values={
  ruleHandle= 0xffff618d58f8
}
```

Step-6: Enable rules on ingress/egress rule handlers

Enable a configured Classifier rule

```
Console# cpss-api call cpssDxChMacSecClassifyRuleEnable devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E ruleHandle 0xffff618d58f8 enable true
result=GT_OK = Operation succeeded
values={ }
```

Step 7: Send plain-text packet from CPU to Port 0/50.

Capture Port 0/50 egress packet and verify if the packets are encrypted.

Note: DSA header is included in the CPU mirror packet context

```
Console# traffic
Console(traffic)# cpu rx dump
Console(traffic)# send port 0/50 data 0x001122334455001122334466
dxChNetIfRxPacketGet: device[0] queue[0] [112] bytes (include DSA bytes !)
0x0000 : 00 11 22 33 44 55 00 11 22 33 44 66 10 9f 30 02
0x0010 : 80 05 c4 c0 80 00 01 98 00 00 00 00 88 e5 2e 00
0x0020 : 00 00 00 01 12 15 35 24 c0 89 5e 81 f9 b7 0b 99
0x0030 : fc ec f3 73 9e e2 0a 29 fa 64 90 bf 68 a2 8b 4a
0x0040 : 54 5e 48 87 12 27 b3 d5 48 98 be ce 57 97 e7 0e
0x0050 : 82 11 25 41 e2 fb 54 88 2e 08 5c b7 3f 24 c8 09
0x0060 : 1e 5a 68 5a 0b 63 45 d0 e9 db f3 84 55 55 55 55

Console(traffic)# dbg dsa-tag decode 0x109f30028005c4c08000019800000000
input DSA string :      "109f30028005c4c08000019800000000"
DSA structure :         {
  dsaInfo={
    toCpu={
      cpuCode="CPSS_NET_FIRST_USER_DEFINED_E",
      tag0TpidIndex=0,
      originByteCount=92,
      wasTruncated=false,
      isTagged=false,
      srcIsTrunk=false,
      isEgressPipe=false,
      hwDevNum=16,
      flowIdTtOffset={
        flowId=0
      },
      timestamp=184,
      packetIsTT=false,
      interface={
        ePort=51,
        srcTrunkId=0,
        portNum=51
      }
    }
  },
  commonParams={
    dsaTagType="CPSS_DXCH_NET_DSA_4_WORD_TYPE_ENT",
    dropOnSource=false,
    packetIsLooped=false,
    cfiBit=0,
    vpt=1,
    vid=2
  },
  dsaType="CPSS_DXCH_NET_DSA_CMD_TO_CPU_E"
}
Successful DSA parsing
```

Step 8: Verify statistics

```
Console(traffic)# do cpss-api call cpssDxChMacSecClassifyRuleIndexGet devNum 0 ruleHandle
0xffff618d58f8
result=GT_OK = Operation succeeded
values={
  ruleIndex=0
}

Console(traffic)# do cpss-api call cpssDxChMacSecClassifyStatisticsTcamHitsGet devNum 0 unitBmp 0
direction CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E ruleId 0 syncEnable true
result=GT_OK = Operation succeeded
values={
  statTcamHitsCounter=0x1
}

Console(traffic)# do cpss-api call cpssDxChMacSecPortClassifyStatisticsGet devNum 0 portNum 50
direction CPSS_DXCH_MACSEC_DIRECTION_EGRESS_E syncEnable true preemptionClass
CPSS_PORT_PREEMPTION_CLASS_EXPRESS_E
result=GT_OK = Operation succeeded
values={
  portStat={
    pktsErrIn=0x0,
    tcamHitMultiple=0x0,
    headerParserDroppedPkts=0x0,
    pktsDropped=0x0,
    tcamMiss=0x0,
    pktsCtrl=0x0,
    pktsData=0x1
  }
}

Console(traffic)# do show interfaces mac counters ethernet 0/50,51
Interface      UC Received          MC Received          BC Received       Octets Received
--------- -------------------- -------------------- -------------------- -------------------
  0/50             0                    0                    0                    0
  0/51             1                    0                    0                   96

