# NVIDIA DOCA East-West Overlay Encryption

## Reference Application

MLNX-15-060467 _v1.2 | January 2022

# Table of Contents

# Chapter 1. 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 characters 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, please refer to strongSwan documentation.

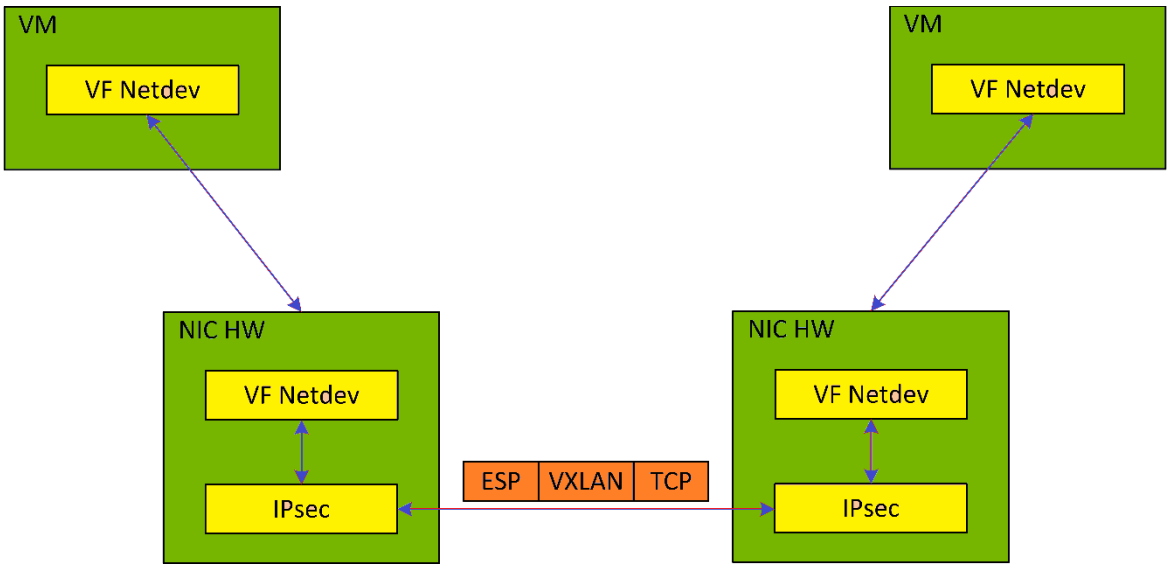
# Chapter 2. System Design

IPsec full offload offloads both IPsec crypto (encrypt/decrypt) and IPsec encapsulation to the hardware. IPsec full offload is configured on the Arm via the uplink netdev.

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.

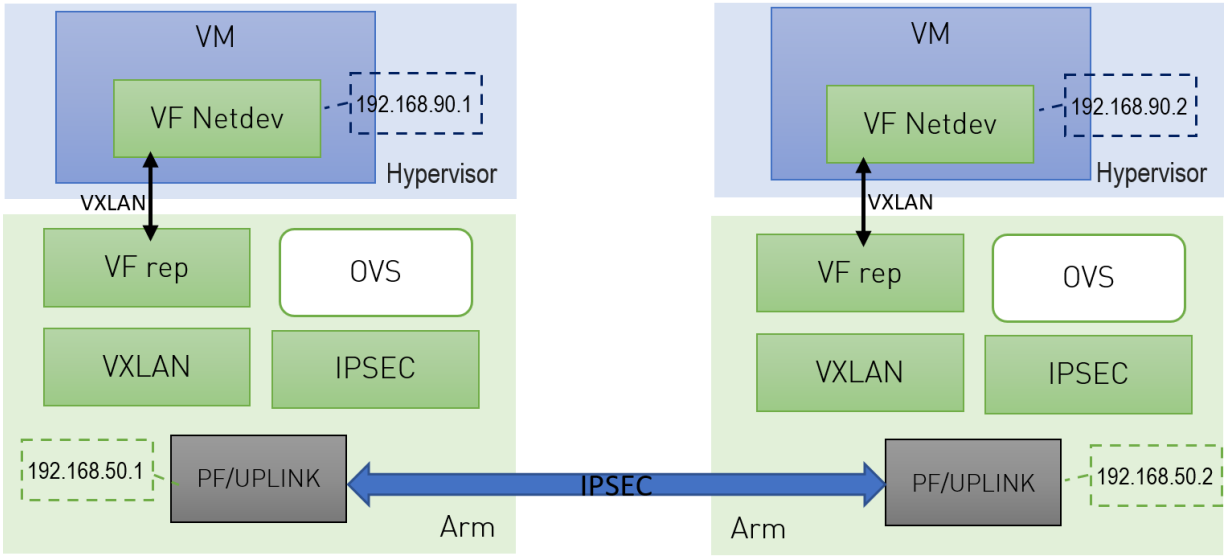
IPsec full offload configuration works with and is transparent to OVS offload. This means all packets from OVS offload are encrypted by IPsec rules.

