IPsec Offload

Functional Specification Document

OpenConfig Support for IPsec-Offload in P4CP/Stratum

Version	**Author/Owner**	**Reviewers**	Date
0.1	Sabeel Ansari		9/22/2022
0.2	Sabeel Ansari		9/30/2022
0.3	Sabeel Ansari		10/12/2022
0.4	Sabeel Ansari		10/21/2022
0.5	Sabeel Ansari		10/27/2022
0.6	Sabeel Ansari		12/15/2022
0.7	Sabeel Ansari		12/21/2022
0.8	Sabeel Ansari		03/03/2023
0.9	Sabeel Ansari		11/21/2023

Known Limitations

- The sgnmi_cli tool is not a hardened tool & is only used for quick internal testing as a gNMI Client. It is not meant to be exposed to external customers since the target gNMI client for customers is strongSwan.
- Workaround on sgnmi_cli yangpath:
    - Use the following yangpath to send gnmi requests from gnmi-cli tool: */ipsec-offload/sad/sad-enrtry[name=xxx]/config*
    - Note that this is a Stratum gnmi-cli limitation only. If using strongSwan, user can send the correct yangpath: */ipsec-offload/sad/sad-enrtry [offload-id=xxx][direction=yyy]/config*

1. OVERVIEW

The Inline IPsec Recipe is an application that enables strongSwan to use the Infrastructure Application Interface, specifically the P4Runtime and OpenConfig gRPCs provided by the Networking Recipe. OpenConfig is used to configure Security Associations (SA) into the infrastructure and P4 is used program how and where the IPsec is in-lined.

A strongSwan plug-in is used to convert security association (SA) information into OpenConfig RPC messages to the underlying infrastructure. The P4 program defines which modes are required in the pipeline, as well as where the IPsec is inserted into the broader virtual networking pipeline: Tunnel vs Transport, encrypted virtual ports, encrypted physical ports.

The IPsec control plane (IKE protocol) is offloaded to the infrastructure. The IKE protocol processing runs in software on the infrastructure cores and programs the data plane with offloaded IPsec flows. The data plane programming is separated into two parts, P4Runtime for programming the P4 pipeline and OpenConfig for programming the IPsec crypto configuration. The IPsec crypto configuration includes the IPsec Security Association (SA) table entries (i.e. crypto keys and algorithm selection) and is used for reporting events related to lifetime expiration of SA entries.

Scope:

Programming of the FXP, CXP and ICE pipeline blocks is outside the scope of this FSD. ICE team will write P4 programs to compile and map to appropriate blocks & P4 tables in the target to program the pipelines.

The communication over gRPC/gNMI channel which conforms to the OpenConfig model for configuring the *Fixed Functions* is the focus of this document

2. USE CASES

List different use cases

- Fetch Security Parameter Index (SPI): gNMI client-initiated call to gNMI server to fetch SPI, which returns an integer value for the configured index from the target.
- Add/delete security association (SA) database entry: gNMI client initiated call to gNMI server to add/delete SADB entry in the target
- Notifications related to SADB timer expiry originating from the target is propagated to the gNMI client

3. REQUIREMENTS

1. **Functional Requirements**

- IPsec offload YANG model must conform to OpenConfig standard.
- The YANG model and related gNMI messages must be target agnostic.

- The gNMI server must process incoming requests from gNMI clients to configure parameters related to IPsec SA, fetching SPI value etc. on the target.
- Notifications, as defined in the OpenConfig YANG model, originating in the target must propagate northbound to gNMI clients.
- Summary of OpenConfig gRPC calls for IPsec crypto configuration are:

Item	IPsec Message	GRPC channel	Data model
1	Fetch SPI	gNMI	TDI
2	Add/Delete Security Association	gNMI	TDI
3	Add Security Policy DB	P4RT	TDI
4	ICE auto config messages (notifications)	gNMI	TDI Attributes
5	Secure gNMI channel (SAD transactions)	gNMI (TLS 1.3 gRPC)	

- All crypto assets (keys) passing through the software stack must be stateless. The encryption keys are secret and Stratum/TDI must not keep them in memory or disk
- SAI interface to be supported in future release (not included as part of 23.01 release).

- Targets supported: MEV-TS.

