NVIDIA Hopper Confidential Computing Attestation Verifier

Application Note

DU-10684-001_v04 | July 2023

Document History

DU-10684-001_v04

Version	Date	Description of Change
01	March 14, 2023	Initial release
02	July 20, 2023	HCC SPT release
03	July 24, 2023	HCC EA release

Table of Contents

Chapter 1. Introduction

The NVIDIA® Hopper™ Confidential Compute (HCC) feature provides a GPU Trusted Execution Environment (TEE) and enables a CPU TEE to establish trust with the GPU to accelerate compute workloads in a secure environment. As a part of establishing trust, the GPU proves its trustworthiness with an attestation report that is verified and compared by CPU TEE.

The attestation report is a collection of measurements that are generated from the GPU's current state and a configuration that directly (or indirectly) influences the execution environment. For the NVIDIA Hopper GPU operating in Confidential Compute mode (CC mode), NVIDIA signed and developed secure software generates and stores measurements and reports the attestation.

To facilitate the GPU attestation report verification, NVIDIA will supply the reference measurements as a Reference Integrity Measurements Manifest (RIM) bundle with every HCC driver release. The rest of this document covers the states that are measured for attestation, a high-level sequence to fetch and verify the attestation report, the format and contents of the reference measurements, and the NVIDIA reference GPU attestation verifier tool's API documentation.

Chapter 2. GPU Measurements

A subset of the states and configurations on the HCC GPU are measured by using the following criteria:

- Static states that are fused at manufacturing and that define the device's personality and identity.
- Configurations required to establish and maintain the GPU TEE.
- The runtime state of the GPU software portion of the GPU TEE (for example, the hash of the driver-loaded microcodes).
- States that define the runtime HCC configuration.

Static states are measured once, and the states that can be modified at runtime are measured during every GPU attestation report request from the CPU TEE. The measurement is a SHA384 digest computed from the representative values from multiple states. While some states might be grouped into a SHA384 digest, not all states will be grouped.

An attestation report carries multiple SHA384 digests. As a result, a grouping of states for measurement computation is based on the availability at the time of computation, the availability at the time of the reference manifest generation, the transformative nature of a state, the role in proving the trustworthiness, and the state's source.

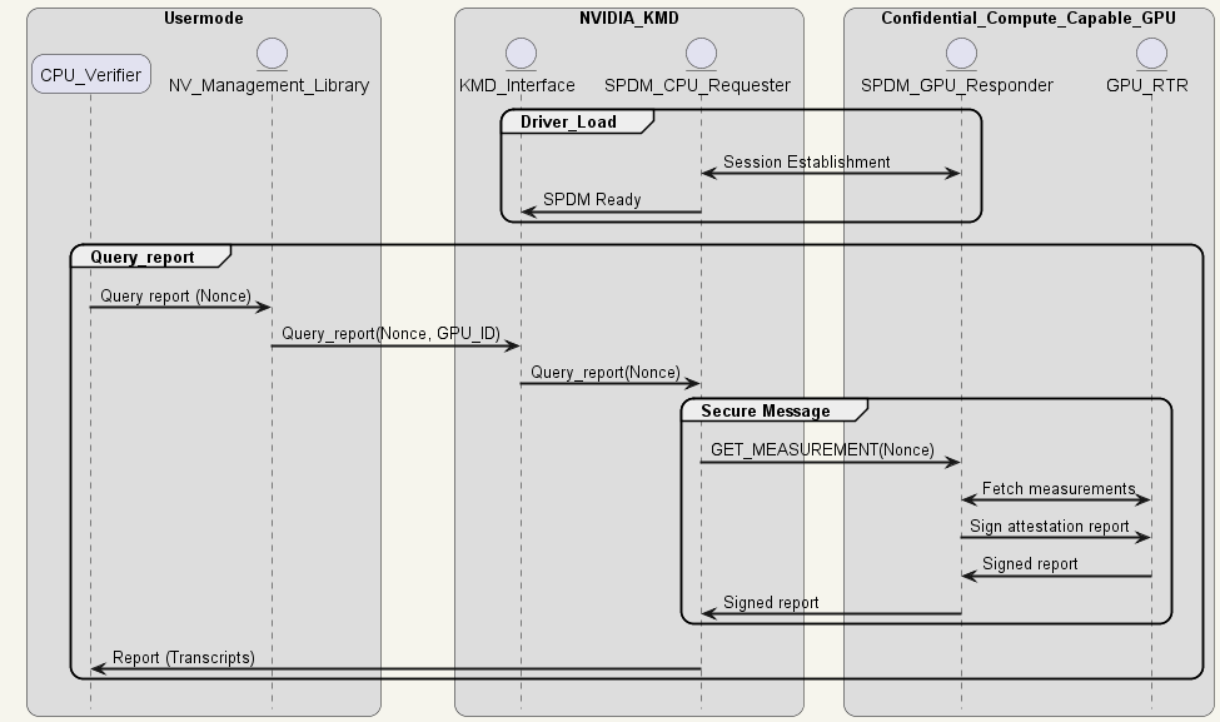
Chapter 3. Querying the GPU Attestation Report

