NVIDIA DOCA IPsec Security Gateway Application Guide
# Table of contents

**Introduction**

**System Design**

**Application Architecture**

- Static Configuration
- Dynamic Configuration
- DOCA Flow Modes
- VNF Mode
- Switch Mode

**DOCA Libraries**

**Compiling the Application**

- Prerequisites
- Compiling All Applications
- Compiling Only the Current Application
- Troubleshooting

**Running the Application**

- Prerequisites
- Application Execution
- Command Line Flags
- Static Configuration IPsec Rules
- Dynamic Configuration IPsec Rules
- Troubleshooting
## Application Code Flow

## Keying Daemon Integration (StrongSwan)

- End-to-end Architecture
- Running the Solution
- Building strongSwan

## References
This document provides an IPsec security gateway implementation 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**

**Note**

DOCA IPsec Security Gateway is supported at alpha level.

DOCA IPsec Security Gateway leverages the DPU's hardware capability for secure network communication. The application demonstrates how to insert rules related to IPsec encryption and decryption based on the DOCA Flow library.

The application demonstrates how to insert rules to create an IPsec tunnel.

**Note**
The application can be configured to receive IPsec rules in one of the following ways:

- **Static configuration** – (default) receives a fixed list of rules for IPsec encryption and decryption

  ![Note]

  When creating the security association (SA) object, the application gets the key, salt, and other SA attributes from the JSON input file.

- **Dynamic configuration** – receives IPsec encryption and decryption rules during runtime through a Unix domain socket (UDS) which is enabled when providing a socket path to the application

  ![Note]

  You may find an example of integrating a rules generator with the application under strongSwan project (DOCA plugin).

The application supports the following IPsec modes: Tunnel, transport, UDP transport.
System Design

DOCA IPsec Security Gateway is designed to run with 2 ports, secured and unsecured:

- Secured port – BlueField receives IPsec encrypted packets and, after decryption, they are sent through the unsecured port

- Unsecured port – BlueField receives regular (plain text) packets and, after encryption, they are sent through the secured port

Example packet path for hardware (HW) offloading:
Example packet path for partial software processing (handling encap/decap in software):

Using the application with SF:
**Application Architecture**

**Static Configuration**

1. Open two DOCA devices, one for the secured port and another for the unsecured port.
2. With the open DOCA devices, the application probes DPDK ports and initializes DOCA Flow and DOCA Flow ports accordingly.

3. On the created ports, build DOCA Flow pipes.

4. In a loop according to the JSON rules:
   1. Create IPSec SA shared resource for the new rule.
   2. Insert encrypt or decrypt rule to DOCA Flow pipes.

## Dynamic Configuration

1. Open two DOCA devices, one for the secured port and another for the unsecured port.

2. With the open DOCA devices, the application probes DPDK ports and initializes DOCA Flow and DOCA Flow ports accordingly.

3. On the created ports, build DOCA Flow pipes.

4. Create UDS socket and listen for incoming data.

5. While waiting for new IPsec policies to be received in a loop, if a new IPsec policy is received:
   1. Parse the policy whether it is an encryption or decryption rule.
2. Create IPSec SA shared resource for the new rule.
3. Insert encrypt or decrypt rule to DOCA Flow pipes.

**DOCA Flow Modes**

The application can run in two modes, **vnf** and **switch**. For more information about the modes, please refer to "Pipe Mode" in the DOCA Flow.

**VNF Mode**

**Encryption**

1. The application builds pipes for encryption. Control pipe as root with four entries that match L3 and L4 types and forward the traffic to the relevant pipes.
1. IPv6 pipes – match the source IP address and forward the traffic to a pipe that matches 5-tuple excluding the source IP.

2. In the 5-tuple match pipes set action of "set meta data", the metadata would be the rule's index in the JSON file.

3. The matched packet is forwarded to the second port.

2. In the secured egress domain, the IP classifier pipe sends the packets to the correct encryption pipe (IPv4 or IPv6) which has a shared IPsec encrypt action. According to the metadata match, the packet is encrypted with the encap destination IP and SPI as defined in the user's rules.

**Decryption**

1. The application builds pipes for decryption. Control pipe as root with two entries that match L3 type and forward the traffic to the relevant decrypt pipe.
2. The decrypt pipe matches the destination IP and SPI according to the rule files and has a shared IPsec action for decryption.