1. **Non-Functional Requirements**

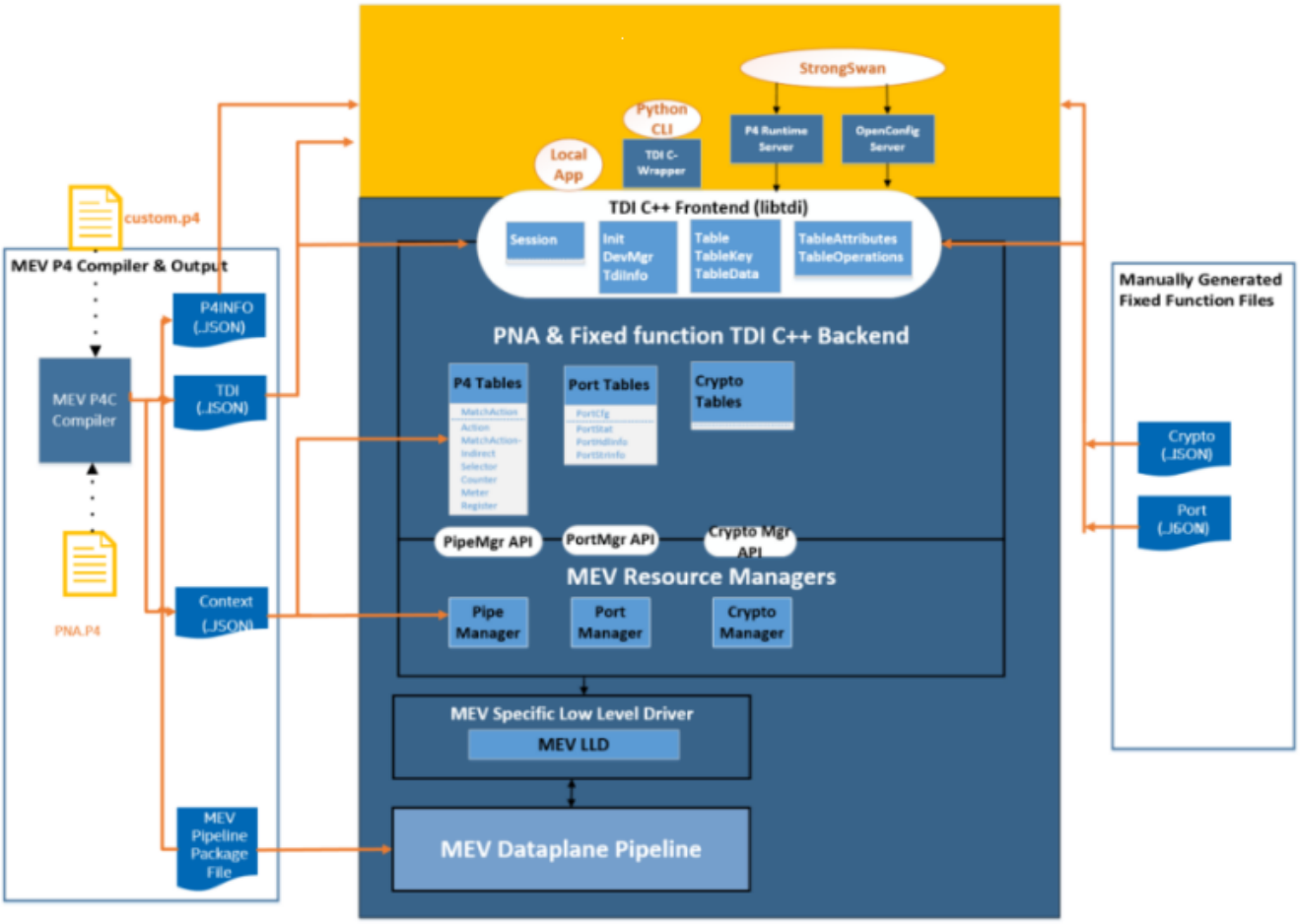
- Programming the FXP, CXP and ICE block pipelines will be via P4Runtime (not covered in this document).
- Target abstraction achieved via TDI API calls to underlying targets.

- TDI & TDI Attributes data models to be used by the gNMI server to communicate with the underlying target.
- Aligned with OPI Security API spec.
- The gNMI transactions should use TLS 1.3 secure channel of communication (#5 in table above).
- Feature should be functionally testable using any gNMI client.