To verify that the GPU is operating in a known good state, the attestation report allows users in a CPU TEE to get a snapshot of the GPU's current state and compare the state with the supplied reference measurements. Verifying the GPU states must be completed when you establish a secure session between the CPU TEE and the GPU TEE. Periodically verifying the GPU state at runtime allows the user to continuously monitor the GPU's operating state. The user in the CPU TEE is expected to query attestation reports by using user-mode APIs that are exposed by the NVIDIA software stack. The APIs and implementation to query the reports offer protection against replay and Man in the Middle (MITM) exploits.

When the GPU is CC mode-capable, and is booted in CC mode, the NVIDIA Kernel mode driver enables the HCC feature. After ensuring that the HCC feature is enabled, the NVIDIA Kernel driver establishes a secure session with the GPU. Establishing a session uses the sequences that are defined in the *Security Protocol and Data Model (SPDM)* specification and creates multiple encrypted channels when the session is successfully established.

The SPDM specification defines messages to query attestation reports in plain-text or encrypted formats using the secure session that was established earlier. For secure session establishment and attestation for HCC, NVIDIA follows the DMTF's SPDM 1.1 specification (DSP0274_1.1.0.pdf). User mode APIs that are exposed by NVIDIA to query attestation reports use SPDM as a backend to fetch and return reports to the users. A high-level sequence is illustrated in Figure 3-1.

Figure 3-1. A Query Attestation Report



In HCC SPT, the driver supplies a GPU attestation report signed using the secret that was derived from a unique device secret. Certificates required to validate the attestation report are supplied at runtime through the HCC driver APIs. Refer to “Device Certificate Chains” on page 7 for more information about certificate chains.

Chapter 4. Device Certificate Chains

The certificate chain for attestation is rooted to the NVIDIA root certificate followed by the Model and Provisioner certificates (see Figure 4-1).

Figure 4-1. The Certificate Chain

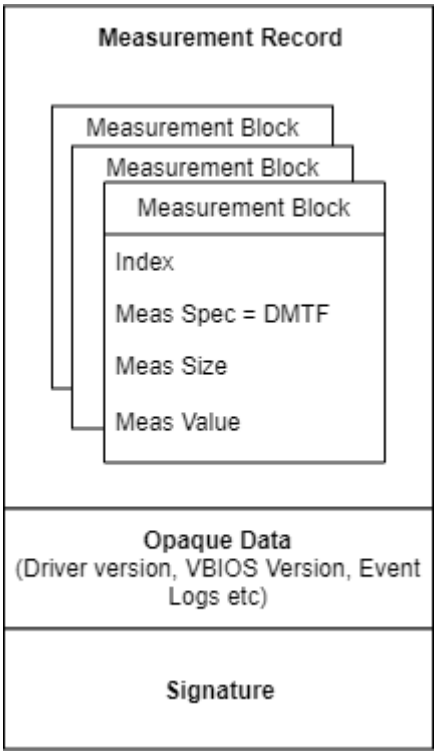
All certificates in this chain are in the x.509 v3 format. The NVIDIA Root cert, Model Cert and Provisioner Cert certificates are the same for all Hopper GPUs.. The Device Certificate is unique per device, and Attestation Certificate is unique per device and RTR software combination.

With the current flow, the first three levels of the HCC Device Certificate Chain are the same for all current devices that support CC. Therefore, these certificates are included as a part of the NVIDIA Kernel Driver.

Chapter 5. Format and Contents of Attestation Report

The SPDM specification defines the format of an attestation report but allows implementations to define the structure of measurement fields in a measurement block. For HCC, the structure of measurement fields in a block follows the DMTF-defined measurement field format. The attestation report from the HCC GPU can be represented in the following format.