3. After decryption, the matched packets are forwarded to the decap pipe and, if the syndrome is non-zero, the packets are dropped. Otherwise, the packets decap the ESP header and forward to the second port.

1. In debug mode, if syndrome is non-zero, then it sends to bad syndrome pipe to match on the syndrome, count and drop/send to application.

Switch Mode

In switch mode, an ingress root pipe matches the source port to decide what the next pipe is:

- Based on the port, the packet passes through almost the same path as VNF mode and the metadata is set. Afterwards, the packet moves to egress root pipe.

In egress root pipe, the match is on encrypt and decrypt bits that were set in the packet meta:

- Decrypt bit is 1 – packet finishes the decrypt path and must be sent to the unsecure port
Encrypt bit is 1 – packet almost finishes the encrypt path and must be sent to the encrypt pipe on the secure egress domain and to the secure port from there

DOCA Libraries

This application leverages the following DOCA libraries:

- **DOCA Flow**

Refer to their respective programming guide for more information.

Compiling the Application

ℹ️ **Info**

Please refer to the NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

💡 **Tip**

For more information about the applications as well as development and compilation tips, refer to the DOCA Applications page.

The sources of the application can be found under the application's directory: `/opt/mellanox/doca/applications/ipsec_security_gw/`. 
Prerequisites

The application relies on the json-c open source, hence requires the following installation:

- Ubuntu/Debian:
  
  $ sudo apt install libjson-c-dev

- CentOS/RHEL:
  
  $ sudo yum install json-c-devel

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
    cd /opt/mellanox/doca/applications/
    meson /tmp/build
    ninja -C /tmp/build
```

Info

doca_ipsec_security_gw is created under /tmp/build/ipsec_security_gw/.
Compiling Only the Current Application

To directly build only the IPsec Security Gateway application:

```
cd /opt/mellanox/doca/applications/
meson /tmp/build -Denable_all_applications=false -Denable_ipsec_security_gw=true
ninja -C /tmp/build
```

![Info]

doca_ipsec_security_gw is created under /tmp/build/ipsec_security_gw/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
   - Set enable_all_applications to false
   - Set enable_ipsec_security_gw to true

2. Run the following compilation commands:

```
cd /opt/mellanox/doca/applications/
meson /tmp/build
ninja -C /tmp/build
```

![Info]

doca_ipsec_security_gw is created under /tmp/build/ipsec_security_gw/.
Troubleshooting

Refer to the NVIDIA DOCA Troubleshooting Guide for any issue encountered with the compilation of the application.

Running the Application

Prerequisites

1. The IPsec security gateway application is based on DOCA Flow. Therefore, the user is required to allocate huge pages.

```
  echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

**Note**

On some operating systems (RockyLinux, OpenEuler, CentOS 8.2) the default huge page size on the DPU (and Arm hosts) is larger than 2MB, and is often 512MB instead. Once can find out the size of the huge pages using the following command:

```
  $ grep -i huge /proc/meminfo
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnonHugePages</td>
<td>0 kB</td>
</tr>
<tr>
<td>ShmemHugePages</td>
<td>0 kB</td>
</tr>
<tr>
<td>FileHugePages</td>
<td>0 kB</td>
</tr>
<tr>
<td>HugePages_Total</td>
<td>4</td>
</tr>
<tr>
<td>HugePages_Free</td>
<td>4</td>
</tr>
<tr>
<td>HugePages_Rsvd</td>
<td>0</td>
</tr>
<tr>
<td>HugePages_Surp</td>
<td>0</td>
</tr>
<tr>
<td>Hugepagesize</td>
<td>524288 kB</td>
</tr>
</tbody>
</table>

Hugetlb: 6291456 kB

Given that the guiding principal is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, one should allocate the matching amount (8 pages):

echo '8' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-524288kB/nr_hugepages

2. VNF mode – the IPsec security gateway application requires disabling some of the hardware tables:

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode legacy

echo none > /sys/class/net/p0/compat/devlink/encap
echo none > /sys/class/net/p1/compat/devlink/encap

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode switchdev
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode switchdev

To restore the old configuration:

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode legacy

echo basic > /sys/class/net/p0/compat/devlink/encap
echo basic > /sys/class/net/p1/compat/devlink/encap

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode switchdev
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode switchdev

3. Switch mode – the IPsec security gateway application requires configuring the ports to run in switch mode:
To restore the old configuration:

```
sudo mlxconfig -d /dev/mst/mt41686(mt41692)_pciconf0 s LAG_RESOURCE_ALLOCATION=1
# power cycle the host to apply this setting

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode legacy

sudo devlink dev param set pci/0000:03:00.0 name esw_pet_insert value false cmode runtime
sudo devlink dev param set pci/0000:03:00.1 name esw_pet_insert value false cmode runtime

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode switchdev
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode switchdev

sudo devlink dev param set pci/0000:03:00.0 name esw_multiport value true cmode runtime
sudo devlink dev param set pci/0000:03:00.1 name esw_multiport value true cmode runtime
```

**Note**

Make sure to perform **graceful shutdown** prior to power cycling the host.

To restore the old configuration:

```
sudo devlink dev param set pci/0000:03:00.0 name esw_multiport value false cmode runtime
sudo devlink dev param set pci/0000:03:00.1 name esw_multiport value false cmode runtime
```

**Application Execution**

The IPsec Security Gateway application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:
Usage: doca_ipsec_security_gw [DOCA Flags] [Program Flags]

DOCA Flags:
- `-h, --help` Print a help synopsis
- `-v, --version` Print program version information
- `-l, --log-level` Set the (numeric) log level for the program: <10=DISABLE, 20=Critical, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  --sdk-log-level Set the SDK (numeric) log level for the program: <10=DISABLE, 20=Critical, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  `-j, --json <path>` Parse all command flags from an input json file

Program Flags:
- `-s, --secured` secured port pci-address
- `-u, --unsecured` unsecured port pci-address
- `-c, --config` Path to the JSON file with application configuration
- `-m, --mode` ipsec mode - {tunnel/transport/udp_transport}
- `-i, --ipc` IPC socket file path
- `-sn, --secured-name` secured port interface name
- `-un, --unsecured-name` unsecured port interface name
- `-n, --nb-cores` number of cores
- `--debug` Enable debug counters

Info

This usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
./doca_ipsec_security_gw -h
```
2. CLI example for running the application on the BlueField or host:

- **Static Configuration:**

  ```
  ./doca_ipsec_security_gw -s 03:00.0 -u 03:00.1 -c ./ipsec_security_gw_config.json -m transport
  ```

  **Note**
  
  Both the PCIe address identifiers ( -s and -u flags) should match the addresses of the desired PCIe devices.

- **Dynamic Configuration:**

  ```
  ./doca_ipsec_security_gw -s 03:00.0 -u 03:00.1 -c ./ipsec_security_gw_config.json -m transport -i /tmp/rules_socket
  ```

  **Note**
  
  Both the PCIe address identifiers ( -s and -u flags) should match the addresses of the desired PCIe devices.

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:
./doca_ipsec_security_gw --json [json_file]