4. DETAILED DESIGN

IPsec-offload YANG model

The YANG model for ipsec-offload is hosted here: https://github.com/intel-innersource/networking.ethernet.acceleration.vswitch.p4-sde.dataplane-p4/blob/master/openconfig/oc_models/public/release/models/ipsec/ipsec-offload.yang



Build integration

Design considerations:

Stratum will support addition of new leafs on the gnmi-server side in a couple of different ways:

1. Start adding newly supported leafs in YangParseTree. The Stratum infrastructure will pick up on the path and appropriate code will get invoked based on the code definition and flow.
   The required changes to add gNMI path in Stratum are documented here: https://github.com/stratum/stratum/tree/main/stratum/docs/gnmi#add-new-gnmi-path-to-stratum
2. ~~Alternatively, generate .proto & .pb.h & .pb.cc files (during build time preferably using yang2protobuf bazel directives), and integrate into build system on how to parse those gnmi messages. This may potentially need more code to have a openconfig converter to convert from incoming gnmi messages to internal datastructures. See this section on how OpenConfig & ChassisConfig messages are parsed in Stratum.~~

~~If proceeding with #2 above: Ideally, build integration should use #a below as it is a cleaner solution and provides a better workflow and code maintenance.~~

1. ~~P4OVS' bazel build system should download yang models for the ipsec-offload from a public repository, compile the yang using *ygot* to generate .proto files and then use *protoc* to generate the C++ (.pb.h and .pb.cc) files. These files can then be integrated and built during the linking phase to invoke the appropriate the code in Stratum where objects can be translated from Openconfig to backend native objects (this is the model used for Openconfig/Hercules module in Stratum).~~
2. ~~Independently generated .proto and C++ (.pb.h and .pb.cc) files are checked into the P4OVS repo that are statically linked during build time to integrate with rest of Stratum codebase (this is the model used in Stratum's hal/lib/common/common.proto & common.pb.h/cc).~~

~~Due to the following restrictions, design option #b is the feasible option for integration of ipsec-offload into Stratum code. Once following issues are resolved, built integration can move to option #a as described above.~~

- ~~The ipsec-offload yang model is currently part of the Intel innersource repo and is currently in a draft state (Cannot opensource it yet).~~
- ~~The ygot tool has limitations where the yang model has to be edited to remove all *Notifications* from the .yang file.~~

- ~~The ygot tool also has design issues where it cannot generate with *bits* yang type, so existing native models need to be edited to workaround this issue (https://github.com/openconfig/ygot/issues/725)~~

Security Considerations

The SADB entry config gNMI message contains the encryption key – which needs to be handled as a secret (no plaintext in-transit and at-rest).

The key is not stored anywhere in software. Once the key is passed to the TDI interface, the TDI implementation will send it directly to ICE HW. After the key is configured in the HW, it is not possible to retrieve the key. Any reads from the HW will zero the key returned in SA entry contents.

In addition, the northbound interface from P4CP for gRPC messages (both P4RT and gNMI) will be secured via SSL/TLS certificates. P4CP/Stratum will use a *secure-by-default* model and will only open secure ports. It will be up to the user to open insecure ports if needed.

Further details of securing gRPC communication and certificate management is detailed in this document: **gRPC Ports & Certificate Management**