Figure 5-1. Attestation Report Format



Chapter 6. Reference Measurements

As mentioned in “GPU Measurements” on page 4, states that are captured during an attestation report query include static hardware configurations, a hardware initialization state, the runtime hardware state, and various GPU software states.

Table 6-1. Measured State and its Scope

State	Scope
Static hardware Configurations	Configured at manufacturing process.
Firmware/VBIOS	Captures code execution and data before use.
Driver Ucodes	Measures different ucodes before they get loaded on engine.
HW Initialization state	Initialization done by VBIOS.
Runtime hardware state	Hardware state that is configured by the GPU driver software.

States that form the basis of an attestation report are impacted by the GPU hardware SKU type and the VBIOS/Driver software versions. For HCC, NVIDIA supplies one RIM bundle for the driver software and one for the VBIOS. The states captured might change based on future security needs. NVIDIA RIM bundles follow the RIM specification that is defined by the Trusted Computing Group’s *TCG Reference Integrity Manifest (RIM) Information Model* specification.

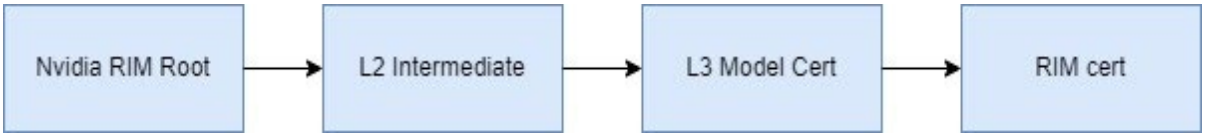
In the HCC SPT package, NVIDIA provides a RIM bundle in TCG RIM format, which is signed by the NVIDIA RIM signing cert. Refer to “RIM Cert Chain” on page 3 for more information about the RIM cert chain. By using the supplied RIM bundles, a user can successfully verify the GPU attestation report with the supplied verifier tool. The reference measurements in the RIM bundle are directly compared to the static and runtime measurements that were retrieved from the attestation report.

For the HCC SPT package, VBIOS RIM is part of the Verifier package and Driver RIM is bundled with driver package.

6.1 RIM Cert Chain

RIM certificate chain is rooted to the NVIDIA RIM signing root certificate followed by the Intermediate cert, Model cert and RIM cert (see Figure 4-1). The certificates in this chain are in the x.509 v3 format.

Figure 6-1. RIM Cert chain



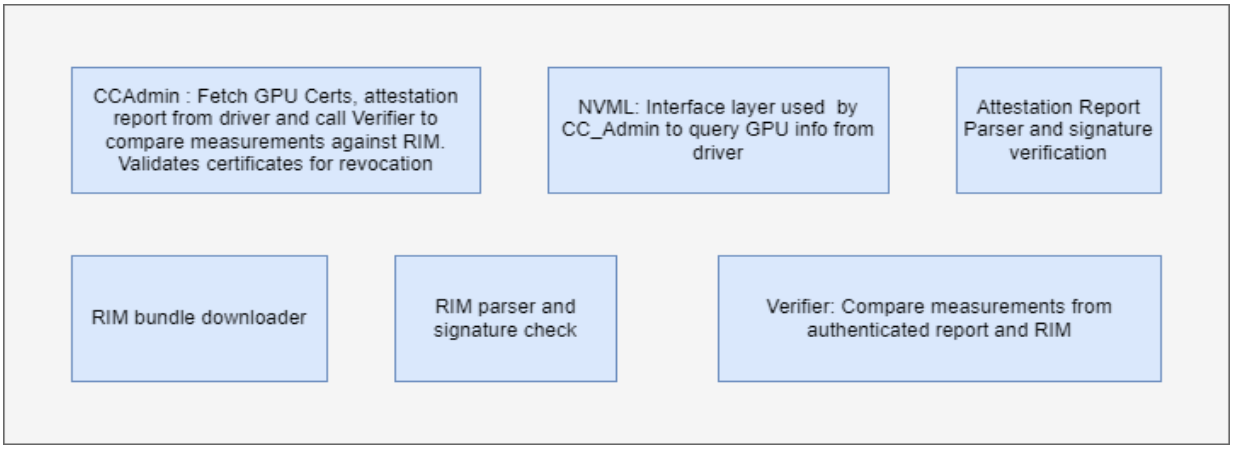
NVIDIA RIM Root, L2 Intermediate, L3 Model Cert are common for all Hopper GPUs. RIM cert is a leaf cert and unique per RIM. The Verifier validates the revocation status of the certs from NVIDIA OCSP server.

