NVIDIA DOCA East-West Overlay Encryption Application
This guide describes IPsec-based strongSwan solution on top of NVIDIA® BlueField® DPU.

**Note**

If your target application utilizes 100Gb/s or higher bandwidth, where a substantial part of the bandwidth is allocated for IPsec traffic, please refer to the *NVIDIA BlueField-2 DPUs Product Release Notes* to learn about a potential bandwidth limitation. To access the relevant product release notes, please contact your NVIDIA sales representative.

**Introduction**

IPsec is used to set up encrypted connections between different devices. It helps keep data sent over public networks secure. IPsec is often used to set up VPNs, and it works by encrypting IP packets as well as authenticating the packets’ originator.

IPsec contains the following main modules:

- **Key exchange** – a key is a string of random bytes that can be used for encryption and decryption of messages. IPsec sets up keys with a key exchange between the connected devices, so that each device can decrypt the other device's messages.

- **Authentication** – IPsec provides authentication for each packet which ensures that they come from a trusted source.

- **Encryption** – IPsec encrypts the payloads within each packet and possibly, based on the transport mode, the packet's IP header.

- **Decryption** – at the other end of the communication, packets are decrypted by the IPsec supported node.

IPsec supports two types of headers:
- Authentication header (AH) – AH protocol ensures that packets are from a trusted source. AH does not provide any encryption.

- Encapsulating security protocol (ESP) – ESP encrypts the payload for each packet as well as the IP header depending on the transport mode. ESP adds its own header and a trailer to each data packet.

IPsec support two types of transport mode:

- IPsec tunnel mode – used between two network nodes, each acting as tunnel initiator/terminator on a public network. In this mode, the original IP header and payload are both encrypted. Since the IP header is encrypted, an IP tunnel is added for network forwarding. At each end of the tunnel, the routers decrypt the IP headers to route the packets to their destinations.

- Transport mode – the payload of each packet is encrypted, but the original IP header is not. Intermediary network nodes are therefore able to view the destination of each packet and route the packet, unless a separate tunneling protocol is used.

strongSwan is an open-source IPsec-based VPN solution. For more information, refer to strongSwan documentation.

**System Design**

IPsec packet offload offloads both IPsec crypto (encrypt/decrypt) and IPsec encapsulation to the hardware.

The deployment model allows the IPsec offload to be transparent to the host with the benefits of securing legacy workloads (no dependency on host SW stack) and to zero CPU utilization on host.

IPsec packet offload configuration works with and is transparent to OVS offload. This means all packets from OVS offload are encrypted by IPsec rules.

The following figure illustrates the interaction between IPsec packet offload and OVS VXLAN offload.
Note

IPsec packet offload is only supported on Ubuntu Bluefield kernel 5.15

Note

OVS offload and IPsec IPv6 do not work together.

Application Architecture

2. Traffic is sent from the host through BlueField.

3. Using OVS, the packets are encapsulated on ingress using tunnel protocols (VXLAN for example) to match IPsec configuration by strongSwan.

4. Set by strongSwan configuration file, traffic will be encrypted using the hardware offload.

5. Egress flow is decryption first, decapsulation of the tunnel header and forward to the relevant physical function.

**DOCA Libraries**

N/A

**Configuration Flow**

The following section provides information on manually configuring IPsec packet offload in general and on using OVS IPsec with strongSwan specifically.

>Note
If you are working directly with the `ip xfrm` tool, use `/opt/mellanox/iproute2/sbin/ip` to benefit from IPsec packet offload support.

There are two parts in the configuration flow

1. Enabling IPsec packet offload mode.
2. Configuring the IPsec OVS bridge using one of three modes of authentication.

**Note**

An alternative for step two is configuring `swanctl.conf` files (configuration files for strongSwan) manually and using strongSwan directly instead of using IPsec OVS (which automatically generates `swanctl.conf` files) as explained in section "Configuring OVS IPsec Using strongSwan Manually".

**Enabling IPsec Packet Offload**

This section explicitly enables IPsec packet offload on the Arm cores before setting up offload-aware IPsec tunnels.

**Note**

If an OVS VXLAN tunnel configuration already exists, stop `openvswitch` service prior to performing the steps below and restart the service afterwards.
Explicitly enable IPsec full offload on the Arm cores.