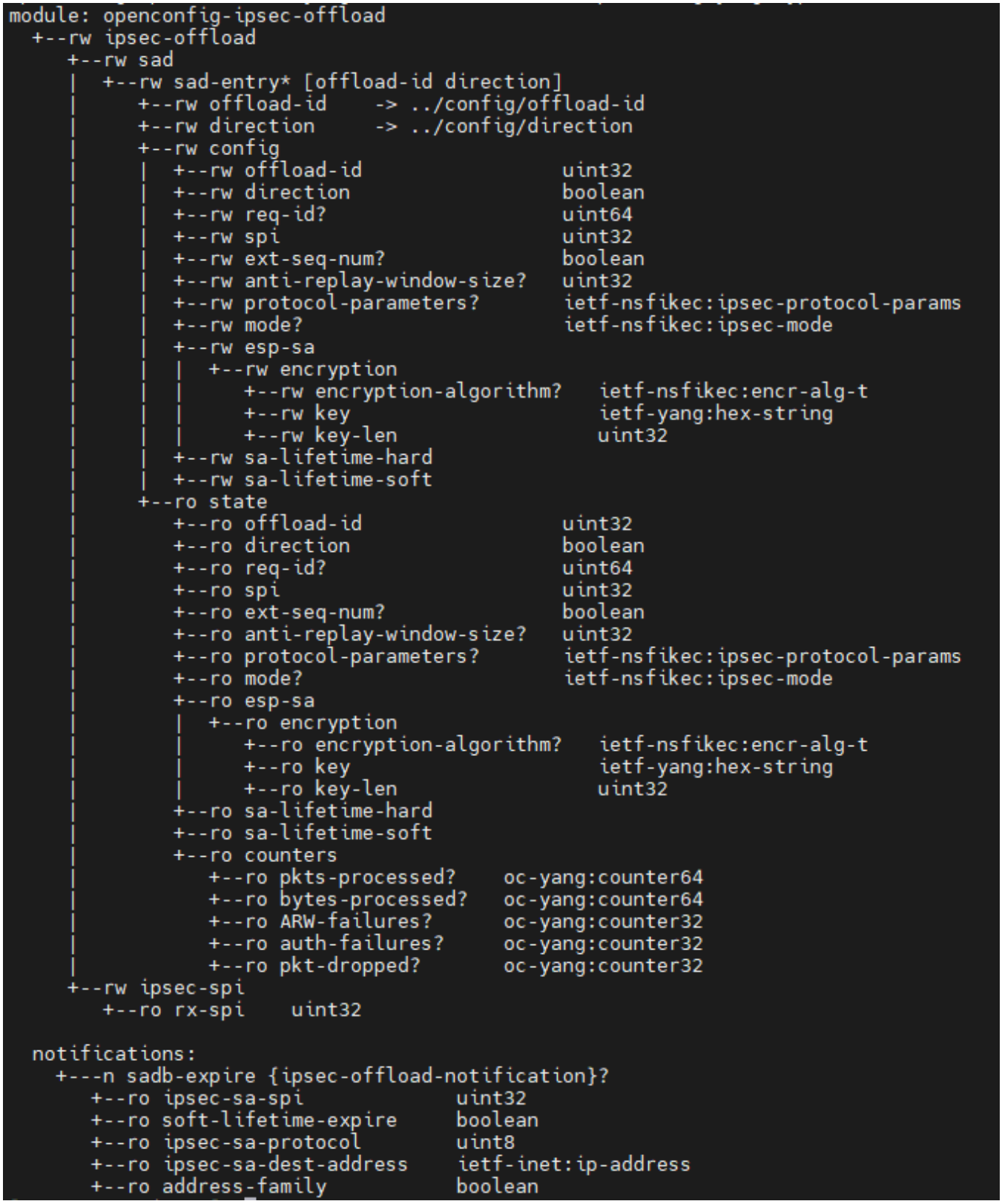
gNMI Client

ICE team intends to use strongSwan for operational needs. However, for component testing, the P4CP team will utilize the *gnmi-cli* to test the IPsec feature.

gNMI messages sent from client to server will be in protobuf format at the following yang paths:

- Fetch SPI: /ipsec-offload/ipsec-spi/rx-spi --> This will return a uint32 integer number from the server as a gRPC response.
- SA DB configuration: Everything under the /ipsec-offload/sad/sad-entry[offload-id=<uint>][direction=<bool>]/config container. This section will be sent as one single self-contained message (more info in next section).

The tree view of ipsec-offload is shown here.



gNMI Server

Stratum code will be appended to add new paths to the AddSubtree*() to support the /ipsec-offload YANG path. This will be invoked for any gNMI GET and SET messages from the gnmi-client.

In addition, the gNMI server will register for subscription callbacks. For notifications, the target-generated messages will be propagated to the northbound gNMI client.

Due to the hard requirement of not building any cache and storing ConfigSAD data in memory or disk (cannot store encryption keys in memory), the gNMI client will be required to send all contents under */ipsec-offload/sad/sad-entry[*]/config* as a single entity. This allows for the gNMI server to not wait for individual leaves of the YANG tree building the pieces before making a TDI API call.