Chapter 7. Reference GPU Attestation Verifier Tool

7.1 Usage Model

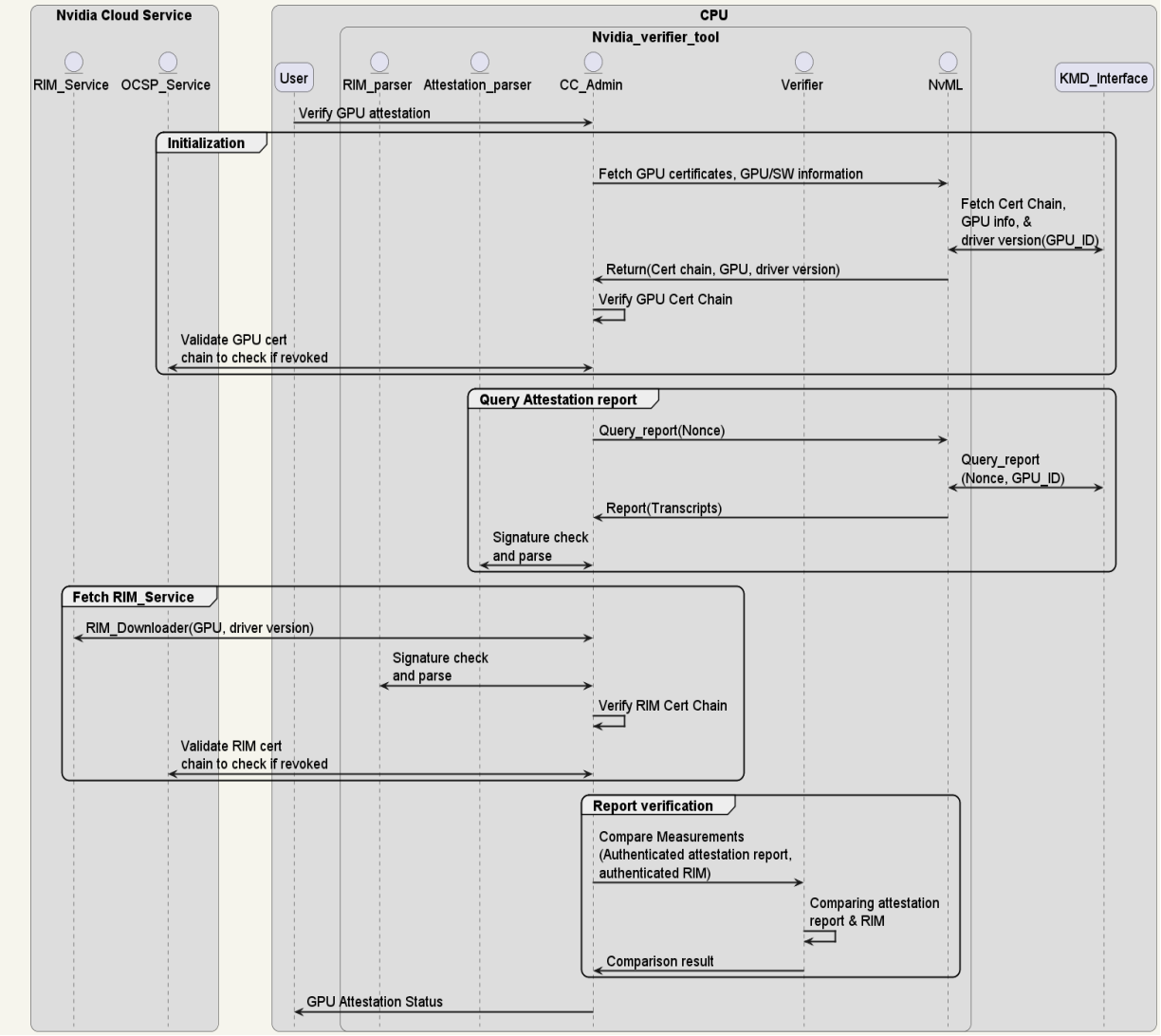
NVIDIA has developed a Python-based tool to verify attestation reports at boot and runtime. The tool includes the modules in Figure 7-1 and implements an end-to-end GPU attestation verification sequence.