For example

./doca_ipsec_security_gw --json ipsec_security_gw_params.json

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>h</td>
<td>help</td>
<td>Prints a help synopsis</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>v</td>
<td>version</td>
<td>Prints program version information</td>
<td>N/A</td>
</tr>
<tr>
<td>Flag Type</td>
<td>Short Flag</td>
<td>Long Flag/JSON Key</td>
<td>Description</td>
<td>JSON Content</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
|           | I          | log-level         | Set the log level for the application:  
- DISABLE=10  
- CRITICAL=20  
- ERROR=30  
- WARNING=40  
- INFO=50  
- DEBUG=60  
- TRACE=70 ( requires compilation with TRACE log level support ) | "log-level": 60 |
| N/A       |            | sdk-log-level     | Sets the log level for the program:  
- DISABLE=10  
- CRITICAL=20  
- ERROR=30  
- WARNING=40  
- INFO=50  
- DEBUG=60  
- TRACE=70 | "sdk-log-level": 40 |
| j         |            | json              | Parse all command flags from an input json file | N/A |
|           |            | config            | Path to JSON file with configurations | "config":  
security_gateway_config.json" |
<p>|           |            | unsecured         | PCIe address for the unsecured port | &quot;unsecured&quot;:  &quot;03:00.1&quot; |</p>
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s</td>
<td>secured</td>
<td>PCIe address for the secured port</td>
<td>&quot;secured&quot;: &quot;03:00.0&quot;</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>mode</td>
<td>IPSec mode. Possible values: tunnel, transport, udp_transport</td>
<td>&quot;mode&quot;: &quot;tunnel&quot;</td>
</tr>
<tr>
<td></td>
<td>un</td>
<td>unsecured-name</td>
<td>Interface name of the unsecured port</td>
<td>&quot;unsecured-name&quot;: &quot;p1&quot;</td>
</tr>
<tr>
<td></td>
<td>sn</td>
<td>secured-name</td>
<td>Interface name of the secured port</td>
<td>&quot;secured-name&quot;: &quot;p0&quot;</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>ipc</td>
<td>IPC socket file path for receiving IPSec rules during runtime</td>
<td>&quot;ipc&quot;: &quot;/tmp/rules_socket&quot;</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>nb-cores</td>
<td>Number of cores</td>
<td>&quot;nb-cores&quot;: 8</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>debug</td>
<td>Add counters to all the entries</td>
<td>&quot;debug&quot;: true</td>
</tr>
</tbody>
</table>

**Info**

Refer to DOCA Arg Parser for more information regarding the supported flags and execution modes.
## Static Configuration IPsec Rules

IPsec rules and other configuration can be added with a JSON config file which is passed using the `--config` parameter.

<table>
<thead>
<tr>
<th>Section</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>switch</td>
<td>bool</td>
<td>Configures whether DOCA Flow runs in VNF (false) or switch (true) mode</td>
<td>&quot;switch&quot;: true</td>
</tr>
<tr>
<td></td>
<td>esp-header-offload</td>
<td>string</td>
<td>Decap and encap offloading: both, encap, decap, or none. Default is both (offloading both encap and decap).</td>
<td>&quot;esp-header-offload&quot;: &quot;none&quot;</td>
</tr>
<tr>
<td>sw-sn-inc-enable</td>
<td>bool</td>
<td></td>
<td>Increments sequence number of ESP in software if set to true. Default is false.</td>
<td>&quot;sw-sn-inc-enable&quot;: true</td>
</tr>
<tr>
<td></td>
<td>sw-antireplay-enable</td>
<td>bool</td>
<td>Enables anti-replay mechanism in software if set to true. Default is false.</td>
<td>&quot;sw-antireplay-enable&quot;: true</td>
</tr>
</tbody>
</table>

### Note

Available only if `esp_header_offload` is decap or none.

---

"sw-sn-inc-enable": true

---

"sw-antireplay-enable": true

---

"esp-header-offload": "none"
<table>
<thead>
<tr>
<th>Section</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Note</strong></td>
<td></td>
<td>Window size is 64. Not ESN. Supports non-zero sn_initial.</td>
</tr>
<tr>
<td>sn-initial</td>
<td>ui int</td>
<td></td>
<td>Initial sequence number for ESP header. Used also when sw_antireplay_enable is true. Default is 0.</td>
</tr>
<tr>
<td>debug</td>
<td>bool</td>
<td></td>
<td>Set debug counter for all entries when true. Default is false. This parameter is also used from CLI, will be taken as true if was sent in one of them.</td>
</tr>
<tr>
<td>fwd-bad-syndrome</td>
<td>string</td>
<td></td>
<td>Forward packets that has bad syndrome: drop, RSS. Default is drop.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
<td></td>
<td>Only available in debug mode.</td>
</tr>
<tr>
<td>perf-measurements</td>
<td>string</td>
<td></td>
<td>Possible values: none, insertion-rate, bandwidth, both. Default is none.</td>
</tr>
<tr>
<td></td>
<td>insertion-rate</td>
<td></td>
<td>- insertion-rate – print the total time it took to add the entries</td>
</tr>
<tr>
<td></td>
<td>bandwidth</td>
<td></td>
<td>- bandwidth – optimize the pipe to improve pps for IPv6</td>
</tr>
<tr>
<td>vxlan-encap</td>
<td>bool</td>
<td></td>
<td>When true, preform vxlan-encap after encryption and decap before decryption. Default is false.</td>
</tr>
<tr>
<td>Section</td>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>vni</td>
<td>uint</td>
<td>When vxlan-encap is true, use this vni value in the VXLAN tunnel.</td>
</tr>
<tr>
<td></td>
<td>marker-encap</td>
<td>bool</td>
<td>When true, add an extra non-ESP marker of 8 bytes. Default is false.</td>
</tr>
<tr>
<td>encrypt_rules</td>
<td>ip-version</td>
<td>int</td>
<td>Source and destination IP version. Possible values: 4, 6. Optional; default is 4.</td>
</tr>
<tr>
<td></td>
<td>src-ip</td>
<td>string</td>
<td>Source IP to match</td>
</tr>
<tr>
<td></td>
<td>protocol</td>
<td>string</td>
<td>L4 protocol: TCP or UDP</td>
</tr>
<tr>
<td></td>
<td>src-port</td>
<td>int</td>
<td>Source port to match</td>
</tr>
<tr>
<td></td>
<td>dst-port</td>
<td>int</td>
<td>Destination port to match</td>
</tr>
<tr>
<td></td>
<td>encap-ip-version</td>
<td>int</td>
<td>Encap IP version: 4 or 6. Optional; default is 4.</td>
</tr>
<tr>
<td></td>
<td>encap-dst-ip</td>
<td>string</td>
<td>Encap destination IP</td>
</tr>
</tbody>
</table>