1. Set `IPSEC_FULL_OFFLOAD="yes"` in `/etc/mellanox/mlnx-bf.conf`.

![Note]

If `IPSEC_FULL_OFFLOAD` does not appear in `/etc/mellanox/mlnx-bf.conf` then you are probably using an old version of the BlueField image. Check the way of enabling IPsec full offload in a previous DOCA versions in the NVIDIA DOCA Documentation Archives.

2. Restart IB driver (rebooting also works). Run:

```
/etc/init.d/openibd restart
```

![Note]

If `mlx-regex` is running:

1. Disable `mlx-regex` prior to running restarting the IB driver:

```
systemctl stop mlx-regex
```

2. Restart IB driver according to the command above.

3. Re-enable `mlx-regex` after the restart has finished:

```
systemctl restart mlx-regex
```
### Configuring OVS IPsec

To revert IPsec full offload mode, redo the procedure from step 1, only difference is to set `IPSEC_FULL_OFFLOAD="no"` in `/etc/mellanox/mlnx-bf.conf`.

#### Note

Before proceeding with this section, make sure to follow the procedure in section "Enabling IPsec Packet Offload" for both DPUs.

This section configures OVS IPsec VXLAN tunnel which automatically generates the `swanctl.conf` files and runs strongSwan (the IPsec daemon). The following figure illustrates an example with two BlueField DPUs, Left and Right, operating with a secured VXLAN channel.
Two BlueField DPs are required to build an OVS IPsec tunnel between the two hosts, Right and Left.

The OVS IPsec tunnel configures an unaware IPsec connection between the two hosts' InfiniBand devices. For the sake of this example, the host's InfiniBand network device is HOST_PF, and the DPU's host representor is PF_REP and the DPU's physical function PF.

This example sets up the following variables on both Arms:

```
# host_ip1=1.1.1.1
# host_ip2=1.1.1.2
# HOST_PF=ens7np0
# ip1=192.168.50.1
# ip2=192.168.50.2
# PF=p0
# PF_REP=pf0hpf
```

**Note**

The name of the HOST_PF could be different in your machine. You may verify this by running:
1. Configure IP addresses for the HOST_PFS of both hosts (x86):

1. On host_1:

   # ifconfig $HOST_PF $host_ip1/24 up

2. On host_2:

   # ifconfig $HOST_PF $host_ip2/24 up

Note

Step 1 is the only command that is performed on the host, the rest of the commands are performed on the Arm (DPU) side.

2. Configure IP addresses for the PFs of both Arms:

1. On Arm_1:

   # ifconfig $PF $ip1/24 up
2. On Arm_2:

```
# ifconfig $PF $ip2/24 up
```

3. Start Open vSwitch. If your operating system is Ubuntu, run the following on both Arm_1 and Arm_2:

```
# service openvswitch-switch start
```

If your operating system is CentOS, run the following on both Arm_1 and Arm_2:

```
# service openvswitch restart
```

4. Start OVS IPsec service. Run on both Arm_1 and Arm_2:

```
# systemctl start openvswitch-ipsec.service
```

4. Start OVS IPsec service. Run on both Arm_1 and Arm_2:

```
# systemctl start openvswitch-ipsec.service
```

5. Set up OVS bridges in both DPUs. Run on both Arm_1 and Arm_2:

```
# ovs-vsctl add-br vxlan-br
# ovs-vsctl add-port ovs-br $PF_REP
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

**Note**

Configuring `other_config:hw-offload=true` sets IPsec Packet offload. Setting it to `false` sets software IPsec.
6. Set up IPsec tunnel on the OVS bridge. Three **authentication methods** are possible, choose your preferred authentication method and follow the steps relevant to it. Note that the last two authentication methods requires you to create certificates (self-signed certificates or certificate authority certificates).

**Note**

The MTU of the MTU of the tunnel interface (PF) should be at least 50 bytes larger than the MTU of the endpoints of the tunnels above (PF_REP) to account for the size of the VXLAN tunnel header. For example, if the MTU of PF_REP is 1500 then the MTU of PF should be at least 1550.

To configure the MTU of the PF:

```bash
# ifconfig $PF mtu $PF_MTU up
```

**Authentication Methods**

The following subsections detail the possible authentication methods for setting up the IPsec tunnel on the OVS bridge.

**Pre-shared Key**

This method configures OVS IPsec using a pre-shared key. You must select a pre-shared key, for example:

```bash
# ifconfig $PF mtu $PF_MTU up
```
1. Set up the VXLAN tunnel:

1. On Arm_1, run:

```
$ psk=swordfish
# ovs-vsctl add-port vxlan-br tun -- \
  set interface tun type=vxlan \
  options:local_ip=$ip1 \
  options:remote_ip=$ip2 \
  options:key=100 \
  options:dst_port=4789 \
  options:psk=$psk
```

2. On Arm_2, run:

```
# ovs-vsctl add-port vxlan-br tun -- \
  set interface tun type=vxlan \
  options:local_ip=$ip2 \
  options:remote_ip=$ip1 \
  options:key=100 \
  options:dst_port=4789 \
  options:psk=$psk
```

### Self-signed Certificate

This method configures OVS IPsec using self-signed certificates. You must generate self-signed certificates and keys. This example demonstrates how to generate self-signed certificates using `ovs-pki` but you may generate them in any other way while skipping step 1.

1. Generate self-signed certificates using `ovs-pki`:

1. On Arm_1, run:
After running this code you should have host_1-cert.pem and host_1-privkey.pem.

2. On Arm_2, run:

```
# ovs-pki req -u host_2
# ovs-pki self-sign host_2
```

After running this code you should have host_2-cert.pem and host_2-privkey.pem.

2. Configure the certificates and private keys:

1. Copy the certificate of Arm_1 to Arm_2, and the certificate of Arm_2 to Arm_1.

2. On each machine, move both host_1-privkey.pem and host_2-cert.pem to /etc/swanctl/x509/ if on Ubuntu, or /etc/strongswan/swanctl/x509/ if on CentOS.

3. On each machine, move the local private key (host_1-privkey.pem on Arm_1 and host_2-privkey.pem on Arm_2) to /etc/swanctl/private if on Ubuntu, or /etc/strongswan/swanctl/private if on CentOS.

3. Set up OVS other_config on both sides.

1. On Arm_1:

```
# ovs-vsctl set Open_vSwitch . other_config:certificate=/etc/swanctl/x509/host_1-cert.pem
  other_config:private_key=/etc/swanctl/private/host_1-privkey.pem
```

2. On Arm_2:

```
# ovs-vsctl set Open_vSwitch . other_config:certificate=/etc/swanctl/x509/host_2-cert.pem
  other_config:private_key=/etc/swanctl/private/host_2-privkey.pem
```
4. Set up the VXLAN tunnel:

1. On Arm_1:

   # ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan
   options:local_ip=$ip1 \ 
   options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \ 
   options:remote_cert=/etc/swanctl/x509/host_2-cert.pem
   # service openvswitch-switch restart

2. On Arm_2:

   # ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan
   options:local_ip=$ip2 \ 
   options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \ 
   options:remote_cert=/etc/swanctl/x509/host_1-cert.pem
   # service openvswitch-switch restart

**Note**

In steps 3 and 4, if you are in CentOS you must change the path of the certificates to /etc/strongswan/swanctl/x509/ and the path of the private keys to /etc/strongswan/swanctl/private.

**CA-signed Certificate**

This method configures OVS IPsec using certificate authority (CA)-signed certificates. You must generate CA-signed certificates and keys. The example demonstrates how to
generate CA-signed certificates using ovs-pki but you may generate them in any other way while skipping step 1.

1. Generate CA-signed certificates using ovs-pki. For this method, all the certificates and the requests must be in the same directory during the certificate generating and signing. This example refers to this directory as certsworkepace.

   1. On Arm_1, run:

   ```
   # ovs-pki init --force
   # cp /var/lib/openvswitch/pki/controllerca/cacert.pem <path_to>/certsworkepace
   # cd <path_to>/certsworkepace
   # ovs-pki req -u host_1
   # ovs-pki sign host1 switch
   ```

   After running this code, you should have host_1-cert.pem, host_1-privkey.pem, and cacert.pem in the certsworkepace folder.

   2. On Arm_2, run:

   ```
   # ovs-pki init --force
   # cp /var/lib/openvswitch/pki/controllerca/cacert.pem <path_to>/certsworkepace
   # cd <path_to>/certsworkepace
   # ovs-pki req -u host_2
   # ovs-pki sign host_2 switch
   ```

   After running this code, you should have host_2-cert.pem, host_2-privkey.pem, and cacert.pem in the certsworkepace folder.

2. Configure the certificates and private keys:

   1. Copy the certificate of Arm_1 to Arm_2 and the certificate of Arm_2 to Arm_1.

   2. On each machine, move both host_1-privkey.pem and host_2-cert.pem to /etc/swanctl/x509/ if on Ubuntu, or /etc/strongswan/swanctl/x509/ if on CentOS.

   3. On each machine, move the local private key (host_1-privkey.pem if on Arm_1 and host_2-privkey.pem if on Arm_2) to /etc/swanctl/private if on Ubuntu, or
4. On each machine, copy cacert.pem to the x509ca directory under 
/etc/swanctl/x509ca/ if on Ubuntu, or /etc/strongswan/swanctl/x509ca/ if on CentOS.

3. Set up OVS other_config on both sides.

1. On Arm_1:

```
# ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/etc/strongswan/swanctl/x509/host_1.pem \
        other_config:private_key=/etc/strongswan/swanctl/private/host_1-privkey.pem \
        other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

2. On Arm_2:

```
# ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/etc/strongswan/swanctl/x509/host_2.pem \
        other_config:private_key=/etc/strongswan/swanctl/private/host_2-privkey.pem \
        other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

4. Set up the tunnel:

1. On Arm_1:

```
# ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan 
        options:local_ip=$ip1 \ 
        options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \ 
        options:remote_name=host_2
# service openvswitch-switch restart
```

2. On Arm_2:

```
# ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan 
        options:local_ip=$ip2 \ 
```
Ensuring IPsec is Configured

Using `/opt/mellanox/iproute2/sbin/ip xfrm state show`, you should be able to see 4 IPsec states for the IPsec connection you configured with the keyword `in mode packet` meaning which means that you are in IPsec packet HW offload mode.

For example, after configuring IPsec using pre-shared key method, you would get something similar to the following on Arm_1:

```
# /opt/mellanox/iproute2/sbin/ip xfrm state show
src 192.168.50.1 dst 192.168.50.2
  proto esp spi 0x88868f8a0ad reqid 1 mode transport
  replay-window 0 flag esn
  aead rfc4106(gcm(aes))
  0x9f45cc45477e70c4e077bccc0c1473a782143e7ad199f58566519639d03b953b8996383f11 128
  anti-replay esn context:
  seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
  replay_window 1, bitmap-length 1
  00000000
  crypto offload parameters: dev p0 dir out mode packet
```
After insuring that the IPsec connection is configured, you can send encrypted traffic between host_1 and host_2 using the HOST_PFS IP addresses.

### Configuring OVS IPsec Using strongSwan Manually

