



NVIDIA DOCA IPsec Security Gateway

Application Guide

Table of Contents

Chapter 1. Introduction.....	1
Chapter 2. System Design.....	3
Chapter 3. Application Architecture.....	5
3.1. Static Configuration.....	5
3.2. Dynamic Configuration.....	6
3.3. DOCA Flow Modes.....	6
3.3.1. VNF Mode.....	6
3.3.1.1. Encryption.....	7
3.3.1.2. Decryption.....	7
3.3.2. Switch Mode.....	8
Chapter 4. DOCA Libraries.....	10
Chapter 5. Configuration Flow.....	11
Chapter 6. Running the Application.....	13
6.1. Static Configuration IPsec Rules.....	15
6.2. Dynamic Configuration IPsec Rules.....	17
Chapter 7. Arg Parser DOCA Flags.....	19
Chapter 8. Keying Daemon Integration (StrongSwan).....	21
8.1. End-to-end Architecture.....	21
8.2. Running the Solution.....	26
8.3. Building strongSwan.....	27
Chapter 9. References.....	28

Chapter 1. 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 and IPsec libraries.

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



Note: An example for configuring the Internet key exchange (IKE) can be found under [Keying Daemon Integration \(StrongSwan\)](#) but is not considered part of the application.

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



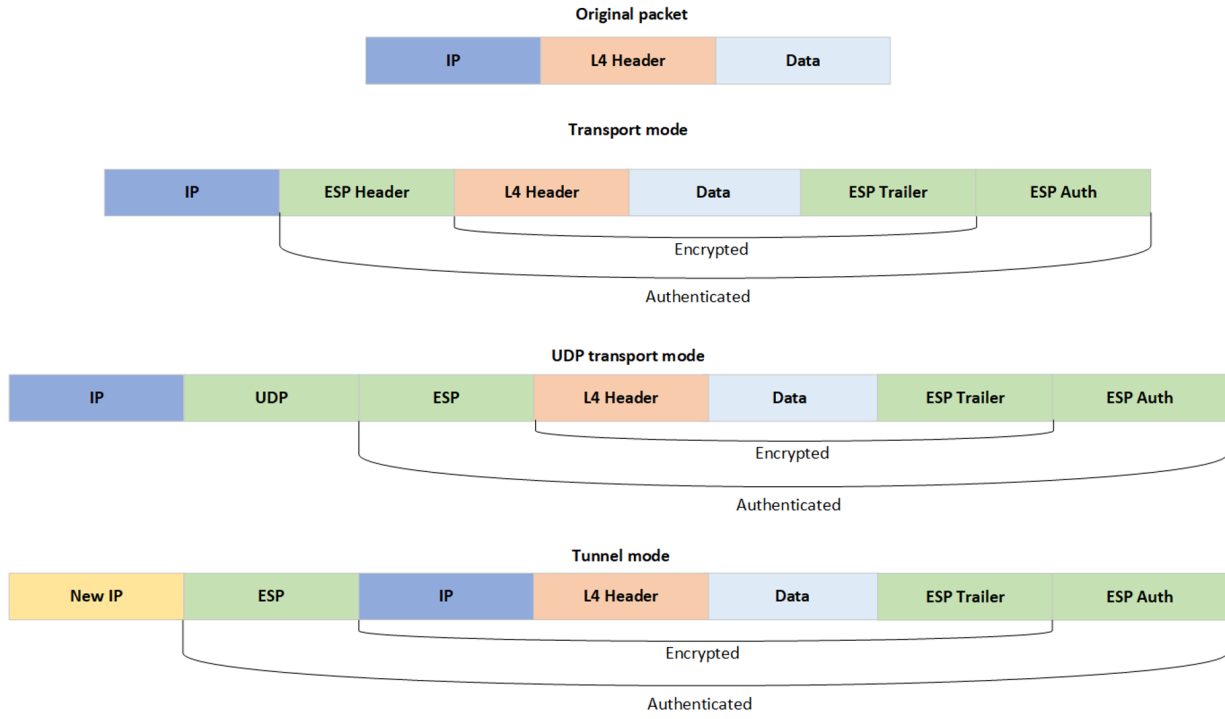
Note: When creating the security association (SA) object, the application gets the key, salt, and other SA attributes from the JSON input file.


- ▶ Dynamic configuration – receives IPsec encryption and decryption rules during runtime through 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.



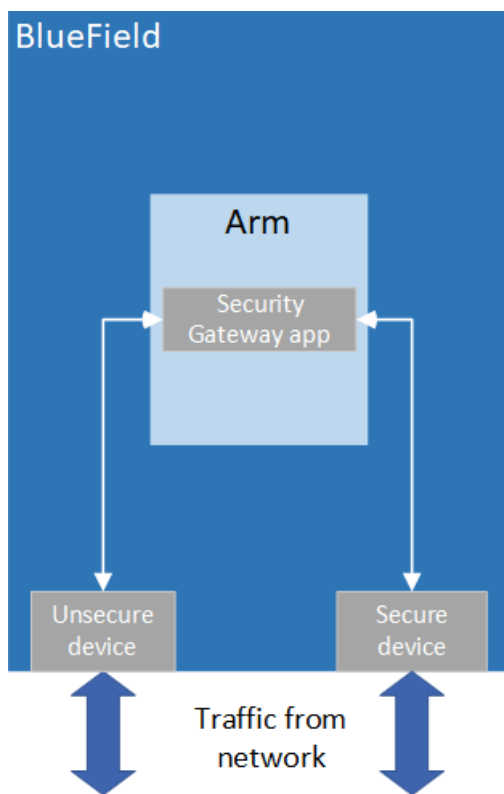
 Note: DOCA IPsec supports only ESP header type.