Figure 7-1. Modules in the Verifier Tool



When run with an NVIDIA driver and a GPU that supports HCC, this tool uses the sequence in Figure 7-2 to verify GPU attestation report in a GPU.

Figure 7-2. Verifier Tool Sequence for HCC Drop 1 and Later



In Figure 7-2, the CC_Admin is provided as the reference implementation of a CPU-based CC Admin tool that aggregates and verifies the attestation report from various devices. The tool orchestrates the sequence of the GPU attestation verification by fetching the required data and passing it as the input to the GPU verifier. Customers can have their own implementation for an admin tool and continue to integrate other NVIDIA Verifier reference implementation modules as required.

The HCC SPT release has the following limitation that will be addressed in a future release:

- The RIM_Downloader uses the embedded RIM bundle instead of downloading from a remote host.

7.2 API Documentation

The verifier tool is implemented in Python and uses standard libraries to handle most of the crypto and XML parsing. The source code has been organized into the `cc_admin.py`, `nvmlHandler.py`, `attestation.py`, `rim.py`, and `verifier.py` files. Here is the list of interfaces in each of the previously mentioned modules.

7.2.1 CC_Admin

This module implements the following APIs to successfully complete the sequence outlined in Figure 7-2.

Table 7-1. CC_Admin APIs

Method	Description
`main()` **Input Parameters** • N/A **Returns** N/A	Main function of the cc_admin that directs the flow GPU attestation verification.
`retry(error, is_user_mode)` **Input Parameters** • error (exceptions.Error): The exception that has occurred. • is_user_mode (bool): If the cc_admin tool is being used in user_mode then it does not changes the gpu ready state. **Returns** N/A	This function is used to try the GPU attestation again when some types of exceptions occur.
`verify_certificate_chain(cert_chain, settings, mode)` **Input Parameters** • cert_chain(list): the certificate chain of the GPU as a list with the root certificate at the end of the list. • settings (config.HopperSettings): the object containing the various config info. • mode (<enum 'CERT CHAIN VERIFICATION MODE'>): Used to determine if the certificate chain verification is for the GPU attestation certificate chain or RIM certificate chain or the ocsp response certificate chain. **Returns**	Performs the certificate chain verification.

Method	Description
Returns true if the certificate chain verification is successful, and otherwise, False.	
`get_driver_RIM(driver_version, settings)` **Input Parameters** • driver_version (str): GPU driver version for which the driver RIM needs to be fetched. • settings (config.HopperSettings): the object containing the various config information. **Returns** Returns the path to the required Driver RIM.	Fetches the RIM from disk that corresponds to the driver version which is fetched from GPU information.
`get_vbios_rim_path(settings, attestation_report)` **Input Parameters** • settings (config.HopperSettings): the object containing the various config info. • attestation_report (SpdmMeasurementResponseMessage): the object representing the attestation report. **Returns** Returns the path to the required VBIOS RIM.	Static method to determine the path of the appropriate VBIOS RIM file corresponding to the VBIOS version fetched in the GPU information.
`verify_attestation_report(attestation_report_obj, gpu_leaf_certificate, nonce, driver_version, vbios_version, settings)` **Input Parameters** • attestation_report_obj (SpdmMeasurementResponseMessage): the object representing the attestation report. • gpu_leaf_certificate (OpenSSL.crypto.X509): the gpu leaf attestation certificate. • nonce (bytes): the nonce generated by the cc_admin. • driver_version (str): the driver version fetched from the GPU.	Completes the following tasks: • Nonce comparison between the generated nonce and the nonce in the GET MEASUREMENT request message in the attestation report. • The driver and VBIOS versions between the version fetched from the GPU information and the version in the attestation report are compared. Signature verification of the attestation report against the attestation key in the GPU leaf certificate