Interface        UC Sent              MC Sent             BRDC Sent           Octets Sent
--------- -------------------- -------------------- -------------------- -------------------
  0/50             1                    0                    0                   96
  0/51             0                    0                    0                    0

Console(traffic)#
```

MACsec Egress Configuration: Port 0/51

Step-1: Initialize MACsec on device (Already done while testing MACsec ingress)

```
Console# cpss-api call cpssDxChMacSecInit devNum 0 unitBmp 0
result=GT_OK = Operation succeeded
values={ }
```

Step-2: Enable MACsec on Port 0/51 ingress side

```
Console# cpss-api call cpssDxChMacSecPortClassifyConfigSet devNum 0 portNum 51 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E
portCfg.bypassMacsecDevice(false)>
portCfg.exceptionCfgEnable(false)>
portCfg.forceDrop(false)>
portCfg.defaultVPortValid(false)>
portCfg.defaultVPort(0)>
portCfg.policyMode(CPSS_DXCH_MACSEC_POLICY_MODE_MUST_SECURE_E)>
result=GT_OK = Operation succeeded
values={ }

Console# cpss-api call cpssDxChMacSecPortSecyConfigGet devNum 0 portNum 52 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E
result=GT_OK = Operation succeeded
values={
  portCfg={
    statCtrl={
      seqNrThreshold=0,
      seqNrThreshold64=0x0
    },
    bypassMacsecDevice=true,
    pktNumThrStrictCompareModeEnable=false,
    policyMode="CPSS_DXCH_MACSEC_POLICY_MODE_MUST_SECURE_E",
    ruleSecTag={
      checkCe=true,
      checkSl=true,
      checkPn=true,
      checkKay=true,
      checkSc=true,
      checkV=true,
      compEtype=true
    }
  }
}

Console# cpss-api call cpssDxChMacSecPortSecyConfigSet devNum 0 portNum 51 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E
portCfg.bypassMacsecDevice(false)>
portCfg.policyMode(CPSS_DXCH_MACSEC_POLICY_MODE_MUST_SECURE_E)>
portCfg.statCtrl.seqNrThreshold(0)>
portCfg.statCtrl.seqNrThreshold64.l[0](0)>
portCfg.statCtrl.seqNrThreshold64.l[1](0)>
portCfg.pktNumThrStrictCompareModeEnable(false)>true
portCfg.ruleSecTag.compEtype(false)>true
portCfg.ruleSecTag.checkV(false)>true
portCfg.ruleSecTag.checkKay(false)>true
portCfg.ruleSecTag.checkCe(false)>true
portCfg.ruleSecTag.checkSc(false)>true
portCfg.ruleSecTag.checkSl(false)>true
portCfg.ruleSecTag.checkPn(false)>true
result=GT_OK = Operation succeeded
values={ }
```

Step-3: Install vPort in Classifier Unit

```
Console# cpss-api call cpssDxChMacSecClassifyVportAdd devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E
vPortParams.secTagOffset(0)>
vPortParams.pktExpansion(CPSS_DXCH_MACSEC_CLASSIFY_PKT_EXPAND_TYPE_NO_EXPANSION_E)>
result=GT_OK = Operation succeeded
values={
  vPortHandle=0xffff618d6990
}
```

```
Console# cpss-api call cpssDxChMacSecClassifyVportIndexGet devNum 0 vPortHandle 0xffff618d6990
result=GT_OK = Operation succeeded
values={
  vPortIndex=0
}
```

Step-4: Install an SA record in Transform Unit