Chapter 2. System Design

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

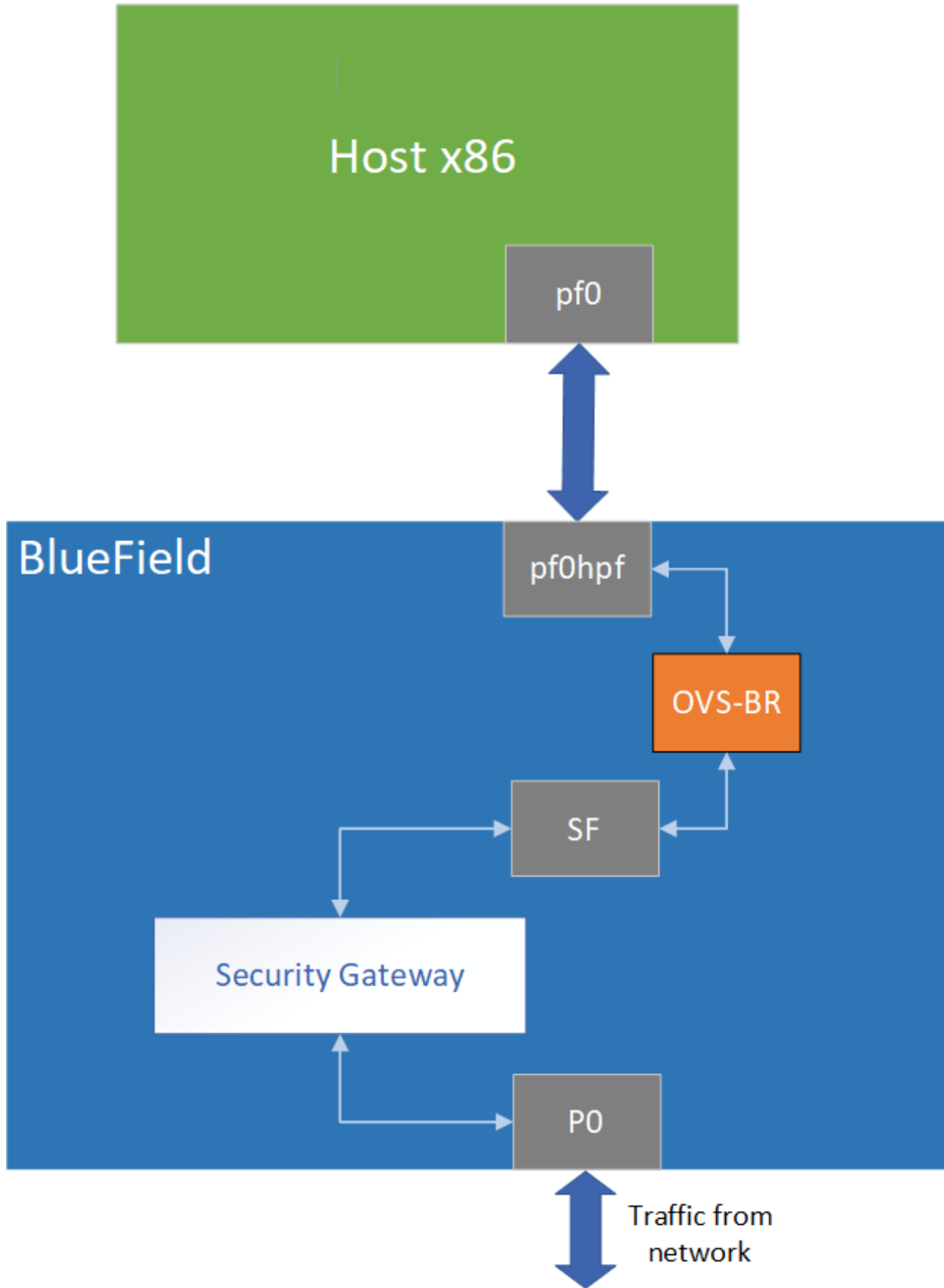
- ▶ Secured port – BlueField receives IPsec encrypted packets and, after decryption, they are sent through the unsecured port
- ▶ Unsecured port – BlueField receives regular (plain text) packets and, after encryption, they are sent through the secured port

Example packet path for hardware offloading:



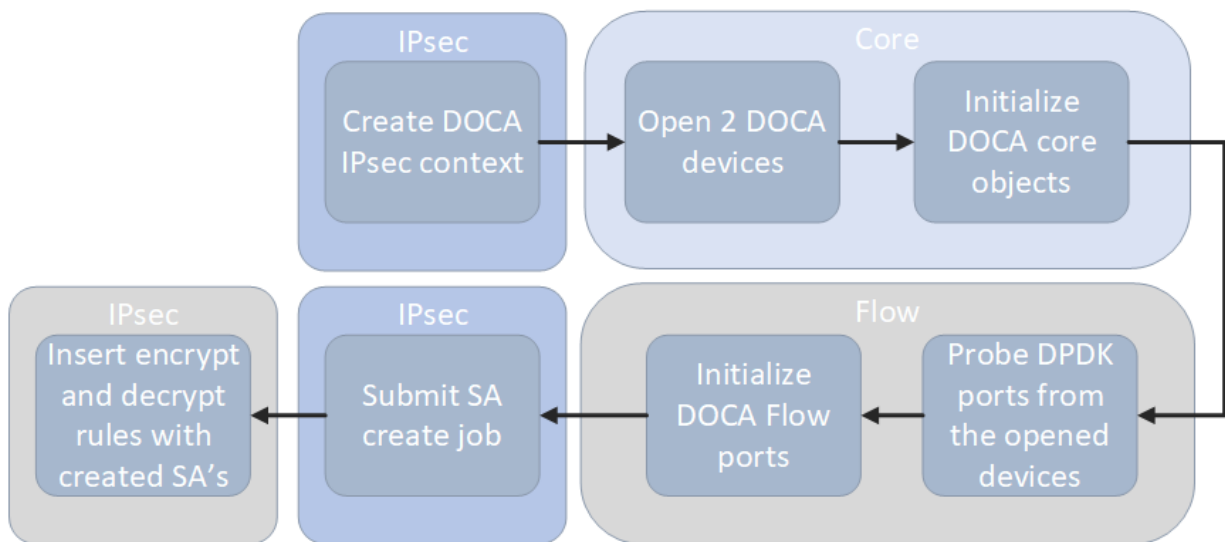
Example packet path for partial software processing (handling encap/decap in software):

Using the application with SF:



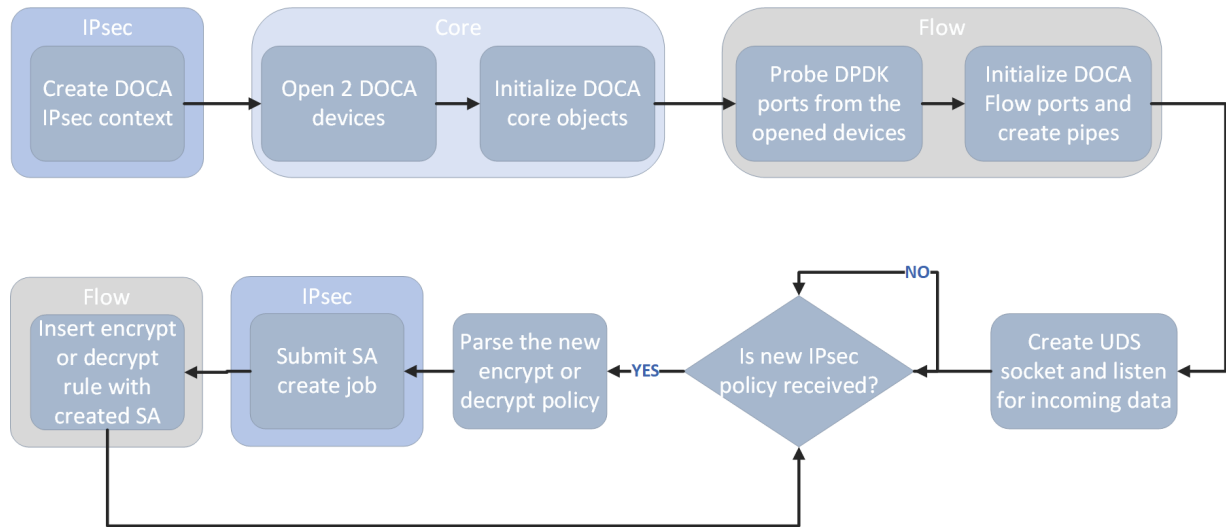
Chapter 3. Application Architecture

3.1. Static Configuration



1. Create IPsec library context.
2. Open two DOCA devices, one for the secured port and another for the unsecured port.
3. Initialize the DOCA work queue.
4. With the open DOCA devices, the application probes DPDK ports and initializes DOCA Flow and DOCA Flow ports accordingly.
5. On the created ports, build DOCA Flow pipes.
6. In a loop according to the JSON rules:
 - a). Create DOCA IPsec SA for the new rule.
 - b). Insert encrypt or decrypt rule to DOCA Flow pipes.