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




**Note:** OVS offload and IPsec IPv6 do not work together.

# Chapter 3. Application Architecture




1. Configure strongSwan IPsec offload using swanctl.conf. configuration file.
2. Traffic is sent from the host through BlueField-2.
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.

# Chapter 4. Configuration Flow

The following subsections provide information on configuring IPsec full offload in general and on using IPsec with strongSwan specifically.

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

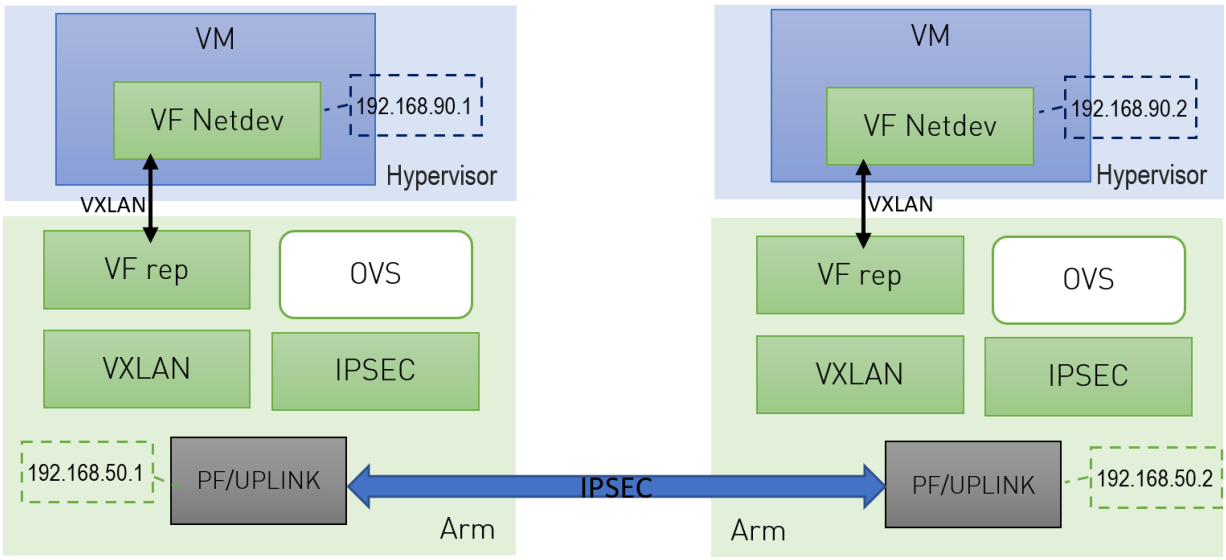
Explicitly enable IPsec full offload on the Arm cores before full offload rule is configured.