```
Console# cpss-api call cpssDxChMacSecSecySaAdd devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E vPortId 0
saParams.params(CPSS_DXCH_MACSEC_SECY_SA_PARAM_UNT)>CPSS_DXCH_MACSEC_SECY_SA_ING_STC
saParams.params.ingress.saInUse(false)>false
saParams.params.ingress.confidentialityOffset(0)>
saParams.params.ingress.replayProtect(false)>true
saParams.params.ingress.validateFramesTagged(CPSS_DXCH_MACSEC_SECY_FRAME_VALIDATE_DISABLE_E)>CPSS_DXCH_
MACSEC_SECY_FRAME_VALIDATE_STRICT_E
saParams.params.ingress.sciArr[0](0)>0x12
saParams.params.ingress.sciArr[1](0)>0x15
saParams.params.ingress.sciArr[2](0)>0x35
saParams.params.ingress.sciArr[3](0)>0x24
saParams.params.ingress.sciArr[4](0)>0xc0
saParams.params.ingress.sciArr[5](0)>0x89
saParams.params.ingress.sciArr[6](0)>0x5e
saParams.params.ingress.sciArr[7](0)>0x81
saParams.params.ingress.an(0)>2
saParams.params.ingress.allowTagged(false)>true
saParams.params.ingress.allowUntagged(false)>false
saParams.params.ingress.validateUntagged(false)>false
saParams.params.ingress.preSecTagAuthStart(0)>
saParams.params.ingress.preSecTagAuthLength(0)>12
saParams.params.ingress.retainSecTag(false)>
saParams.params.ingress.retainIcv(false)>
saParams.actionType(CPSS_DXCH_MACSEC_SECY_SA_ACTION_BYPASS_E)>CPSS_DXCH_MACSEC_SECY_SA_ACTION_INGRESS_E
saParams.destPort(CPSS_DXCH_MACSEC_SECY_PORT_COMMON_E)>CPSS_DXCH_MACSEC_SECY_PORT_CONTROLLED_E
trRecParams.an(0)>2
trRecParams.keyArr[0](0)>0xad
trRecParams.keyArr[1](0)>0x7a
trRecParams.keyArr[2](0)>0x2b
trRecParams.keyArr[3](0)>0xd0
trRecParams.keyArr[4](0)>0x3e
trRecParams.keyArr[5](0)>0xac
trRecParams.keyArr[6](0)>0x83
trRecParams.keyArr[7](0)>0x5a
trRecParams.keyArr[8](0)>0x6f
trRecParams.keyArr[9](0)>0x62
trRecParams.keyArr[10](0)>0x0f
trRecParams.keyArr[11](0)>0xdc
trRecParams.keyArr[12](0)>0xb5
trRecParams.keyArr[13](0)>0x06
trRecParams.keyArr[14](0)>0xb3
trRecParams.keyArr[15](0)>0x45
trRecParams.keyArr[16](0)>*
trRecParams.keyByteCount(0)>16
trRecParams.sciArr[0](0)>0x12
trRecParams.sciArr[1](0)>0x15
trRecParams.sciArr[2](0)>0x35
trRecParams.sciArr[3](0)>0x24
trRecParams.sciArr[4](0)>0xc0
trRecParams.sciArr[5](0)>0x89
trRecParams.sciArr[6](0)>0x5e
trRecParams.sciArr[7](0)>0x81
trRecParams.seqTypeExtended(false)>
trRecParams.seqNumLo(0)>*
result=GT_OK = Operation succeeded
values={
  saHandle=0xffff618d9f38
}
```

Step-5: Install Rules in Classifier Unit

```
Console# cpss-api call cpssDxChMacSecClassifyRuleAdd devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E vPortHandle 0xffff618d6990
ruleParams.key.packetType(0)>
ruleParams.key.numTags(0)>1
```

```
ruleParams.key.portNum(0)>51
ruleParams.mask.packetType(0)>
ruleParams.mask.numTags(0)>0x7f
ruleParams.mask.portNum(0)>0x3f
ruleParams.data[0](0)>0x00112233
ruleParams.data[1](0)>0x44550000
ruleParams.data[2](0)>0
ruleParams.data[3](0)>0
ruleParams.dataMask[0](0)>0xffffffff
ruleParams.dataMask[1](0)>0xffff0000
ruleParams.dataMask[2](0)>0
ruleParams.dataMask[3](0)>0
ruleParams.policy.rulePriority(0)>
ruleParams.policy.drop(false)>
ruleParams.policy.controlPacket(false)>
ruleParams.preemptionClass(CPSS_PORT_PREEMPTION_CLASS_EXPRESS_E)>
result=GT_OK = Operation succeeded
values={
  ruleHandle=0xffff618d71b8
}
```

Step-6: Enable rules on ingress rule handlers