3.2. Dynamic Configuration



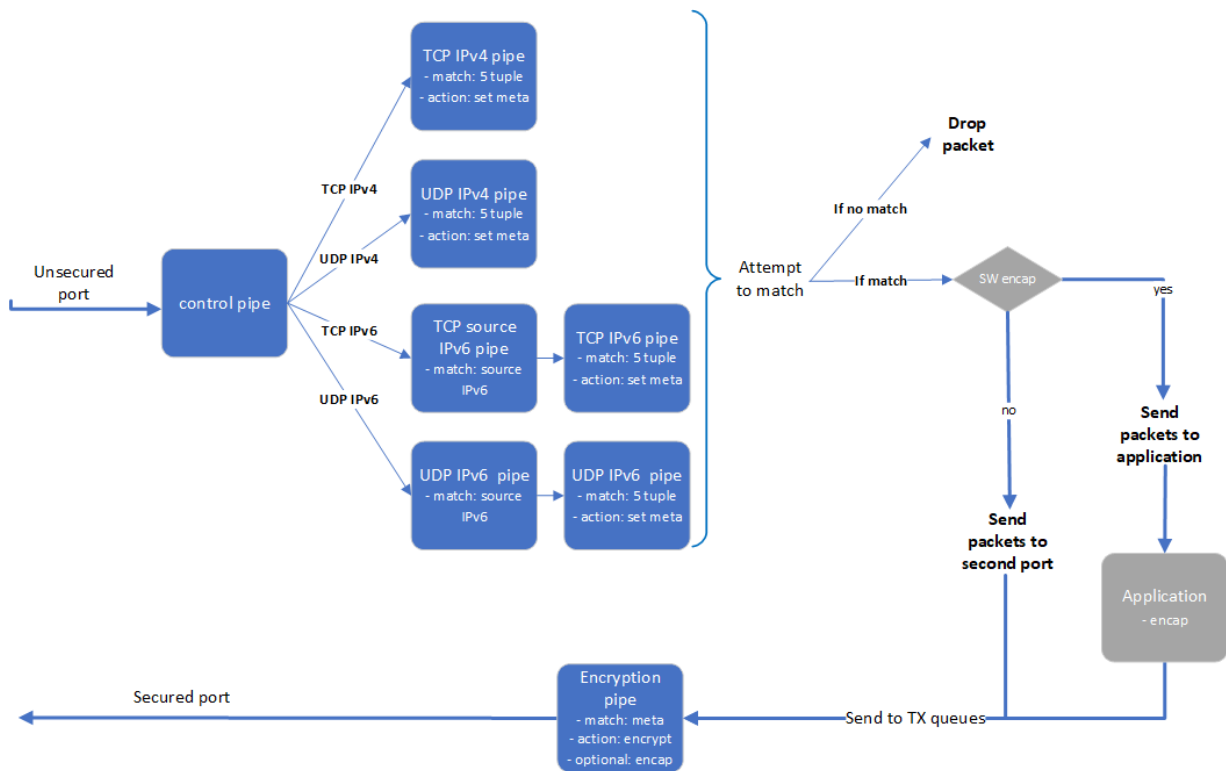
1. Create IPsec library context.
2. Open two DOCA devices, one for the secured port and another for the unsecured port.
3. Initialize the DOCA work queue.
4. With the open DOCA devices, the application probes DPK ports and initializes DOCA Flow and DOCA Flow ports accordingly.
5. On the created ports, build DOCA Flow pipes.
6. Create UDS socket and listen for incoming data.
7. While waiting for new IPsec policies to be received in a loop, if a new IPsec policy is received:
 - a). Parse the policy whether it is an encryption or decryption rule.
 - b). Create DOCA IPsec SA for the new rule.
 - c). Insert encrypt or decrypt rule to DOCA Flow pipes.

3.3. DOCA Flow Modes

The application can run in two modes, `vnf` and `switch`. For more information about the modes, please refer to section "Pipe Mode" in the [NVIDIA DOCA Flow Programming Guide](#).