1. Delete the two SFs created on the BlueField device upon boot (one per port if the port is in switchdev mode). Instructions for deleting SFs may be found at Scalable Function Setup Guide.
2. Move the SR-IOV mode to legacy. Run on Arm:
   ```
   devlink dev eswitch set pci/0000:03:00.0 mode legacy
   ```
3. Set the IPsec to full offload. Run on Arm:
   ```
   echo full > /sys/class/net/p0/compat/devlink/ipsec_mode
   ```
4. Enable firmware steering. Run on Arm:
   ```
   echo dmfs > /sys/bus/pci/devices/0000\:03\:00.0/net/p0/compat/devlink/
   steering_mode
   ```
5. Enable the switchdev SR-IOV mode. Run on Arm:
   ```
   devlink dev eswitch set pci/0000:03:00.0 mode switchdev
   ```
6. To enable IPsec full offload on the second port, please perform steps 2-5 on pci/0000:03:00.1.

**Note:** Make sure the MTU of the Arm PF used by IPsec is at least 26 bytes larger than the Arm VXLAN-REP MTU.

**Note:** BlueField DPU supports configuring IPsec rules using strongSwan 5.9.0bf (yet to be upstreamed) which supports new fields in `swanctl.conf` file.

The following figure illustrates an example with two BlueField DPUs, Left and Right, operating with a secured VXLAN channel.




Support for strongSwan IPsec full HW offload requires using VXLAN together with IPSec as shown here.

1. Follow the procedure described above.
2. Configure VXLAN tunnel.

   a). Consider `p0` to be the local VXLAN tunnel interface.

   b). Build a VXLAN tunnel over OVS `arm-ovs`. Run:

   ```
   ovs-vsctl add-port arm-ovs vxlan11 -- set interface vxlan11 type=vxlan
   options:local_ip=192.168.50.1 options:remote_ip=192.168.50.2 options:key=100
   options:dst_port=4789
   ```

   c). Connect pf0hpf to the same arm-ovs.

   d). Run traffic over `pf0` on x86 (the one connected to `pf0hpf`) to the host the DPU connected.

   e). Configure the MTU of the PF used by VXLAN to at least 50 bytes larger than VXLAN-REP MTU.

3. Query OVS VXLAN `hw_offload` rules.

   a). To query OVS VXLAN `hw_offload` rules, run:

   ```
   ovs-appctl dpctl/dump-flows type=offloaded
   in_port(2),eth(src=ae:fd:f3:31:7e:7b,dst=a2:fb:09:85:84:48),eth_type(0x0800),
    packets:1, bytes:98, used:0.900s,
    actions:set(tunnel(tun_id=0x64,src=192.168.50.1,dst=192.168.50.2,tp_dst=4789,flags(key))),
   tunnel(tun_id=0x64,src=192.168.50.1,dst=192.168.50.2,tp_dst=4789,flags(+key)),in_port(3),eth
    packets:75, bytes:7350, used:0.900s, actions:2
   ```

   b). For the host PF, in order for VXLAN to work properly with the default 1500 MTU, disable host PF as the port owner from Arm. Run:

   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```

   The MTU of the end points (`pf0hpf` in the example above) of the VXLAN tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the VXLAN headers. For example, you can set the MTU of P0 to 2000.

i. Make sure the MTU of the PF used by VXLAN is at least 50 bytes larger than VXLAN-REP MTU.

ii. Make sure the MTU of the Arm PF used by IPsec is at least 26 bytes larger than the Arm VXLAN-REP MTU.

iii. Enable TC offloading. Run:

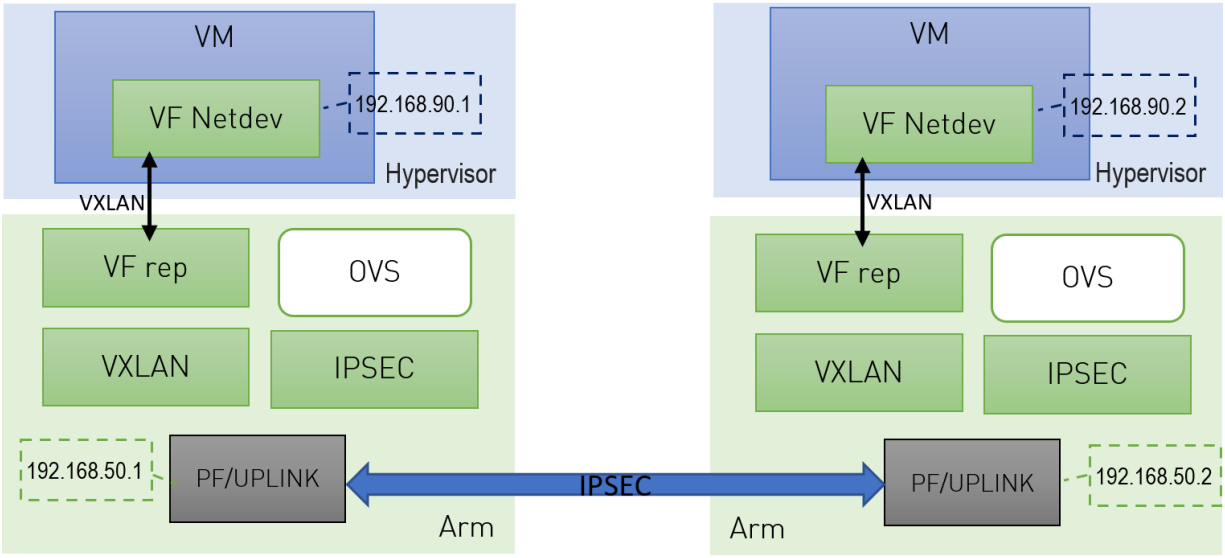
```
ethtool -K <PF> hw-tc-offload on
```

**Note:** Do not add the PF itself using `ovs-vsctl add-port` to the OVS.

# 4.1. Setting IPsec Full Offload Using strongSwan

strongSwan configures IPSec HW full 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) are used to identify the two nodes that communicate corresponding with the following figure:




In this example, 192.168.50.1 is used for the left PF uplink and 192.168.50.2 for the right PF uplink.

```
connections {
   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 {
            local_ts = 192.168.50.1/24 [udp/4789]
            remote_ts = 192.168.50.2/24 [udp/4789]
            esp_proposals = aes128gcm128-x25519
            mode = transport
            policies_fwd_out = yes
            hw_offload = full
         }
      }
      version = 2
      mobike = no
      reauth_time = 0
      proposals = aes128-sha256-x25519
   }
}

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

BFB installation will place two example swanctl.conf files for both Left and Right nodes (`BFL.swanctl.conf` and `BFR.swanctl.conf` respectively) in the strongSwan conf.d directory. Please move one of them manually to the other BlueField-2 machine and edit it according to your configuration.

Note that:

- "`hw_offload = full`" is responsible for configuring IPSec HW full offload
- Full offload support has been added to the existing `hw_offload` field and preserves backward compatibility.

  For your reference:

| Value | Description |
|---|---|
| no | Do not configure HW offload, fail if not supported |
| yes | Configure crypto HW offload if supported by the kernel, fail if not supportedExisting |
| auto | Configure crypto HW offload if supported by the kernel, do not failExisting |
| full | Configure full HW offload if supported by the kernel, fail if not supportedNew |

- 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 full HW offload requires setting up `devlink/ipsec_mode` to `full` beforehand.

- `[udp/4789]` is crucial for instructing strongSwan to IPSec only VXLAN communication.
- Full HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

| Fields | Limitation |
|---|---|
| reauth_time | Ignored if set |
| rekey_time | Do not use. Ignored if set. |
| rekey_bytes | Do not use. Not supported and will fail if it is set. |
| rekey_packets | Use for rekeying |

# Chapter 5. Running Application on BlueField

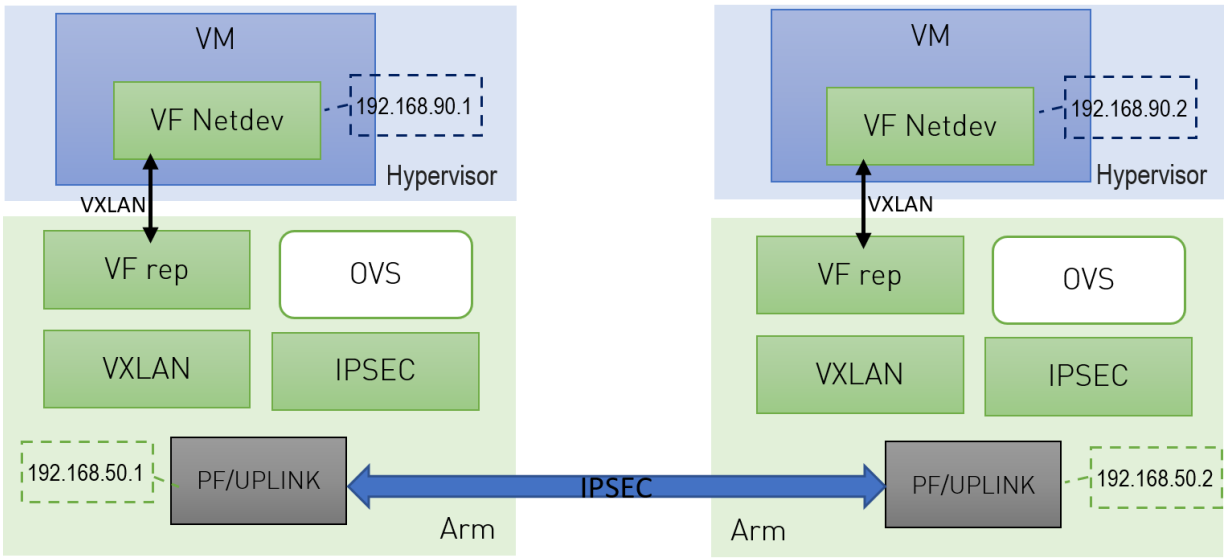
Please refer to the DOCA Installation Guide for details on how to install BlueField related software.

## 5.1. Running strongSwan Example

Notes:

- IPsec daemons are started by systemd strongswan-starter.service
- Use `systemctl [start | stop | restart]` to control IPsec daemons through strongswan-starter.service. For example, to restart, the command `systemctl restart strongswan-starter.service` will effectively do the same thing as `ipsec restart`. Do not use the `ipsec` script to restart/stop/start.
- If you are using the `ipsec` script, then, in order to restart or start the daemons, `openssl.cnf.orig` must be copied to `openssl.cnf` before performing `ipsec restart` or `ipsec start`. Then `openssl.cnf.mlnx` can be copied to `openssl.cnf` after restart or start. Failing to do so can result in errors since `openssl.cnf.mlnx` allows IPsec PK and RNG hardware offload via the OpenSSL plugin.
  - On Ubuntu/Debian/Yocto, `openssl.cnf*` can be found under `/etc/ssl/`
  - On CentOS, `openssl.cnf*` can be found under `/etc/pki/tls/`
- The strongSwan package installs `openssl.cnf` config files to enable hardware offload of PK and RNG operations via the OpenSSL plugin
- The OpenSSL dynamic engine is used to carry out the offload to hardware. OpenSSL dynamic engine ID is "pka".

Procedure:




1. Perform the following on Left and Right devices corresponding with the figure above:

```
# systemctl start strongswan-starter.service
# swanctl --load-all
```

   The following should appear:

```
Starting strongSwan 5.9.0bf IPsec [starter]...
no files found matching '/etc/ipsec.d/*.conf'
# deprecated keyword 'plutodebug' in config setup
# deprecated keyword 'virtual_private' in config setup
loaded ike secret 'ike-BF'
no authorities found, 0 unloaded
no pools found, 0 unloaded
loaded connection 'BFL-BFR'
successfully loaded 1 connections, 0 unloaded
```

2. Perform the actual connection on **one side only** (client, Left in this case).