**Note**
<table>
<thead>
<tr>
<th>Section</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>spi</td>
<td>int</td>
<td>SPI integer to set in the ESP header</td>
<td>&quot;spi&quot;: 5</td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>string</td>
<td>Key for creating the SA (in hex format)</td>
<td>&quot;key&quot;: &quot;112233445566778899aabbcddd&quot;</td>
</tr>
<tr>
<td></td>
<td>key-type</td>
<td>int</td>
<td>Key size: 128 or 256. Optional; default is 256.</td>
<td>&quot;key-type&quot;: 128</td>
</tr>
<tr>
<td></td>
<td>salt</td>
<td>int</td>
<td>Salt value for creating the SA. Default is 6.</td>
<td>&quot;salt&quot;: 1212</td>
</tr>
<tr>
<td></td>
<td>icv-length</td>
<td>int</td>
<td>ICV length value: 8, 12, or 16. Default is 16.</td>
<td>&quot;icv-length&quot;: 12</td>
</tr>
<tr>
<td></td>
<td>lifetime-threshold</td>
<td>int</td>
<td>Set IPsec lifetime threshold. Ignored if sw-sn-inc-enable is true. Default is 0.</td>
<td>&quot;lifetime-threshold&quot;: 1000000</td>
</tr>
<tr>
<td></td>
<td>esn_en</td>
<td>bool</td>
<td>Enables extended sequence number. Default is false.</td>
<td>&quot;esn_en&quot;: true</td>
</tr>
<tr>
<td></td>
<td>decrypt_rules</td>
<td>ip-version</td>
<td>int</td>
<td>Destination IP version: 4 or 6. Optional; default is 4.</td>
</tr>
<tr>
<td>Section</td>
<td>Field</td>
<td>Type</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>dst-ip</td>
<td>str</td>
<td>Destination IP to match</td>
<td>&quot;dst-ip&quot;: &quot;1122:3344:5566:7788:99aa:bbcc:dee:ff00&quot;</td>
</tr>
<tr>
<td></td>
<td>inner-ip-version</td>
<td>int</td>
<td>Inner IP version: 4 or 6. Optional; default is 4.</td>
<td>&quot;inner-ip-version&quot;: 4</td>
</tr>
<tr>
<td></td>
<td>spi</td>
<td>int</td>
<td>SPI to match in the ESP header</td>
<td>&quot;spi&quot;: 5</td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>str</td>
<td>Key for creating the SA (in hex format)</td>
<td>&quot;key&quot;: &quot;112233445566778899aabbccdd&quot;</td>
</tr>
<tr>
<td></td>
<td>key-type</td>
<td>int</td>
<td>Key size: 128 or 256. Optional; default is 256.</td>
<td>&quot;key-type&quot;: 128</td>
</tr>
<tr>
<td></td>
<td>salt</td>
<td>int</td>
<td>Salt value for creating the SA. Default is 6.</td>
<td>&quot;salt&quot;: 1212</td>
</tr>
<tr>
<td></td>
<td>icv-length</td>
<td>int</td>
<td>ICV length value: 8, 12, or 16. Default is 16.</td>
<td>&quot;icv-length&quot;: 12</td>
</tr>
<tr>
<td>Section</td>
<td>Field</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lifetime-threshold</td>
<td>int</td>
<td>Set IPsec lifetime threshold. Ignored if sw-antireplay-enable is true. Default is 0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>esn_en</td>
<td>bool</td>
<td>Enables extended sequence number. Default is false.</td>
<td></td>
</tr>
</tbody>
</table>