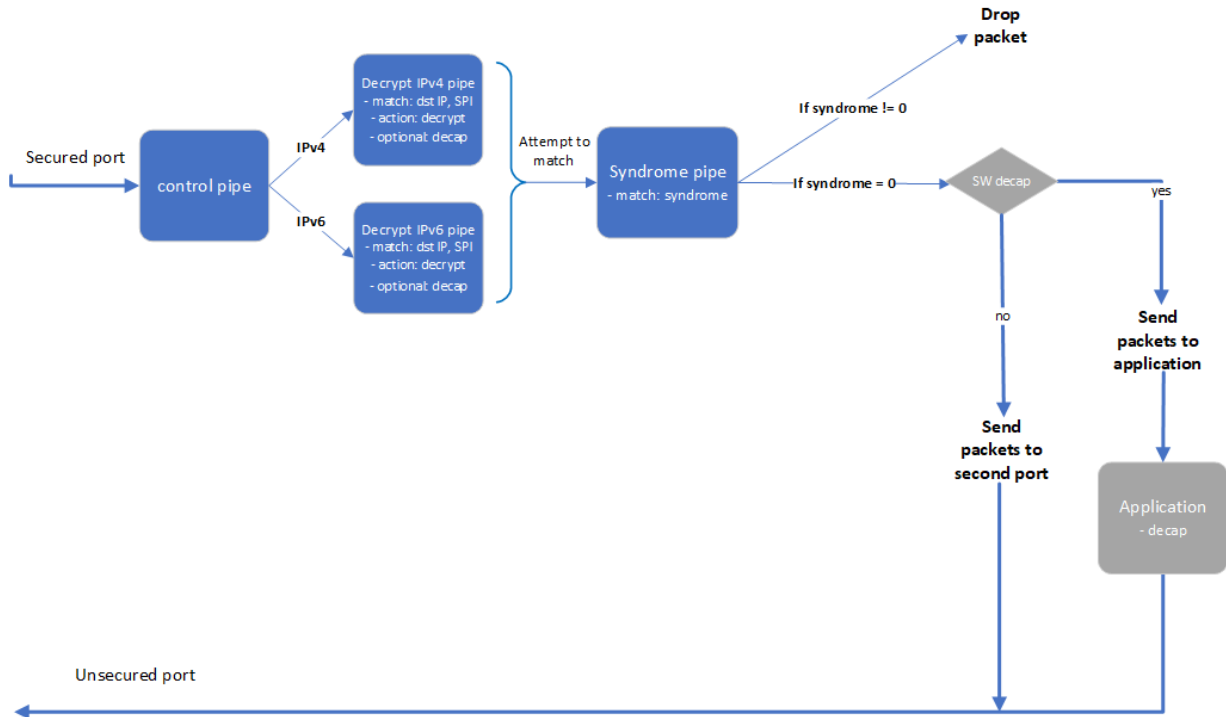
3.3.1. VNF Mode

3.3.1.1. Encryption



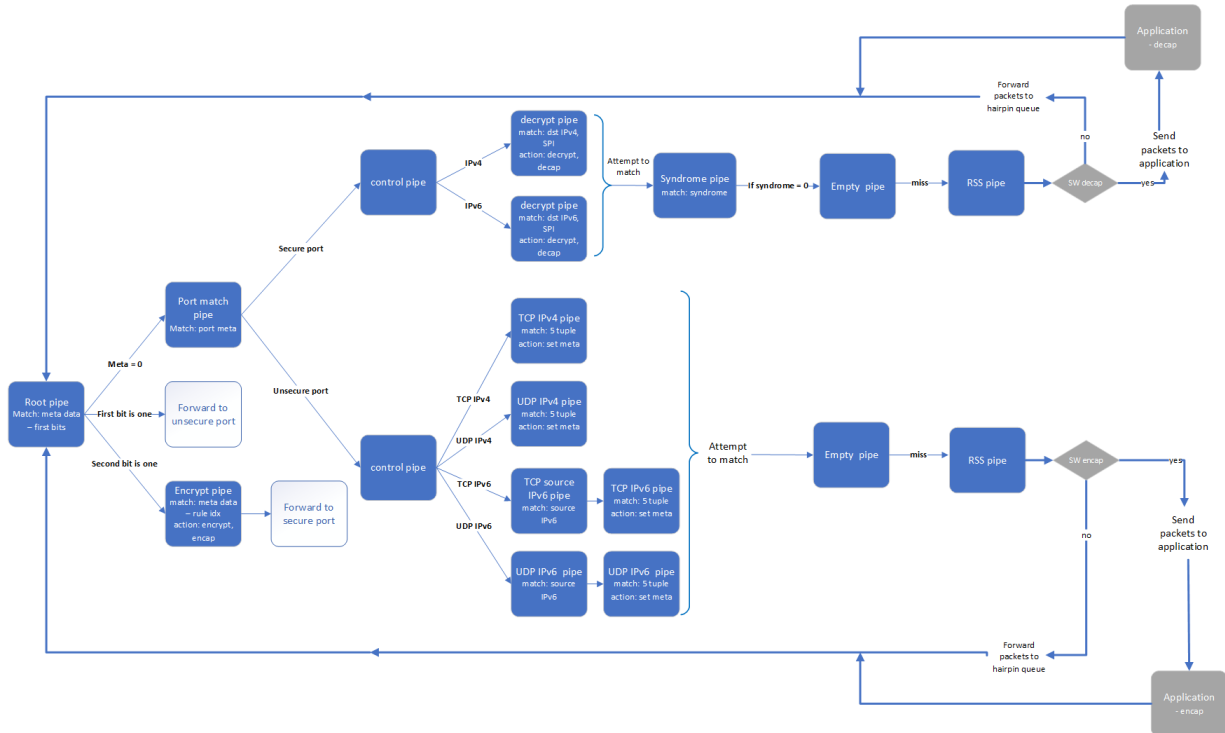
1. The application builds 8 pipes for encryption. Control pipe as root with four entries that match L3 and L4 types and forward the traffic to the relevant pipes.
 - a). IPv6 pipes – match the source IP address and forward the traffic to a pipe that matches 5-tuple excluding the source IP.
 - b). In the 5-tuple match pipes set action of "set meta data", the metadata would be the rule's index in the JSON file.
 - c). The matched packet is forwarded to the second port.
2. In the secured egress domain, there is an encryption pipe that 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.

3.3.1.2. Decryption



1. The application builds 4 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 syndrome pipe and, if the syndrome is non-zero, the packets are dropped. Otherwise, the packets are forwarded to the second port.

3.3.2. Switch Mode



In switch mode, a root pipe matches the first 2 most significant bits (MSBs) to decide what the next pipe is:

- ▶ Metadata is 0 – packet arrives and continues to pipe that matches on the port's meta. Based on the port, the packet passes through almost the same path as VNF mode and the metadata is set in the 2 MSBs. Afterwards, the packet moves to pipes that send the packets to the root pipe.
- ▶ First bit is 1 – packet finishes the decrypt path and must be sent to the unsecure port.
- ▶ Second 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.

Chapter 4. DOCA Libraries

This application leverages the following DOCA libraries:

- ▶ [DOCA Flow library](#)
- ▶ [DOCA IPsec library](#)

Chapter 5. Configuration Flow

1. Parse application argument.

a). Initialize the arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

b). Register application parameters.

```
register_ipsec_security_gw_params();
```

c). Parse application flags.

```
doca_argp_start();
```

i. 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.