```
set src 192.168.50.1/32 dst 192.168.50.2/32 proto udp sport 4789
src 192.168.50.2 dst 192.168.50.1
  proto esp spi 0xce8bf4b6 reqid 1 mode transport
  replay-window 0 flag esn
  aead rfc4106(gcm(aes))
0xf2d0e335d9a64ef6e385a630a32b0e43bb5f81290cd34bb8f7592d54f11657ed0258e 128
  anti-replay esn context:
    seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
    replay_window 32, bitmap-length 1
    00000000
  crypto offload parameters: dev p0 dir in mode packet
set src 192.168.50.2/32 dst 192.168.50.1/32 proto udp dport 4789
src 192.168.50.1 dst 192.168.50.2
  proto esp spi 0xcb600a84 reqid 2 mode transport
  replay-window 0 flag esn
  aead rfc4106(gcm(aes))
0x7fb26035299bcb973abea5d581acfbcf87c4bd053b745c4d95c62311f934010973f6 128
  anti-replay esn context:
    seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
    replay_window 1, bitmap-length 1
    00000000
  crypto offload parameters: dev p0 dir out mode packet
set src 192.168.50.1/32 dst 192.168.50.2/32 proto udp dport 4789
src 192.168.50.2 dst 192.168.50.1
  proto esp spi 0xc137d5a0 reqid 2 mode transport
  replay-window 0 flag esn
  aead rfc4106(gcm(aes))
0x28e3d12ad4e24aa9d9e9459de8ef8bb4379e8e12faac0054c5b629b6aa50f7ed4574e 128
  anti-replay esn context:
    seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
    replay_window 32, bitmap-length 1
    00000000
  crypto offload parameters: dev p0 dir in mode packet
set src 192.168.50.2/32 dst 192.168.50.1/32 proto udp sport 4789
```
This section configures an OVS VXLAN tunnel which then uses `swanctl.conf` files and runs strongSwan (the IPsec daemon) manually.

⚠️ **Note**

Before proceeding with this section, make sure to follow the procedure in section "Enabling IPsec Packet Offload" for both DPUs.

1. Build a VXLAN tunnel over OVS and connect the PF representor to the same OVS bridge.

   1. On **Arm_1**:

      ```
      # ovs-vsctl add-br vxlan-br
      # ovs-vsctl add-port vxlan-br PF_REP
      # ovs-vsctl add-port vxlan-br vxlan11 -- set interface vxlan11 type=vxlan
          options:local_ip=$ip1 \       
          options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \
      # ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
      ```

   2. On **Arm_2**:

      ```
      # ovs-vsctl add-br vxlan-br
      # ovs-vsctl add-port vxlan-br PF_REP
      # ovs-vsctl add-port vxlan-br vxlan11 -- set interface vxlan11 type=vxlan
          options:local_ip=$ip2 \       
          options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \
      # ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
      ```

2. If your operating system is Ubuntu, run on both **Arm_1** and **Arm_2**:

   ```
   service openvswitch-switch start
   ```
If your operating system is CentOS, run:

```
service openvswitch restart
```

3. Enable TC offloading for the PF. Run on both Arm_1 and Arm_2:

```
# ethtool -K $PF hw-tc-offload on
```

4. Disable host PF as the port owner from Arm. Run on both Arm_1 and Arm_2:

```
# mlxprivhost -d /dev/mst/mt${pciconf} --disable_port_owner r
```

**Note**

To get `${pciconf}`, run the following on the DPU:

```
# ls --color=never /dev/mst/ | grep --color=never '^m.*f0$' | cut -c 3-
```

For example:

```
# mlxprivhost -d /dev/mst/mt41686_pciconf0 --disable_port_owner r
```

5. Configure the `swanctl.conf` files for each machine. See section `swanctl.conf Files`.

**Note**
Each machine should have exactly one .swanctl.conf file in /etc/swanctl/conf.d/.

6. Load the swanctl.conf files and initialize strongSwan. Run:

1. On the Arm_2, run:

   systemctl restart strongswan.service
   swanctl --load-all

2. On the Arm_1, run:

   systemctl restart strongswan.service
   swanctl --load-all
   swanctl -i --child bf

Now the IPsec connection should be established.

**swanctl.conf Files**

strongSwan configures IPSec packet HW offload using a new value added to its configuration file swanctl.conf. The file should be placed under sysconfdir which by default can be found at /etc/swanctl/swanctl.conf.

The terms Left (BFL) and Right (BFR), in reference to the illustration under "Application Architecture", are used to identify the two nodes (or machines) that communicate.

ℹ️ **Note**

Either side (BFL or BFR) can fulfill either role (initiator or receiver).
In this example, 192.168.50.1 is used for the left PF uplink and 192.168.50.2 for the right PF uplink.

```plaintext
collections {
  BFL-BFR {
    local_addrs = 192.168.50.1
    remote_addrs = 192.168.50.2

    local {
      auth = psk
      id = host1
    }
    remote {
      auth = psk
      id = host2
    }
    children {
      bf-out {
        local_ts = 192.168.50.1/24 [udp]
        remote_ts = 192.168.50.2/24 [udp/4789]
        esp_proposals = aes128gcm128-x25519-esn
        mode = transport
        policies_fwd_out = yes
        hw_offload = packet
      }
      bf-in {
        local_ts = 192.168.50.1/24 [udp/4789]
        remote_ts = 192.168.50.2/24 [udp]
        esp_proposals = aes128gcm128-x25519-esn
        mode = transport
        policies_fwd_out = yes
        hw_offload = packet
      }
    }
    version = 2
    mobike = no
    reauth_time = 0
    proposals = aes128-sha256-x25519
  }
}
}