### Dynamic Configuration IPsec Rules

The application listens on the UDS socket for receiving a predefined structure for the IPsec policy defined in the policy.h file.

The client program or keying daemon should connect to the socket with the same socket file path provided to the application by the --ipc/-i flags, and send the policy structure as packed to the application through the same socket.

ℹ️ **Note**

In the dynamic configuration, the application uses the config section from the JSON config file and ignores the encrypt_rules and decrypt_rules sections.

The IPsec policy structure:

```c
struct ipsec_security_gw_ipsec_policy {
    /* Protocols attributes */
    uint16_t src_port;       /* Policy inner source port */
    uint16_t dst_port;       /* Policy inner destination port */
    uint8_t l3_protocol;     /* Policy L3 proto {POLICY_L3_TYPE_IPV4, POLICY_L3_TYPE_IPV6} */
    uint8_t l4_protocol;     /* Policy L4 proto {POLICY_L4_TYPE_UDP, POLICY_L4_TYPE_TCP} */
};
```
uint8_t outer_l3_protocol;        /* Policy outer L3 type {POLICY_L3_TYPE_IPV4, POLICY_L3_TYPE_IPV6} */
/* Policy attributes */
uint8_t policy_direction;        /* Policy direction {POLICY_DIR_IN, POLICY_DIR_OUT} */
uint8_t policy_mode;             /* Policy IPSEC mode {POLICY_MODE_TRANSPORT, POLICY_MODE_TUNNEL} */

/* Security Association attributes */
uint8_t esn;                     /* Is ESN enabled? */
uint8_t icv_length;              /* ICV length in bytes {8, 12, 16} */
uint8_t key_type;                /* AES key type {POLICY_KEY_TYPE_128, POLICY_KEY_TYPE_256} */
uint32_t spi;                    /* Security Parameter Index */
uint32_t salt;                   /* Cryptographic salt */
uint8_t enc_key_data[MAX_KEY_LEN]; /* Encryption key (binary) */

/* Policy inner and outer addresses */
char src_ip_addr[MAX_IP_ADDR_LEN + 1]; /* Policy inner IP source address in string format */
char dst_ip_addr[MAX_IP_ADDR_LEN + 1]; /* Policy inner IP destination address in string format */
char outer_src_ip[MAX_IP_ADDR_LEN + 1]; /* Policy outer IP source address in string format */
char outer_dst_ip[MAX_IP_ADDR_LEN + 1]; /* Policy outer IP destination address in string format */

};

1 Note

The policy type, whether it is encrypted or decrypted, is classified according to the policy_direction attribute:

- POLICY_DIR_IN  – decryption policy
- POLICY_DIR_OUT – encryption policy

Troubleshooting
Refer to the NVIDIA DOCA Troubleshooting Guide for any issue encountered with the installation or execution of the DOCA applications.

**Application Code Flow**

1. Parse application argument.
   1. Initialize arg parser resources and register DOCA general parameters.
      
      ```
      doca_argp_init();
      ```
   2. Register the application's parameters.
      
      ```
      register_ipsec_security_gw_params();
      ```
   3. Parse the arguments.
      
      ```
      doca_argp_start();
      ```

1. Parse app parameters.

2. DPDK initialization.
   
   ```
   rte_eal_init();
   ```

   Call `rte_eal_init()` to initialize EAL resources with the provided EAL flags for not probing the ports.

4. Initialize devices and ports.

1. Open DOCA devices with input PCIe addresses / interface names.
2. Probe DPDK port from each opened device.

5. Initialize and start DPDK ports.

1. Initialize DPDK ports, including mempool allocation.
2. Initialize hairpin queues if needed.
3. Binds hairpin queues of each port to its peer port.