3. Parse config file.

```
ipsec_security_gw_parse_config();
```

4. Initialize devices and ports.

```
ipsec_security_gw_init_devices();
```

a). Open DOCA devices with input PCIe addresses.

b). Probe DPDK port from each opened device.

5. Initialize and start DPDK ports.

```
dpdk_queues_and_ports_init();
```

a). Initialize DPDK ports, including mempool allocation.

b). Initialize hairpin queues if needed.

c). Binds hairpin queues of each port to its peer port.

6. Initialize objects for DOCA IPsec library.

```
ipsec_security_gw_ipsec_ctx_create();
```

a). Create IPsec library context.

b). Create DOCA Work queue.

7. Initialize DOCA Flow.

```
ipsec_security_gw_init_doca_flow();
```

a). Initialize DOCA Flow library.

b). Find the indices of the DPDK-probed ports and start DOCA Flow ports with them.

8. Insert rules.**a). Insert encryption rules.**

```
ipsec_security_gw_insert_encrypt_rules();
```

b). Insert decryption rules.

```
ipsec_security_gw_insert_decrypt_rules();
```

9. Wait for traffic.

```
ipsec_security_gw_wait_for_traffic();
```

a). Wait in a loop until the user terminates the program.**10. IPsec security gateway cleanup:****a). DOCA Flow cleanup; destroy initialized ports.**

```
doca_flow_cleanup();
```

b). SA destruction.

```
ipsec_security_gw_destroy_sas();
```

c). IPsec objects destruction.

```
ipsec_security_gw_ipsec_ctx_destroy();
```

d). Destroy DPDK ports and queues.

```
dpdk_queues_and_ports_fini();
```

e). DPDK finish.

```
dpdk_fini();
```

Calls `rte_eal_destroy()` to destroy initialized EAL resources.

f). Arg parser destroy.

```
doca_argp_destroy();
```

Chapter 6. Running the Application

1. Refer to the following documents:

- ▶ [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.
- ▶ [NVIDIA DOCA Troubleshooting Guide](#) for any issue you may encounter with the installation, compilation, or execution of DOCA applications.
- ▶ [NVIDIA DOCA Applications Overview](#) for additional compilation instructions and development tips for the DOCA applications.

2. DOCA IPsec Security Gateway binary is located under `/opt/mellanox/doca/applications/ipsec_security_gw/bin/doca_ipsec_security_gw`. To build all the applications together, run:

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

3. To build only the IPsec security gateway application:

a). Edit the following flags in `/opt/mellanox/doca/applications/meson_option.txt`:

- ▶ Set `enable_all_applications` to `false`
- ▶ Set `enable_ipsec_security_gw` to `true`

b). Run the commands in step 2.



Note: `doca_ipsec_security_gw` will be created under `./build/ipsec_security_gw/src/`.

Application usage:

Usage: `doca_ipsec_security_gw` [DOCA Flags] [Program Flags]

DOCA Flags:

<code>-h, --help</code>	Print a help synopsis
<code>-v, --version</code>	Print program version information
<code>-l, --log-level</code>	Set the log level for the program
<CRITICAL=20, ERROR=30, WARNING=40, INFO=50, DEBUG=60>	

Program Flags:

<code>-s, --secured</code>	Secured port pci-address
<code>-u, --unsecured</code>	Unsecured port pci-address
<code>-c, --config</code>	Path to the JSON file with application configuration
<code>-m, --mode</code>	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
```



Note: For additional information on the application, use `-h`:

```
/opt/mellanox/doca/applications/<application name>/bin/doca_<application
name> -- -h
```

4. Running the application on BlueField:

► Pre-run setup:

- The IPsec security gateway application is based on DPDK libraries. Therefore, the user is required to allocate huge pages:

```
echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

- VNF mode – the IPsec security gateway example 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
```

- Switch mode – the IPsec security gateway application requires configuring the ports to run in switch mode:

```
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
    
```

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
    
```

- ▶ Example for running the application in static configuration:

```

cd /opt/mellanox/doca/applications/ipsec_security_gw/bin
./doca_ipsec_security_gw -s 03:00.0 -u 03:00.1 -c ./
ipsec_security_gw_config.json -m transport
    
```

- ▶ Example for running the application in dynamic configuration:


```

cd /opt/mellanox/doca/applications/ipsec_security_gw/bin
./doca_ipsec_security_gw -s 03:00.0 -u 03:00.1 -c ./
ipsec_security_gw_config.json -m transport -i /tmp/rules_socket
    
```

5. Running the application on the host, CLI example:

```

cd /opt/mellanox/doca/applications/ipsec_security_gw/bin
./doca_ipsec_security_gw -s 08:00.0 -u 08:00.1 -c ./ipsec_security_gw_config.json
-m transport
    
```

 **Note:** Refer to section "Running DOCA Application on Host" in [NVIDIA DOCA Virtual Functions User Guide](#).

6. To run doca_ipsec_security_gw using a JSON file:

```

doca_ipsec_security_gw --json [json_file]
    
```

For example:

```

cd /opt/mellanox/doca/applications/ipsec_security_gw/bin
./doca_ipsec_security_gw --json ./ipsec_security_gw_params.json
    
```

