IPsec Full Offload

Table of contents

Note

This feature is supported on crypto-enabled products of BlueField-2 DPUs, as well as on ConnectX-6 Dx, ConnectX-6 Lx and ConnectX-7 adapter cards. Note that it is not supported on ConnectX-6 cards.

Newer/future crypto-enabled DPU and adapter product generations should also support this feature, unless explicitly stated otherwise in their documentation.

Note

When using NVIDIA® BlueField®-2 DPUs and NVIDIA® ConnectX®-6 Dx adapters only: If your target application utilizes 100Gb/s or a higher bandwidth, where a substantial part of the bandwidth is allocated for IPsec traffic, please refer to the relevant DPU or adapter card Product Release Notes to learn about a potential bandwidth limitation. To access the Release Notes, visit https://docs.nvidia.com/networking/, or contact your NVIDIA sales representative.

Note

ConnectX-6 Dx adapters only support Full Offload: Encrypted Overlay (where a Hypervisor controls IPsec offload - See for example OVS IPsec - https://docs.openvswitch.org/en/latest/tutorials/ipsec/) in a Linux OS with NVIDIA drivers.

> **Note**
>
> This feature requires Linux kernel v6.6, or higher.

This feature is designed to enable IPsec full offload in switchdev mode. The ip-xfrm command is used to configure IPsec states and policies, and it is similar to legacy mode configuration. However, there are several limitations to the use of full offload in this mode:

1. Only IPsec Transport Mode and Tunnel Mode are supported.
2. The first IPsec TX state/policy is not allowed to be offloaded if any offloaded TC rule exists, and the same applies for the first RX state/policy. More specifically, IPsec RX/TX tables must be created before offloading any TC rule. For this reason, it is a common practice to configure IPsec rules before adding any TC rule.

Following is an example for IPsec configuration with a VXLan tunnel:

- Enable switchdev mode:

```
echo 1 > /sys/class/net/$PF0 /device/sriov_numvfs
echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
devlink dev param set pci/0000:08:00.0 name flow_steering_mode value dmfs
cmode runtime
devlink dev eswitch set pci/0000:08:00.0 mode switchdev
echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

- Configure PF/VF/REP netdevices, and place a VF in a namespace:

```
ifconfig $PF $LOCAL_TUN/16 up
ip l set dev $PF mtu 2000
```

```
ifconfig $REP up
ip netns add ns0
ip link set dev $VF netns ns0
ip netns exec ns0 ifconfig $VF $IP/16 up
```

- Configure IPsec states and policies:

```
ip xfrm state add src $LOCAL_TUN/16 dst $REMOTE_IP/16 proto esp spi
0xb29ed314 reqid 0xb29ed314 mode transport aead
'rfc4106(gcm(aes))' 0x20f01f80a26f633d85617465686c32552c92c42f 128 offload packet dev
$PF dir out sel src $LOCAL_TUN/16 dst $REMOTE_IP/16 flag esn replay-window
64
ip xfrm state add src $REMOTE_IP/16 dst $LOCAL_TUN/16 proto esp spi
0xc35aa26e reqid 0xc35aa26e mode transport aead
'rfc4106(gcm(aes))' 0x6cb228189b4c6e82e66e46920a2cde39187de4ba 128 offload packet
dev $PF dir in sel src $REMOTE_IP/16 dst $LOCAL_TUN/16 flag esn replay-
window 64

ip xfrm policy add src $LOCAL_TUN dst $REMOTE_IP offload packet dev $PF dir
out tmpl src $LOCAL_TUN/16 dst $REMOTE_IP/16 proto esp reqid 0xb29ed314
mode transport priority 12
ip xfrm policy add src $REMOTE_IP dst $LOCAL_TUN offload packet dev $PF dir
in tmpl src $REMOTE_IP/16 dst $LOCAL_TUN/16 proto esp reqid 0xc35aa26e
mode transport priority 12
```

- Configure Openvswitch:

```
ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs $REP
ovs-vsctl add-port br-ovs vxlan1 -- set interface vxlan1 type=vxlan
options:local_ip=$LOCAL_TUN options:remote_ip=$REMOTE_IP
options:key=$VXLAN_ID options:dst_port=4789
```

IPsec Full Offload for RDMA Traffic

This IPsec Full Offload for RDMA Traffic option provides a significant performance improvement compared to the software IPsec counterpart, and enables the use of IPsec over RoCE packets, which are outside the network stack and cannot be used without full hardware offload. As a result, users can leverage the benefits of the IPsec protocol with RoCE V2, even when using SR-IOV VFs.

The configuration steps for this feature should be identical to the steps mentioned above, but if this feature is supported, the traffic that will be sent can also be RoCEV2 IPsec traffic.

To configure this feature:

1. Configure an SR-IOV VF normally, and add its OVS/TC rules.

2. Enable IPsec over VF. For more information, please see IPsec Functionality.

3. Configure IPsec policies and states on the relevant VF net device. This should be identical to the software configuration of IPsec rules, which can be done using one of the following implementation options:

Command	Offload Request Parameter
iproute2 ip xfrm	offload packet
libreswan	nic-offload=packet
strongswan	

Note

For this feature to work, switchdev mode and dmfs steering mode must be enabled.

- The following is a full minimalistic configuration example using iproute, whereas PF0 is the netdevice PF, F0_REP is the VF representor, and NIC is the VF netdevice to configure IPsec over:

```
1. echo 1 > /sys/class/net/$PF0 /device/sriov_numvfs
2. echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
3. devlink dev eswitch set pci/0000:08:00.0 mode switchdev
4. devlink dev param set pci/0000:08:00.0 name flow_steering_mode value dmfs
cmode runtime
5. devlink port function set pci/0000:08:00.0/1 ipsec_packet enable
6. echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
7. tc qdisc add dev $PF0 ingress
tc qdisc add dev $VF0_REP ingress
tc filter add dev $PF0 parent ffff: protocol 802.1q chain 0 flower vlan_id 10
vlan_ethtype 802.1q cvlan_id 5 action vlan pop action vlan pop action mirred
egress redirect dev $VF0_REP

tc filter add dev $VF0_REP parent ffff: protocol all chain 0 flower action vlan
push protocol 802.1q id 5 action vlan push protocol 802.1q id 10 action mirred
egress redirect dev $PF0

8. ifconfig $PF0 $PF_IP/24 up
ifconfig $NIC $LOC_IP/$SUB_NET up
ip link set dev $VF_REP up
9. ip xfrm state flush
ip xfrm policy flush
```

- Configure ipsec states and policies:

```
#states
ip -4 xfrm state add src $LOC_IP/$SUB_NET dst $REMOTE_IP/$SUB_NET proto
esp spi 1000 reqid 10000 aead
'rfc4106(gcm(aes))' 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 mode transport
sel src $LOC_IP dst $REMOTE_IP offload packet dev $NIC dir out
ip -4 xfrm state add src $REMOTE_IP/$SUB_NET dst $LOC_IP/$SUB_NET proto
esp spi 1001 reqid 10001 aead
'rfc4106(gcm(aes))' 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 mode transport
sel src $REMOTE_IP dst $LOC_IP offload packet dev $NIC dir in
#policies
```

```
ip -4 xfrm policy add src $LOC_IP dst $REMOTE_IP offload packet dev $NIC dir out tmpl src $LOC_IP/$SUB_NET dst $REMOTE_IP/$SUB_NET proto esp reqid 10000 mode transport
ip -4 xfrm policy add src $REMOTE_IP dst $LOC_IP offload packet dev $NIC dir in tmpl src $REMOTE_IP/$SUB_NET dst $LOC_IP/$SUB_NET proto esp reqid 10001 mode transport
ip -4 xfrm policy add src $REMOTE_IP dst $LOC_IP dir fwd tmpl src $REMOTE_IP/$SUB_NET dst $LOC_IP/$SUB_NET proto esp reqid 10001 mode transport
```

Note that the configuration above is for one side only, yet IPsec must be configured for both sides in order for them to communicate properly. The configuration for the other side should be almost identical, but Step 9 would be configured in an asymmetrical way, meaning the first policy would look the following, and all other states/policies would be adjusted accordingly:

```
ip -4 xfrm state add src $LOC_IP/$SUB_NET dst $REMOTE_IP/$SUB_NET proto esp spi 1001 reqid 10001 aead 'rfc4106(gcm(aes))' 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 mode transport sel src $LOC_IP dst $REMOTE_IP offload packet dev $NIC dir out
```

Once this step is completed, you can send any RoCE traffic of your choice between the two machines with configured IPsec. For example, `ibv_rc_pingpong -g 3 -d VF_device` : on one side, and `ibv_rc_pingpong -g 3 -d VF_device $IP_OF_OTHER_SIDE` : on the other side.

Finally, you can verify that the traffic was encrypted using IPsec by using the ipsec counters:

```
ethtool -S VF_NETDEV | grep ipsec
```