secrets {
  ike-BF {
```
The BFB installation will place two example swanctl.conf files for BFL and BFR (BFL.swanctl.conf and BFR.swanctl.conf respectively) in the strongSwan conf.d directory. Each node should have only one swanctl.conf file in its strongSwan conf.d directory.

Note that:

- "hw_offload = packet" is responsible for configuring IPsec packet offload
- Packet offload support has been added to the existing hw_offload field and preserves backward compatibility.

For your reference:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Do not configure HW offload.</td>
</tr>
<tr>
<td>crypto</td>
<td>Configure crypto HW offload if supported by the kernel and hardware, fail if not supported.</td>
</tr>
<tr>
<td>yes</td>
<td>Same as crypto (considered legacy).</td>
</tr>
<tr>
<td>packet</td>
<td>Configure packet HW offload if supported by the kernel and hardware, fail if not supported.</td>
</tr>
<tr>
<td>auto</td>
<td>Configure packet HW offload if supported by the kernel and hardware, do not fail (perform fallback to crypto or no as necessary).</td>
</tr>
</tbody>
</table>

- Whenever the value of hw_offload is changed, strongSwan configuration must be reloaded.
- Switching to crypto HW offload requires setting up devlink/ipsec_mode to none beforehand.
- Switching to packet HW offload requires setting up
- [udp/4789] is crucial for instructing strongSwan to IPSec only VXLAN communication.
- Packet HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>reauth_time</td>
<td>Ignored if set</td>
</tr>
<tr>
<td>rekey_time</td>
<td>Do not use. Ignored if set.</td>
</tr>
<tr>
<td>rekey_bytes</td>
<td>Do not use. Not supported and will fail if it is set.</td>
</tr>
<tr>
<td>rekey_packets</td>
<td>Use for rekeying</td>
</tr>
</tbody>
</table>

**Running the Application**

**Installation**

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

**Application Execution**

Notes:

- IPsec daemons are started by `systemd strongswan.service`
- Use `systemctl [start | stop | restart]` to control IPsec daemons through `strongswan.service`. For example, to restart, run:

```
systemctl restart strongswan.service
```

This command effectively does the same thing as `ipsec restart`. 
This subsection explains how to configure and set an IPsec connection using the script. To configure the IPsec connection, you need two DPUs, referred to as the initiator and receiver machines. There are no differences between the two machines except that the initiator is the one that initiates the connection between the two (and should run the script after the receiver).

The script is located under
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh.

### Script Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Valid Values</th>
<th>Use when</th>
<th>Notes</th>
</tr>
</thead>
</table>
| side      | The side of the connection (receiver or initiator). | r | receiver  
  i | initiator | Always | This parameter must be always passed on the command line and cannot be passed in the JSON parameter file. |
| json      | The JSON parameters file full path. | JSON file path, written according to the template demonstrated in the following file:  
  /opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption_params.json. | To pass the parameters as a JSON file | When running the script with JSON file, you cannot pass on the command line other parameters than the side and the JSON file. |

**Note**

Do not use the `ipsec` script (located under `/usr/sbin/ipsec`) to restart/stop/start the IPsec connection.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Valid Values</th>
<th>Use when</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiator_ip_addr</td>
<td>The IP address of the initiator machine's port interface for the IPsec connection.</td>
<td>A valid IP address, ranging from 1.1.1.1 to 255.255.255.255.</td>
<td>Always</td>
<td>In the JSON file, it is set by default to 192.168.50.1.</td>
</tr>
<tr>
<td>receiver_ip_addr</td>
<td>The IP address of the receiver machine's port interface for the IPsec connection.</td>
<td>A valid IP address, ranging from 1.1.1.1 to 255.255.255.255.</td>
<td>Always</td>
<td>In the JSON file, it is set by default to 192.168.50.2.</td>
</tr>
<tr>
<td>port_num</td>
<td>The number of the port interface (p0/p1) for the IPsec connection.</td>
<td>0 or 1.</td>
<td>Always</td>
<td>In the JSON file, it is set by default to 0.</td>
</tr>
<tr>
<td>auth_method</td>
<td>the authentication method of IPsec. can be psk (pre-shared key), ssc (self-signed certificate) or ca (CA-signed certificate). Set by default to psk.</td>
<td>Can be psk (pre-shared key), ssc (self-signed certificate) or ca (CA-signed certificate).</td>
<td>Always</td>
<td>In the JSON file, it is set by default to psk.</td>
</tr>
<tr>
<td>preshared_key</td>
<td>The pre-shared key.</td>
<td>A sequence of characters (string).</td>
<td>The auth_method parameter is psk</td>
<td>In the JSON file it is set by default to swordfish. Both the initiator and receiver must configure the same preshared_key.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Valid Values</th>
<th>Use when</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiator_cert_path</td>
<td>The initiator's certificate.</td>
<td>Any valid self-signed or CA signed certificate. Must provide full path of</td>
<td>The auth_method parameter is ssc or ca</td>
<td>Both the initiator and receiver must configure the same initiator_cert_path.</td>
</tr>
<tr>
<td>receiver_cert_path</td>
<td>The receiver's certificate.</td>
<td>Any valid self-signed or CA signed certificate. Must provide full path of</td>
<td>The auth_method parameter is ssc or ca</td>
<td>Both the initiator and receiver must configure the same receiver_cert_path.</td>
</tr>
<tr>
<td>initiator_key_path</td>
<td>the initiator's private-key.</td>
<td>Any valid private key that is generated with the certificate. Must provide</td>
<td>The side parameter is set to initiator and the auth_method parameter is</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full path of private key.</td>
<td>set to ssc or ca</td>
<td></td>
</tr>
<tr>
<td>receiver_key_path</td>
<td>the receiver's private-key.</td>
<td>Any valid private key that is generated with the certificate. Must provide</td>
<td>The side parameter is set to receiver and the auth_method parameter is</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full path of private key.</td>
<td>set to ssc or ca</td>
<td></td>
</tr>
<tr>
<td>initiator_cacert_path</td>
<td>The initiator's CA certificate.</td>
<td>Any valid CA certificate. Must provide full path of certificate.</td>
<td>The side and auth_method parameters are set to initiator and ca</td>
<td>N/A</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Valid Values</td>
<td>Use when</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>receiver_cert_path</td>
<td>The receiver's CA certificate.</td>
<td>Any valid CA certificate. Must provide full path of certificate.</td>
<td>receiver and ca respectively</td>
<td>N/A</td>
</tr>
<tr>
<td>initiator_cn</td>
<td>The common name (CN) of the initiator's certificate.</td>
<td>Must be the same as the CN described in the initiator's certificate.</td>
<td>receiver and ca respectively</td>
<td>N/A</td>
</tr>
<tr>
<td>receiver_cn</td>
<td>The CN of the receiver's certificate.</td>
<td>Must be the same as the CN described in the receiver's certificate.</td>
<td>receiver and ca respectively</td>
<td>N/A</td>
</tr>
</tbody>
</table>

There are two ways of passing the parameters, either using the JSON parameters file or by passing the parameters on the command line.

**Using JSON Parameters File**
In this method, you must configure the parameters file and then run the script:

1. Configure the JSON parameters file located under
   `/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption_params.js`
   or create a JSON file according to the template of
   `east_west_overlay_encryption_params.json` for the script according to the explanation under
   section "Script Parameters".

2. Run the script on the receiver's DPU with the JSON file:

   ```bash
   /opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
   --side=r --
   json=/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption
   ```

3. Run the script on the initiator's DPU:

   ```bash
   /opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
   --side=i --
   json=/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption
   ```

   You may now send encrypted data over the PF interface (192.168.50.[1|2])
   configured for VXLAN.

**Passing Parameters on Command Line**

In this method, you do not need to configure the parameters file and can run the script
with the appropriate parameters.

**Passing Parameters for Pre-shared Key Authentication Method**

1. Run the script on the receiver's DPU:

   ```bash
   /opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
   --side=r --initiator_ip_addr=INITIATOR_IP_ADDRESS --receiver_ip_addr=RECEIVER_IP_ADDRESS --port_num=PORT_NUM \
   ```
2. Run the script on the initiator's DPU:

```
--auth_method=psk --preshared_key=PRESHARED_KEY
```

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
--side=i --initiator_ip_addr=INITIATOR_IP_ADDRESS --receiver_ip_addr=RECEIVER_IP_ADDRESS --
port_num=PORT_NUM
--auth_method=psk --preshared_key=PRESHARED_KEY
```

Passing Parameters for Self-signed Certificates Authentication Method

1. Run the script on the receiver's DPU:

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
--side=r --initiator_ip_addr=INITIATOR_IP_ADDRESS --receiver_ip_addr=RECEIVER_IP_ADDRESS --
port_num=PORT_NUM
--auth_method=ssc --initiator_cert_path=INITIATOR_CERT_PATH --
receiver_cert_path=RECEIVER_CERT_PATH --receiver_key_path=RECEIVER_KEY_PATH
```

2. Run the script on the initiator's DPU:

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
--side=i --initiator_ip_addr=INITIATOR_IP_ADDRESS --receiver_ip_addr=RECEIVER_IP_ADDRESS --
port_num=PORT_NUM
--auth_method=ssc --initiator_cert_path=INITIATOR_CERT_PATH --
receiver_cert_path=RECEIVER_CERT_PATH --initiator_key_path=INITIATOR_KEY_PATH
```

Passing Parameters for CA Certificates Authentication Method

1. Run the script on the receiver's DPU:
2. Run the script on the initiator's DPU:

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
--side=r --initiator_ip_addr=INITIATOR_IP_ADDRESS --receiver_ip_addr=RECEIVER_IP_ADDRESS --
port_num=PORT_NUM \
--auth_method=ca --initiator_cert_path=INITIATOR_CERT_PATH --
receiver_cert_path=RECEIVER_CERT_PATH --receiver_key_path=RECEIVER_KEY_PATH --
receiver_cacert_path=RECEIVER_CACERT_PATH --initiator_cn=INITIATOR_CN
```

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh
--side=i --initiator_ip_addr=INITIATOR_IP_ADDRESS --receiver_ip_addr=RECEIVER_IP_ADDRESS --
port_num=PORT_NUM \
--auth_method=ssc --initiator_cert_path=INITIATOR_CERT_PATH --
receiver_cert_path=RECEIVER_CERT_PATH --initiator_key_path=INITIATOR_KEY_PATH --
initiator_cacert_path=INITIATOR_CACERT_PATH --receiver_cn=RECEIVER_CN
```

For help and usage, run the script with --help/-h flag:

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh -h
```

**Troubleshooting**

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue you may encounter with the installation or execution of the DOCA applications.

**Building strongSwan**

**Note**
1. Install dependencies mentioned here. `libgmp-dev` is missing from that list, so make sure to install that as well.

2. Git clone https://github.com/Mellanox/strongswan.git.

3. Git checkout BF-5.9.10. This branch is based on the official strongSwan 5.9.10 branch (https://github.com/strongswan/strongswan/tree/5.9.10) with added packaging and support for DOCA IPsec plugin (check NVIDIA DOCA IPsec Security Gateway Application Guide for more information regarding strongSwan DOCA plugin).

4. Run autogen.sh within the strongSwan repo.

5. Run the following:

   ```
   configure --enable-openssl --disable-random --prefix=/usr/local --sysconfdir=/etc --enable-systemd
   make
   make install
   ```

Notes:

- `--enable-systemd` enables the systemd service for strongSwan present inside the GitHub repo (see step 3) at `init/systemd-starter/strongswan.service.in`.

- When building strongSwan on your own, the `openssl.cnf.mlnx` file, required for PK and RNG HW offload via OpenSSL plugin, is not installed. It must be copied over manually from GitHub repo inside the `openssl-conf` directory. See section "Running Strongswan Example" for important notes.

- The `openssl.cnf.mlnx` file references PKA engine shared objects. `libpka` (version 1.3 or later) and `openssl` (version 1.1.1) must be installed for this to work.
Reverting IPsec Configuration

To destroy IPsec configuration, run the following on both machines:

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh -d
```

**Note**

Make sure to run this command only after at least two minutes have passed from running the application on either machines. Otherwise, this may lead to errors.

**Note**

If you run this command without initializing the connection first (in Running strongSwan Example), you may receive errors. These errors have no functional impact and may be safely ignored.

References

- `/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh`
- `/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption_path`