6.1. Static Configuration IPsec Rules

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

Section	Field	Type	Description	Example
config	switch	bool	True for running DOCA Flow in switch mode. Default is false (VNF mode).	<code>"switch": true</code>
	esp_header_offload	string	Decap and encap offloading: both, encap, decap, or none. Default is both (offloading	<code>"esp_header_offload": "none"</code>

Section	Field	Type	Description	Example
encrypt_rules	sw_sn_inc_enable	bool	both encap and decap). Increments sequence number of ESP in software if set to true. Default is false. Available only if esp_header_offload is decap OR none.	<code>"sw_sn_inc_enable": true</code>
	sw_antireplay_enable	bool	Enables anti-replay mechanism in software if set to true. Default is false. Available only if esp_header_offload is encap OR none.	<code>"sw_antireplay_enable": true</code>
	sn_initial	uint	Initial sequence number for ESP header. Used also when sw_antireplay_enable is true. Default is 0.	
	ip-version	int	Source and destination IP version. 4 / 6. Optional. Default is 4.	
	src-ip	string	Source IP to match	<code>"src-ip": "1.2.3.4"</code>
	dst-ip	string	Destination IP to match	<code>"dst-ip": "101:101:101:101:101:101:101:101"</code>
	protocol	string	L4 protocol: TCP or UDP	<code>"protocol"</code>

Note:
Window size is 64.
Not ESN.
Supports non-zero sn_initial.

Section	Field	Type	Description	Example
	src-port	int	Source port to match	
	dst-port	int	Destination port to match	
	encap-ip-version	int	Encap IP version: 4 or 6. Optional; default is 4.	
	encap-dst-ip	string	Encap destination IP. Mandatory for tunnel mode only.	"encap-dst-ip": "1.1.1.1"
	spi	int	SPI integer to set in the ESP header	
	key	string	Key for creating the SA (in hex format)	"key": "112233445566778899aabb"
	key_type	int	Key size: 128 or 256. Optional; default is 256.	
decrypt_rules	ip-version	int	Destination IP version: 4 or 6. Optional; default is 4.	
	dst-ip	string	Destination IP to match	"dst-ip": "1122:3344:5566:7788:99aa"
	inner-ip-version	int	Inner IP version. Mandatory for tunnel mode only. Optional; default is 4.	
	spi	int	SPI to match in the ESP header	
	key	string	Key for creating the SA (in hex format)	"key": "112233445566778899aabb"
	key_type	int	Key size: 128 or 256. Optional; default is 256.	

6.2. 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.



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:

```
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 */
};
```



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

Chapter 7. Arg Parser DOCA Flags

Refer to [NVIDIA DOCA Arg Parser Programming Guide](#) for more information.

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
General flags	l	log-level	Sets the log level for the application: <ul style="list-style-type: none"> ▶ CRITICAL=20 ▶ ERROR=30 ▶ WARNING=40 ▶ INFO=50 ▶ DEBUG=60 	<code>"log-level": 60</code>
	v	version	Print program version information	N/A
	h	help	Print a help synopsis	N/A
Program flags	c	config	Path to JSON file with configurations	<code>"config": security_gateway_config.json</code>
	u	unsecured	PCIe address for the unsecured port	<code>"unsecured": "03:00.1"</code>
	s	secured	PCIe address for the secured port	<code>"secured": "03:00.0"</code>
	m	mode	IPsec mode. Possible values: tunnel,	<code>"mode": "tunnel"</code>

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
			transport, udp_transport.	
	un	unsecured-name	Interface name of the unsecured port	<code>"unsecured- name": "p1"</code>
	sn	secured-name	Interface name of the secured port	<code>"secured- name": "p0"</code>
	i	ipc	IPC socket file path for receiving IPsec rules during runtime	<code>"ipc": "/tmp/ rules_socket"</code>