Sample proto byte message from gNMI client sent to gNMI server:

```
offload_id: 555,
direction: false,
req_id: 873,
spi: 999,
ext_seq_num: false,
anti_replay_window_size: 64,
protocol_parameters: IPSEC_PROTOCOL_PARAMS_ESP,
mode: IPSEC_MODE_TUNNEL,
esp_payload {
  encryption {
    encryption_algorithm: 12,
    key: "1234567890abcdef",
    key_len: 16,
  }
},
sa_lifetime_hard {
  bytes: 5000000
},
sa_lifetime_soft {
  bytes: 3000000
}
```

Stratum Yang Tree Design

Existing design for Stratum yang tree nodes is to initialize the yang tree with known tree node values (such as ports, interfaces etc) from a ChassisConfig configuration that may come from a disk or other methods. The *YangParseTreePaths* then maintains this tree to be invoked for incoming gNMI messages. There is no support in Stratum currently to add a TreeNode dynamically during runtime.

Considering the following reasons that the incoming gNMI messages will contain keys (offload-id + direction) that are unknown, and there is no dynamic runtime addition of TreeNode and more importantly, the scale of these messages is very large (to the order of 32 million ConfigSA SET gNMI messages), we cannot maintain a tree in-memory.

The design chosen to handle this is to filter incoming gNMI Update messages at the GnmiPublisher to check if it is a IPsec ConfigSADB message and if yes, strip the keys from the ::gnmi::Path. This allows a single TreeNode leaf to act as destination for the messages which can process the request. Before returning the response from the gNMI server, the offload-id + direction keys are added back to the ::gnmi::Path in order to reflect a correct response path to the client.

To summarize:

```
During init YangParseTreePaths::AddSubtreeIPsec()
  gnmi::path registered for ConfigSADB is "/ipsec-offload/sad/sad-entry/config"


method GnmiPublisher::HandleUpdate()
  // gnmi::path is now /ipsec-offload/sad/sad-entry[offload-id=xxx][direction=yyy]/config
  if IsPathSupportedIPsec(path) { // this checks if the gnmi::path is /ipsec-offload/sad/sad-entry[key1][key2]
/config (ignores the keys)
        Strip the keys from path
        // gnmi::path is now /ipsec-offload/sad/sad-entry/config. The FindNodeOrNull() will find the only
TreeNode available to handle this message
  }
  normal processing of HandleUpdate() continues...


method SetUpIPsecSAConfig()
   // gnmi::path is now /ipsec-offload/sad/sad-entry/config
  Handle gNMI message, parse, process, make southbound calls etc
```

```
Update path to re-add the keys to the path
// gnmi::path is now /ipsec-offload/sad/sad-entry[offload-id=xxx][direction=yyy]/config
return response to gNMI client
```

This design also allows us to avoid major refactoring of Stratum code since it assumes single key with "name" as key field hardcoded in codebase.

Backend integration

The existing framework in Stratum's ConfigMonitoringService (which implements the GnmiPublisher and yang tree parsers) is extended to add support to the new ipsec-offload yang model.

A new class called *ipsec_manager* is added to handle IPsec-related interface from Stratum's YangParseTree (similar to Stratum's *chassis_manager*). The *ipsec_manager* object is owned by *TdiInterface* which is initialized during startpup of ConfigMonitoringService in Stratum.

In addition, a *FixedFunctionTableManager* is added for handling all southbound TDI API calls.

TDI integration

The southbound APIs in Stratum will use TDI (TDI tables and TDI Attributes). New table definitions being added in SDE to support tables for IPsec feature will be utilized.

We have 4 TDI tables:

Table Name	Table Index	Feature support	Table r/w	Note
ipsec-offload.ipsec-offload.sad.sad-entry.ipsec-sa-config	Offload-id + direction	Config SADB entry	Read-write	
ipsec-offload.ipsec-offload.sad.sad-entry.ipsec-sa-state	Offload-id + direction	State/Counters	Read-only	
ipsec-offload.ipsec-offload.ipsec-spi	None	FetchSPI	Read-only	Keyless table
ipsec-offload	None	Notifications	N/A	Keyless & dataless table

More info here:

https://wiki.ith.intel.com/pages/viewpage.action?pageId=2399161176#FixedFunctionTDIControlPlaneIntegrationArch-DesiredTDITablesfromYANGmodels

TDI APIs to use:

- Add IPsec SAD entry in TX direction

```
crypto.ipsec_sad.add_with(saidx=1, dir=1, spi=23243, key="hfgskdghksg",...)
```

- Remove IPsec SAD entry in TX direction

```
crypto.ipsec_sad.del_with(saidx=1, dir=1)
```

- Add IPsec SAD entry in RX direction

```
crypto.ipsec_sad.add_with(saidx=1, dir=0, spi=23243, key="hfgskdghksg",...)
```

- Remove IPsec SAD entry in RX direction

```
crypto.ipsec_sad.del_with(saidx=1, dir=0)
```

- Fetch SPI

```
  crypto.ipsec_sad.get_with(uint32 *spi)
```

Note: The TDI.json will be hand-written during the development of this feature (yang2tdi tool will not be ready until > 23.01)

5. EXCEPTION and ERROR HANDLING

- Explain exception and error cases specific to this feature if any.