```
# swanctl -i --child bf
```

   The following should appear:

```
[IKE] initiating IKE_SA BFL-BFR[1] to 192.168.50.2
[ENC] generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP)
 N(FRAG_SUP) N(HASH_ALG) N(REDIR_SUP) ]
[NET] sending packet: from 192.168.50.1[500] to 192.168.50.2[500] (240 bytes)
[NET] received packet: from 192.168.50.2[500] to 192.168.50.1[500] (273 bytes)
[ENC] parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ
 N(FRAG_SUP) N(HASH_ALG) N(CHDLESS_SUP) N(MULT_AUTH) ]
[CFG] selected proposal: IKE:AES_CBC_128/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/
CURVE_25519
[IKE] received 1 cert requests for an unknown ca
[IKE] authentication of 'host1' (myself) with pre-shared key
[IKE] establishing CHILD_SA bf{1}
[ENC] generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr AUTH N(USE_TRANSP)
 SA TSi TSr N(MULT_AUTH) N(EAP_ONLY) N(MSG_ID_SYN_SUP) ]
[NET] sending packet: from 192.168.50.1[500] to 192.168.50.2[500] (256 bytes)
[NET] received packet: from 192.168.50.2[500] to 192.168.50.1[500] (224 bytes)
[ENC] parsed IKE_AUTH response 1 [ IDr AUTH N(USE_TRANSP) SA TSi TSr
 N(AUTH_LFT) ]
[IKE] authentication of 'host2' with pre-shared key successful
[IKE] IKE_SA BFL-BFR[1] established between
 192.168.50.1[host1]...192.168.50.2[host2]
[IKE] scheduling reauthentication in 10027s
[IKE] maximum IKE_SA lifetime 11107s
```

```
[CFG] selected proposal: ESP:AES_GCM_16_128/NO_EXT_SEQ
[IKE] CHILD_SA bf{1} established with SPIs ce543905_i c60e98a2_o and TS
 192.168.50.1/32 === 192.168.50.2/32
initiate completed successfully
```

You may now send encrypted data over the HOST VF interface (192.168.70.[1|2]) configured for VXLAN.

# 5.2. Building strongSwan

**Note:** Perform the following only if you want to build your own BFB and would like to rebuild strongSwan.

1. strongSwan IPsec full version can be found here (tag: 5.9.0bf).
2. Install dependencies mentioned here. libgmp-dev is missing from that list, so make sure to install that as well.
3. Git clone https://github.com/Mellanox/strongswan.git.
4. Git checkout BF-5.9.0.
5. Run `autogen.sh` within the strongSwan repo.
6. Run the following:

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

Note:

- `--enable-systemd` enables the systemd service for strongSwan present inside the GitHub repo (see step 3) at `init/systemd-starter/strongswan-starter.service.in`. This service file is meant for Ubuntu, Debian and Yocto distributions. For CentOS, the contents of the above file must be replaced by the one present in `systemd-conf/strongswan-starter.service.in.centos` (inside the GitHub repo) before running the configure script above.
- 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.

# 5.3. Reverting IPsec Configuration

To destroy IPsec configuration, run the following commands:

```
sudo ip x s f                                          // Disable IPsec flows
sudo ip x p f                                          // Disable IPsec flows
sudo systemctl start NetworkManager
sudo ip addr flush dev <port_name>                     // <port_name> = uplinks = p<0|
1>
sudo rm -f /etc/openvswitch/conf.db
```

For Ubuntu/Debian:

```
sudo /etc/init.d/openvswitch-switch restart
```

For CentOS/RHEL:

```
systemctl restart openvswitch;
systemctl enable openvswitch;
```

## 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 NONINFRINGEMENT, 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



NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com