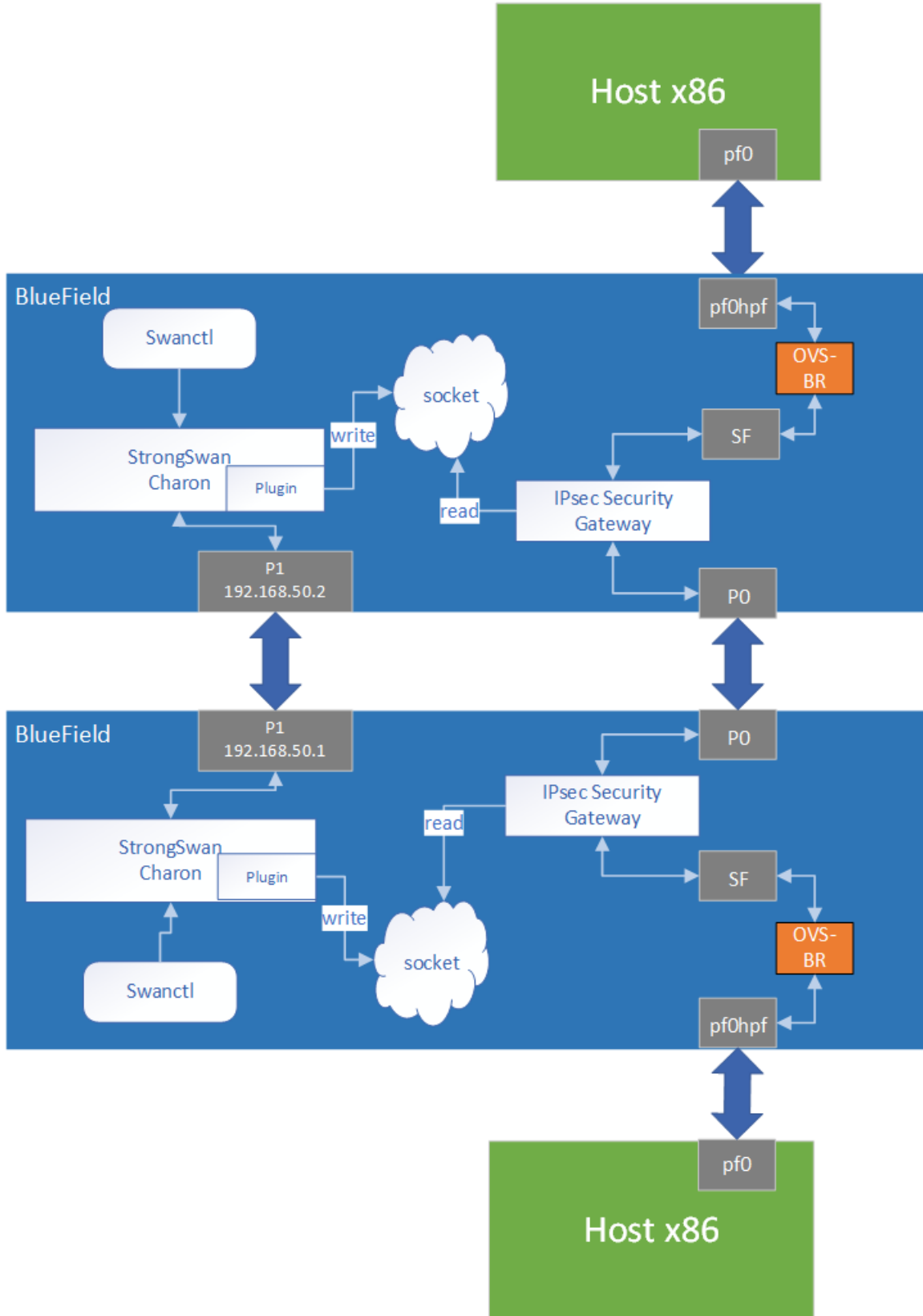
Chapter 8. Keying Daemon Integration (StrongSwan)

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, please refer to [strongSwan documentation](#).

8.1. 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 that is used for strongSwan IPsec configuration:

1. Run DOCA IPsec Security Gateway on both sides in 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:

a). On BF1:

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

b). On BF2:

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



Note: It is possible to configure multiple IP addresses to uplinks to run key exchanges with different policy attributes.

2. Verify the connection between two BlueField DPUs.

```
BF1> ping 192.168.50.2
```



Note: Make sure that the uplink is not in OVS bridges.

3. Configure the `swanctl.conf` files for each machine. They should be located under `/etc/swanctl/conf.d/`. Examples for adding `swanctl.conf` file:

▶ Transport mode:

▶ `swanctl.conf` example for BF1:

```
connections {
  BF1-BF2 {
    local_addr = 192.168.50.1
    remote_addr = 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
    mobike = no
    reauth_time = 0
    proposals = aes128-sha256-x25519
}

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

```

► swanctl.conf example for BF2:

```

connections {
    BF2-BF1 {
        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 = 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
        reauth_time = 0
        proposals = aes128-sha256-x25519
    }
}

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

```

► 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
}

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

```



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.

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:

ESP Proposal	Algorithm Type Including ICV Length	Key Size
aes128gcm8	ENCR_AES_GCM_ICV8	128
aes128gcm64	ENCR_AES_GCM_ICV8	128
aes128gcm12	ENCR_AES_GCM_ICV12	128
aes128gcm96	ENCR_AES_GCM_ICV12	128
aes128gcm16	ENCR_AES_GCM_ICV16	128
aes128gcm128	ENCR_AES_GCM_ICV16	128
aes128gcm	ENCR_AES_GCM_ICV16	128
aes256gcm8	ENCR_AES_GCM_ICV8	256
aes256gcm64	ENCR_AES_GCM_ICV8	256
aes256gcm12	ENCR_AES_GCM_ICV12	256
aes256gcm96	ENCR_AES_GCM_ICV12	256
aes256gcm16	ENCR_AES_GCM_ICV16	256


ESP Proposal	Algorithm Type Including ICV Length	Key Size
aes256gcm128	ENCR_AES_GCM_ICV16	256
aes256gcm	ENCR_AES_GCM_ICV16	256

8.2. 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`.


```
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`.


```
doca {
# Whether to load the plugin
load = yes

# Path to DOCA socket
socket_path = /tmp/rules_socket
}
```


 Note: You must provide the application with this path as well.

3. Restart the strongSwan server:

```
systemctl restart strongswan-starter.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.

 Note: It is recommended to remove any unused conf files under `/etc/swanctl/conf.d/`.

5. Load IPsec configuration:

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

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

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

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

8.3. 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>.
3. Git checkout BF-5.9.6 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-starter.service
```

Chapter 9. References

- ▶ `/opt/mellanox/doca/applications/ipsec_security_gw/src`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023 NVIDIA Corporation & affiliates. All rights reserved.