6. DEBUGGING

List all the available mechanisms for debugging eg: logs, traces, stats/counters etc.

7. TEST CRITERIA

Step 1: (On IMC) Load crypto package

Note: New instructions post Feb-27-2023, with changes in SDE procedure. Old procedure details are available in Appendix A below.

On IMC:

- Reboot IMC
  - Hit 'N & Enter' during bootup (10s window) so that it doesn't "start init app and auxilliary script"
- In */etc/dpcp/package* folder, delete all .pkg files & copy the *crypto.pkg*. Rename *crypto.pkg* to *default_pkg.pkg*
- Edit /etc/dpcp/cp_init.cfg file with following:
  - disable_lpm=true
  - cpf_host = 4 (acc) or 0 (host)
  - pf_allowed_to_create_p2p = 12 (acc) or 8 (host)
  - sem_num_pages=25
- Start dpcp with following command

```
ipumgmtd -m 0xD95DF -p 0x1000 cli_mbx &
```

Step 2: (On MEV host) Set environment variables & devbind interface

```
# Set environment variables

$SDE_INSTALL/bin/vfio_bind.sh 8086:1453
modprobe vfio-pci
$SDE_INSTALL/bin/dpdk-devbind.py -b vfio-pci af:00.6

echo 512 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo 512 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
echo 512 >  /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
echo 512 >  /sys/devices/system/node/node1/hugepages/hugepages-1048576kB/nr_hugepages
```

The *'no_proxy'* environment variable needs to be set on the MEV NRB (for gnmi messages to work betwee client and server). Current settings on some of the MEV NRB have following settings

```
[root@mev08 ~]# env | grep proxy
no_proxy=intel.com,*.intel.com,localhost,127.0.0.1,10.0.0.0/8,192.168.0.0/16,172.16.0.0/12
https_proxy=http://proxy-chain.intel.com:912
http_proxy=http://proxy-chain.intel.com:911
[root@mev08 ~]#
```

Step 3: (On MEV host) Edit /usr/share/stratum/es2k/esk2_skip_p4.conf

The conf file needs to contain a ***fixed_functions*** block with a "name" value of "*crypto*". A sample is shown here:

```
        ...
    "eal-args": "--lcores=1-2 -a af:00.6,vport=[0-1] -- -i --rxq=1 --txq=1 --hairpinq=1 --hairpin-mode=0x0",
    "fixed_functions" : [
      {
       "name": "crypto",
       "tdi": "/root/fixed/tdi.json",
       "ctx": "/root/fixed/crypto-mgr-ctx.json"
      }
    ],
    "p4_programs": [
        ...
```

The P4CP code has been tested with the following tdi.json and context.json files:

TDI.json: https://github.com/intel-innersource/networking.ethernet.acceleration.vswitch.p4-sde.dataplane-p4/blob/master/openconfig/oc_models/public/release/models/ipsec/ipsec_sad_offload.json

Context.json: https://github.com/intel-innersource/networking.ethernet.acceleration.vswitch.p4-sde.dataplane-p4/blob/master/openconfig/oc_models/public/release/models/ipsec/ipsec_sad_offload_ctx.json

Step 4: (On MEV host) Run infrp4d

Step 5: (On IMC) Set Registers

Following registers need to be set via IMC if running OVS test on host system

```
# This is needed to pass traffic
devmem 0x202920C100  64  0x80

# Following registers need to be set with Queue ID. The ID number (353 here) is printed by SDE console and can
be seen on infrap4d if run in non-detached mode
# Look for following lines:
#    cp_add_cfgqs(): added cfgq to hw. queue id: 353
#    cp_add_cfgqs(): added cfgq to hw. queue id: 353
#
devmem 0x2024902038  32  353
devmem 0x2024A02038  32  353
```

Step 6: (On MEV host) Create P4 artifacts

Get the latest P4 program, P4 compiler for crypto from respective repos, compile to generate the p4info.txt and other artifacts.