Method	Description
• vbios_version (str): the vbios version fetched from the GPU. • settings (config.HopperSettings): the object containing the various config info. **Returns** Returns True if authentication of the attestation report is successful, and otherwise, False.	
`generate_nonce(size)` **Input Parameters** • size(int): the length of random bytes in number of bytes **Returns** random bytes of the given length	Generates cryptographically strong random bytes as nonce to be sent to the SPDM requester using the `nvml api` for the attestation report.
`ocsp_certificate_chain_validation (cert_chain, settings, mode)` **Input Parameters** • cert_chain (list): the list of the input certificates of the certificate chain • settings (config.HopperSettings): the object containing the various config info. • mode (<enum 'CERT CHAIN VERIFICATION MODE'>): Used to determine if the certificate chain verification is for the GPU attestation certificate chain or RIM certificate chain or the OCSP response certificate chain. **Returns** [Bool]: True if the OCSP status of all the appropriate certificates in the certificate chain, otherwise False.	A static method to perform the OCSP status check of the input certificate chain.
`send_ocsp_request(data)` **Input Parameters** • data (bytes): the raw ocsp request message. **Returns** [cryptography.hazmat. backends.openssl.ocsp._OCSPResponse]: the ocsp response message object.	A static method to prepare http request and send it to the OCSP server and returns OCSP response data.
`verify_ocsp_signature(ocsp_response)`	A static method to perform the signature verification of the OCSP response message.

Method	Description
Input Parameters • ocsp_response (cryptography.hazmat.backends.openssl.ocsp._OCSPResponse): the input ocsp response message object. **Returns** [Bool]: returns True if the signature verification is successful, otherwise returns False	

7.2.2 NvmlHandler

This is the interface layer to the NVIDIA KMD, or to sample files, depending on available support.

Table 7-2. NvmlHandler APIs

Method	Description
`get_attestation_report(self)` **Input Parameters** • self(NvmlHandler class object reference) **Returns** [bytes]: The attestation report data of the GPU.	Fetches the attestation report of the GPU.
`get_attestation_cert_chain(self)` **Input Parameters** • self(NvmlHandler class object reference) **Returns** [list]: The list of x509 certificates in the certificate chain.	Fetches the GPU attestation certificate chain.
`get_gpu_architecture(self)` **Input Parameters** • self(NvmlHandler class object reference) **Returns** [str]: The GPU architecture.	Fetches the current GPU architecture.

Method	Description
`get_vbios_version(self)` **Input Parameters** • self(NvmlHandler class object reference) **Returns** [str]: the vbios version	Fetches the VbiosVersion field of the NvmlHandler class object.
`get_driver_version(self)` **Input Parameters** • self(NvmlHandler class object reference) **Returns** [str]: The driver version	Fetches the DriverVersion field of the NvmlHandler class object.
`fetch_attestation_report(self, index, nonce)` **Input Parameters** • self(NvmlHandler class object reference) • index (int): index of the GPU. • nonce (bytes): then nonce. **Returns** [bytes]: The raw attestation report data	Fetches the attestation report of the GPU.
`set_gpu_ready_state(state):` **Input Parameters** • state (bool): True to set it as ready and False for not ready. **Returns** N/A	Static method to set GPU state as ready if the input is True otherwise set as not ready to accept workload.
`get_gpu_ready_state ( )` **Input Parameters** • N/A **Returns** [int]: Returns 0 for the not ready state and 1 for the ready state.	Static method to check the GPU state.
`is_cc_dev_mode( )` **Input Parameters** • N/A	Static method to check whether the driver is in the CC DEV mode.

Method	Description
Returns [bool]: Returns True if the driver is in CC DEV mode, and otherwise, False.	
`init_nvml()` **Input Parameters** • N/A **Returns** N/A	Static method to initialize the nvml library.
`close_nvml()` **Input Parameters** • N/A **Returns** N/A	Shuts down the nvml.
`get_number_of_gpus(cls)` **Input Parameters** • cls: NvmlHandler class reference **Returns** [int]: The number of available GPUs.	Provides the number of the available GPUs.

Table 7-3. NvmlHandler Exceptions

Exception	Description
AtestationReportFetchError(PynvmlError)	It is raised when there is a failure in fetching the Attestation report.
CertChainFetchError(PynvmlError)	It is raised when there is a failure in fetching the GPU attestation certificate chain.
CertExtractionError(PynvmlError)	It is raised when there is an issue when extracting certificates from the certificate chain.
UnknownGpuArchitectureError(PynvmlError)	It is raised when the GPU architecture is not correct.
UnsupportedGpuArchitectureError(PynvmlError)	It is raised when the GPU architecture is not supported.
NoGpuFoundError(PynvmlError)	It is raised when the number of available GPU is zero.
TimeoutError(PynvmlError)	It is raised when the pynvml API call exceeds the threshold limit.