1. Initialize DOCA Flow library.
2. Find the indices of the DPDK-probed ports and start DOCA Flow ports with them.

7. Insert rules.

1. Insert encryption rules.
2. Insert decryption rules.

8. Wait for traffic.

1. wait in a loop until the user terminates the program.

9. IPsec security gateway cleanup:

1. DOCA Flow cleanup; destroy initialized ports.

```c
  doca_flow_cleanup();
```

2. SA destruction.

```c
  ipsec_security_gw_destroy_sas();
```

3. IPsec objects destruction.

```c
  ipsec_security_gw_ipsec_ctx_destroy();
```

4. Destroy DPDK ports and queues.
5. DPDK finish.

```c
dpdk_queues_and_ports_fini();

dpdk_fini();

calls rte_eal_destroy() to destroy initialized EAL resources.

6. Arg parser destroy.

```c
doca_argp_destroy()
```

**Keying Daemon Integration (StrongSwan)**

strongSwan is a keying daemon that uses the Internet Key Exchange Version 2 (IKEv2) protocol to establish SAs between two peers. strongSwan includes a DOCA plugin that is part of the strongSwan package in BFB. The plugin is loaded only if the DOCA IPsec Security Gateway is triggered. The plugin connects to UDS socket and sends IPsec policies to the application after the key exchange completes.

For more information about the key daemon, refer to [strongSwan documentation](#).

**End-to-end Architecture**

The following diagram presents an architecture where two BlueField DPUs are connected to each other with DOCA IPsec Security Gateway running on each.
swanctl is a command line tool used for strongSwan IPsec configuration:

1. Run DOCA IPsec Security Gateway on both sides in a dynamic configuration.

2. Start strongSwan service.

3. Configure strongSwan IPsec using the swanctl.conf configuration file on both sides.
4. Start key exchange between the two peers. At the end of the flow, the result arrives to the DOCA plugin, populates the policy-defined structure, and sends it to the socket.

5. DOCA IPsec Security Gateway on both sides reads new policies from the socket, performs the parsing, creates a DOCA SA object, and adds flow decrypt/encrypt entry.

This architecture uses P1 uplink on both BlueField DPUs to run the strongSwan key daemon. To configure the uplink:

1. Configure an IP addresses for the PFs of both DPUs:

   1. On BF1:

   ```bash
   ip addr add 192.168.50.1/24 dev p1
   ```

   2. On BF2:

   ```bash
   ip addr add 192.168.50.2/24 dev p1
   ```

2. Verify the connection between two BlueField DPUs.

   ```bash
   BF1> ping 192.168.50.2
   ```

   **Note**

   It is possible to configure multiple IP addresses to uplinks to run key exchanges with different policy attributes.
3. Configure the swanctl.conf files for each machine. The file should be located under /etc/swanctl/conf.d/.

Adding swanctl.conf file examples:

- Transport mode

  - swanctl.conf example for BF1:

```plaintext
connections {
  BF1-BF2 {
    local_addrs = 192.168.50.1
    remote_addrs = 192.168.50.2
    rekey_time = 0

    local {
      auth = psk
      id = host1
    }
    remote {
      auth = psk
      id = host2
    }
  }

  children {
    bf {
      local_ts = 192.168.50.1/32 [udp/60]
      remote_ts = 192.168.50.2/32 [udp/90]
      esp_proposals = aes128gcm128-x25519-esn
      mode = transport
      policies_fwd_out = yes
      life_time = 0
    }
  }