To generate the crypto.pb.bin file:

```
touch tofino.bin
tdi_pipeline_builder --p4c_conf_file=crypto.conf --bf_pipeline_config_binary_file=crypto.pb.bin
```

Step 7: (On MEV host) Set P4 pipeline

Note: *p4rt-ctl* and *sgnmi_cli* binaries are found in *$IPDK_RECIPE/install/bin/* (if path is not added to env var)

The SDE layer only adds device and initializes TDI tables after a pipeline is set. In order to use the IPsec gnmi feature, users are expected to set a pipeline.

```
p4rt-ctl set-pipe br0 crypto.pb.bin crypto_tunnel_main.p4info.txt
```

Step 8: Set security policy (SPD) and match-action tables via P4RT commands

```
p4rt-ctl add-entry br0 crypto_tunnel_control.ipsec_tx_spd_table "hdrs.ipv4[meta.common.depth].dst_ip=192.0.0.2
/255.255.255.255,priority=1,action=crypto_tunnel_control.ipsec_protect"

p4rt-ctl add-entry br0 crypto_tunnel_control.ipsec_tx_sa_classification_table "meta.common.crypto_offload=1,
hdrs.ipv4[meta.common.depth].src_ip=192.0.0.1,hdrs.ipv4[meta.common.depth].dst_ip=192.0.0.2,hdrs.ipv4[meta.
common.depth].protocol=6,action=crypto_tunnel_control.ipsec_tx_transport(0x1)"

p4rt-ctl add-entry br0 MainControlDecrypt.ipsec_rx_sa_classification_table "hdrs.ipv4[meta.common.depth].
src_ip=192.0.0.2,hdrs.ipv4[meta.common.depth].dst_ip=192.0.0.1,hdrs.esp.spi=0x256,action=MainControlDecrypt.
ipsec_decrypt(1)"

p4rt-ctl add-entry br0 crypto_tunnel_control.ipsec_rx_post_decrypt_table "meta.common.crypto_status=0,meta.
common.crypto_offload=1,meta.common.crypto_tag=2,hdrs.ipv4[meta.common.depth].dst_ip=192.0.0.1/255.255.255.255,
priority=1,action=crypto_tunnel_control.ipsec_transport_post_decrypt"
```

Step 9: Send gNMI messages

Note that gnmi-cli (built as part of infrap4d/Stratum) does not support multiple keys in the yang path. In order to test, current implementation will process the incoming yang path with /ipsec-offload/sad/sad-entry["*name=<offload_id>"]config* and will delete both directions for that offload-id.

However, if using other gnmi clients, where the multiple key yangpath is supported, functionality is supported to delete only the specified offload-id and direction.

ConfigSA SET message

To test the IPsec ConfigSA db message:

1. A sample ConfigSA message has been posted here: https://gist.github.com/5abeel/a42d45b9fd2d51168fb04ec10653212b
2. If using Stratum's gnmi-cli, use the following command to send a message to infrap4d/Stratum GnmiServer

```
sgnmi_cli --proto_bytes="`cat /root/proto/ipsec-example-protobytes.txt`" set "/ipsec-offload/sad/sad-entry
[name=1]/config" --grpc_addr localhost:9339
<repeat with another file for with direction=true>
or
<other-gnmi-client> --proto_bytes="`cat /root/proto/ipsec-example-protobytes.txt`" set "/ipsec-offload/sad/sad-
entry[offload-id=1][direction=false]/config" --grpc_addr localhost:9339
<repeat for direction=true>
```

FetchSPI GET message

```
sgnmi_cli get "/ipsec-offload/ipsec-spi/rx-spi" --grpc_addr localhost:9339
```

SADB DELETE message

```
sgnmi_cli del "/ipsec-offload/sad/sad-entry[name=1]/config" --grpc_addr localhost:28000
or
<other-gnmi-client> del "/ipsec-offload/sad/sad-entry[offload-id=1][direction=false]/config" --grpc_addr
localhost:28000
```

Notification message

The following command opens a stream from gNMI client to listen to notification changes from target.

```
sgnmi_cli sub-onchange /ipsec-offload/sadb-expire
```

To test notification, generate traffic that will cross the set soft bytes limit via SADBConfig message

```
scapy
sendp(Ether()/IP(dst="192.0.0.2",src="192.0.0.1")/TCP()/Raw(load="0"*50),iface='eth1',count=10)
```

When the soft limit is crossed, a notification on the gnmi-cli stream will be generated.