```
Console# cpss-api call cpssDxChMacSecClassifyRuleEnable devNum 0 unitBmp 0 direction
CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E ruleHandle 0xffff618d71b8 enable true
result=GT_OK = Operation succeeded
values={ }
Console#
```

Step 7: Send plain-text packet from CPU to Port 0/50.

Capture Port 0/50 egress packet and verify if the packets are encrypted.

Note: DSA header is included in the CPU mirror packet context

```
Console# traffic
Console(traffic)# cpu rx dump
Console(traffic)# send port 0/50 data 0x001122334455001122334466
dxChNetIfRxPacketGet: device[0] queue[0] [80] bytes (include DSA bytes !)
0x0000 : 00 11 22 33 44 55 00 11 22 33 44 66 10 9f 30 02
0x0010 : 80 03 c4 c0 80 00 01 98 00 00 00 00 00 00 00 00
0x0020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0040 : 00 00 00 00 00 00 00 00 00 00 00 00 55 55 55 55

Console(traffic)# dbg dsa-tag decode 0x109f30028003c4c08000019800000000
input DSA string :      "109f30028003c4c08000019800000000"
DSA structure :         {
  dsaInfo={
    toCpu={
      cpuCode="CPSS_NET_FIRST_USER_DEFINED_E",
      tag0TpidIndex=0,
      originByteCount=60,
      wasTruncated=false,
      isTagged=false,
      srcIsTrunk=false,
      isEgressPipe=false,
      hwDevNum=16,
      flowIdTtOffset={
        flowId=0
      },
      timestamp=120,
      packetIsTT=false,
      interface={
        ePort=51,
        srcTrunkId=0,
        portNum=51
      }
    }
  },
  commonParams={
    dsaTagType="CPSS_DXCH_NET_DSA_4_WORD_TYPE_ENT",
    dropOnSource=false,
    packetIsLooped=false,
    cfiBit=0,
```

```
    vpt=1,
    vid=2
  },
  dsaType="CPSS_DXCH_NET_DSA_CMD_TO_CPU_E"
}
Successful DSA parsing
```

Step 8: Verify statistics

```
Console(traffic)# do cpss-api call cpssDxChMacSecClassifyRuleIndexGet devNum 0 ruleHandle ?
Cpss function name and parameters
Console(traffic)# do cpss-api call cpssDxChMacSecClassifyRuleIndexGet devNum 0 ruleHandle
0xffff618d71b8
result=GT_OK = Operation succeeded
values={
  ruleIndex=0
}

Console(traffic)# do cpss-api call cpssDxChMacSecClassifyStatisticsTcamHitsGet devNum 0 unitBmp 0
direction CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E ruleId 0 syncEnable true
result=GT_OK = Operation succeeded
values={
  statTcamHitsCounter=0x1
}

Console(traffic)# do cpss-api call cpssDxChMacSecPortClassifyStatisticsGet devNum 0 portNum 51
direction CPSS_DXCH_MACSEC_DIRECTION_INGRESS_E syncEnable true preemptionClass
CPSS_PORT_PREEMPTION_CLASS_EXPRESS_E
result=GT_OK = Operation succeeded
values={
  portStat={
    pktsErrIn=0x0,
    tcamHitMultiple=0x0,
    headerParserDroppedPkts=0x0,
    pktsDropped=0x0,
    tcamMiss=0x0,
    pktsCtrl=0x0,
    pktsData=0x1
  }
}

Console(traffic)# do show interfaces mac counters ethernet 0/50,51

Interface      UC Received          MC Received          BC Received       Octets Received
--------- -------------------- -------------------- -------------------- -------------------
  0/50             0                    0                    0                    0
  0/51             1                    0                    0                   96

Interface        UC Sent              MC Sent             BRDC Sent           Octets Sent
--------- -------------------- -------------------- -------------------- --------------------
  0/50             1                    0                    0                   96
  0/51             0                    0                    0                    0
```