  version = 2
}
```

Note: Make sure that the uplink is not in OVS bridges.
mobike = no
reauth_time = 0
proposals = aes128-sha256-x25519
}
}

secrets {
ike-BF {
id-host1 = host1
id-host2 = host2
secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
}
}

connections {
BF2-BF1 {
local_addr = 192.168.50.2
remote_addr = 192.168.50.1
rekey_time = 0

local {
auth = psk
id = host2
}
remote {
auth = psk
id = host1
}

children {
bf {
local_ts = 192.168.50.2/32 [udp/90]
remote_ts = 192.168.50.1/32 [udp/60]
esp_proposals = aes128gcm128-x25519-esn
mode = transport
life_time = 0
}
}
version = 2
mobike = no

- swanctl.conf example for BF2:
Tunnel mode

connections {
  BF1-BF2 {
    local_addrs = 192.168.50.2
    remote_addrs = 192.168.50.1
    rekey_time = 0

    local {
      auth = psk
      id = host2
    }
    remote {
      auth = psk
      id = host1
    }

    children {
      bf {
        local_ts = 2001:db8:85a3::8a2e:370:7334/128 [udp/3030]
        remote_ts = 2001:db8:85a3::8a2e:370:7335/128 [udp/55]
        esp_proposals = aes128gcm128-x25519-esn
        life_time = 0
      }
    }
  }
  version = 2
  mobike = no
  proposals = aes128-sha256-x25519
}
DOCA IPsec only supports ESP headers, AES-GCM encryption algorithm, and key sizes 128 or 256. Therefore, when setting ESP proposals in the swanctl.conf, please adhere to the values provided in the following table:

<table>
<thead>
<tr>
<th>ESP Proposal</th>
<th>Algorithm Type Including ICV Length</th>
<th>Key Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes128gcm8</td>
<td>ENCR_AES_GCM_ICV8</td>
<td>128</td>
</tr>
<tr>
<td>aes128gcm64</td>
<td>ENCR_AES_GCM_ICV8</td>
<td>128</td>
</tr>
<tr>
<td>aes128gcm12</td>
<td>ENCR_AES_GCM_ICV12</td>
<td>128</td>
</tr>
<tr>
<td>aes128gcm96</td>
<td>ENCR_AES_GCM_ICV12</td>
<td>128</td>
</tr>
<tr>
<td>aes128gcm16</td>
<td>ENCR_AES_GCM_ICV16</td>
<td>128</td>
</tr>
</tbody>
</table>

**Note**

local_ts and remote_ts must have a netmask of /32 for IPv4 addresses and /128 for IPv6 addresses.

**Note**

SA rekey is not supported in DOCA plugin.
connection.rekey_time must be set to 0 and connection.child.life_time must be set to 0.
Running the Solution

Run the following commands on both BlueField peers.

1. Run DOCA IPsec Security Gateway in dynamic configuration, assuming the socket location is /tmp/rules_socket.

```bash
doca_ipsec_security_gw -s 03:00.0 -un <sf_net_dev> -c ./ipsec_security_gw_config.json -m transport -i /tmp/rules_socket
```

**Note**

DOCA IPsec Security Gateway application should be run first.

2. Edit the /etc/strongswan.d/charon/doca.conf file and add the UDS socket path. If the socket_path is not set, the plugin uses the default path /tmp/strongswan_doca_socket.
3. Restart the strongSwan server:

```bash
systemctl restart strongswan.service
```

**Note**

If the application has been run with log level debug, you can see that the connection has been done successfully and the application is waiting for new IPsec policies.

4. Verify that the `swanctl.conf` file exists in `/etc/swanctl/conf.d/` directory.
5. Load IPsec configuration:

```
swanctl --load-all
```

6. Start IKE protocol on either the initiator or the target side:

```
swanctl -i --child <child_name>
```

**Info**

In the example above, the child's name is `bf`.

---

**Building strongSwan**

To perform some changes in the DOCA plugin in strongSwan zone:

1. Verify that the dependencies listed [here](#) are installed in your environment. `libgmp-dev` is missing from that list so make sure to install that as well.

2. Git clone [https://github.com/Mellanox/strongswan.git](https://github.com/Mellanox/strongswan.git).

3. Git checkout BF-5.9.10 branch.

4. Add your changes in the plugin located under `src/libcharon/plugins/doca`.

5. Run `autogen.sh` within the strongSwan repo.
6. Run the following:

```
./configure --enable-openssl --disable-random --prefix=/usr/local --sysconfdir=/etc --enable-systemd --enable-doca
make
make install
systemctl daemon-reload
systemctl restart strongswan.service
```

References

- /opt/mellanox/doca/applications/ipsec_security_gw/

- /opt/mellanox/doca/applications/ipsec_security_gw/ipsec_security_gw_params.json

© Copyright 2024, NVIDIA. PDF Generated on 08/15/2024