For e.g.:

```
RESPONSE
update {
  timestamp: 1700161871310321278
  update {
    path {
      elem {
        name: "ipsec-offload"
      }
      elem {
        name: "sadb-expire"
      }
    }
    val {
      string_val: "ipsec-sa-spi: 3361230391, soft-lifetime-expire: 0, ipsec-sa-protocol: 50, ipsec-sa-dest-
address: 192.0.0.2, address-family: 1"
    }
  }
}
```

Once a limit is hit, to continue repeated testing, send a delete SADB message via gNMI and re-add SADB config.

SDLe compliance/guidelines

Care has been taken to avoid any logging of the encryption key and all temporary variables are cleared from memory. Secure code review has been performed and following section performs a scrub of the data structure from infrap4'd *FixedFunctionManager*.

https://github.com/ipdk-io/stratum-dev/blob/cb409bf5611fa761bb4e911e23c74c59e37f6522/stratum/hal/lib/tdi/tdi_fixed_function_manager.cc#L29

8. References

1. IPsec-offload yang model: https://github.com/intel-innersource/networking.ethernet.acceleration.vswitch.p4-sde.dataplane-p4/blob/master/openconfig/oc_models/ipdk/models/ipsec-offload.yang
2. ICE Software Overview: https://wiki.ith.intel.com/display/ITSMEVTSMonterey/Inline+Crypto+Engine+Software
3. Crypto Tunnel IPDK.io: https://wiki.ith.intel.com/display/IPDK/Crypto+Tunnel+IPDK.io
4. UML diagram of YangParseTree: https://wiki.ith.intel.com/display/ndvswitch/Stratum+-+YangParseTree+Class
5. Fixed Function TDI Control Plane Integration Architecture: Fixed Function TDI Control Plane Integration Arch
6. TDI Attributes: https://wiki.ith.intel.com/display/NDSWCloud/TDI+ATTRIBUTES
7. P4 reference file (note that it is for MEV target only):
    a. https://github.com/intel-innersource/networking.ethernet.acceleration.vswitch.p4-sde.dataplane-p4/tree/ea03bb2dfeb7c91c356d5c27f84b1fbcc3361ba0/mev_reference_p4_programs/ipsec
    b. https://github.com/intel-innersource/networking.ethernet.acceleration.vswitch.p4-sde.dataplane-p4/blob/master/mev_reference_p4_programs/fxp-cxp-features/fxp-cxp_ipsec/fxp-cxp_ipsec.p4
8. Unit Test IPsec Recipe: Unit Test IPsec recipe (CES or Validation team do not use this page for reference)
9. Running IPsec Recipe (offloaded strongSwan) on MEV host: Running ipsec recipe (offloaded strongSwan) on MEV host

9. APPENDIX

Appendix A: Load crypto pkg on IMC (old procedure for MEV 0.7 and earlier)

On IMC:

- Reboot IMC
    - Hit 'N & Enter' during bootup (10s window) so that it doesn't "start init app and auxilliary script"
- In /usr/bin/cplane folder, delete all .pkg files & copy the crypto.pkg that is being tested
- Edit cp_init.cfg file with following:
    - disable_lpm=true
    - cpf_host = 0
    - correct package name
    - sem_num_pages=25
- Start IMCCP with following commands

```
cat /etc/issue.net

# mount is already done by IMC
cp /mnt/acc/acc-os-* /lib/firmware/
unlink /etc/hwconf/active

# For MEV B0 with link partner 100G
ln -s -f /etc/hwconf/BID_61108 /etc/hwconf/active
rm -rf /etc/hwconf/active/autoloads/rdma_AL
sh /usr/bin/imc-scripts/vfio_bind.sh 8086:1453 NOIOMMU
sh /usr/bin/imc-scripts/vfio_bind.sh mev_imc_platform NOIOMMU
/usr/bin/mev_imc_init_app -m 0x119f &

sleep 25s

cd /usr/bin/cplane
./imccp 0000:00:01.6 0 cp_init.cfg &
```

- Confirm that crypto.pkg is loaded by checking /log/messages