7.2.3 Attestation

The Attestation module consists of the `AttestationReport` class, which encapsulates `SpdmMeasurementRequestMessage` and `SpdmMeasurementResponseMessage` classes.

These APIs handle attestation report authentication and parsing.

Table 7-4. AttestationReport API

Method	Description
`extract_response_message(self, attestation_report_data)` **Input Parameters** • self (AttestationReport class object reference) • attestation_report_data (bytes): the attestation report coming from gpu via the nvml api. **Returns** [bytes]: The extracted SPDM MEASUREMENT response message.	Extracts the SPDM MEASUREMENT response message from the attestation report.
`extract_request_message(self, attestation_report_data)` **Input Parameters** • self (AttestationReport class object reference). • attestation_report_data (bytes): the attestation report coming gpu via the nvml api. **Returns** [bytes]: The extracted SPDM GET_MEASUREMENT response message.	Extracts the SPDM GET_MEASUREMENT request message from the attestation report.
`concatenate(request_data, response_data, length_of_signature)` **Input Parameters** • request (bytes): the SPDM GET_MEASUREMENTS request message • response (bytes): the SPDM successful MEASUREMENT response message • signature_length (int): Returns the size of the signature in number of bytes. **Returns** [bytes]: The concatenated request followed by a response message except the signature of the response message.	Concatenate GET_MEASUREMENT request and response message. Signature field in GET_MEASUREMENT response message is not filed in this API.

Method	Description
`get_measurements(self)` **Input Parameters** • self (AttestationReport class object reference). **Returns** Returns the list of GPU runtime measurements.	Fetches the runtime measurements from the attestation report.
`get_request_message(self)` **Input Parameters** • self (AttestationReport class object reference). **Returns** [SpdmMeasurementRequestMessage]: The object that represents the SPDM GET MEASUREMENT request message.	Fetches the SPDM GET MEASUREMENT request message, which is represented as an object of the SpdmMeasurementRequestMessage class.
`get_response_message (self)` **Input Parameters** • self (AttestationReport class object reference). **Returns** [SpdmMeasurementResponseMessage]: The object that represents the SPDM GET MEASUREMENT response message.	Fetches the SPDM GET MEASUREMENT response message represented as an object of the SpdmMeasurementResponseMessage class.
`verify_signature( self, certificate, signature_length, hashfunc)` **Input Parameters** • self (AttestationReport class object reference). • certificate (OpenSSL.crypto.X509): The GPU attestation leaf certificate • signature_length (int): the length of the signature field of the attestation report. • hashfunc (_hashlib.HASH): The hashlib hash function. **Returns** bool: Returns True if the signature verification is successful, and otherwise, False.	Performs the signature verification of the attestation report.
`print_obj(self, logger)` **Input Parameters** • self (AttestationReport class object reference).	Prints the fields in the request and response message in the attestation report object.

Method	Description
• logger (logging.Logger): the logger object which prints the output according to its set level. **Returns** N/A	

Table 7-5. Attestation Exceptions

Exception	Description
SignatureVerificationError(AttestationReportError)	It is raised when the signature verification of attestation report fails.
NoMeasurementsError(AttestationReportError)	It is raised when there are no / blank measurement blocks.
ParsingError(AttestationReportError)	It is raised when there are issues during parsing of the attestation report data.
NoMeasurementBlockError(AttestationReportError)	It is raised when there are no measurement blocks.
MeasurementSpecificationError(AttestationReportError)	It is raised when a measurement block does not follow DMTF specification.
NoCertificateError(AttestationReportError)	It is raised when there are no certificates in the GPU attestation certificate chain.
IncorrectNumberOfCertificatesError(AttestationReportError)	It is raised when there are unexpected number of certificates in the GPU attestation certificate chain.
CertChainVerificationFailureError(AttestationReportError)	It is raised when there is a GPU attestation certificate chain verification failure.
AttestationReportVerificationError(AttestationReportError)	It is raised when there is an attestation report signature verification failure.
NonceMismatchError(AttestationReportError)	It is raised when the nonce in the SPDM GET MEASUREMENT request message does not match the generated nonce.
DriverVersionMismatchError(AttestationReportError)	It is raised when the driver version in the attestation report does not match with the driver version that was fetched from the driver.
VBIOSVersionMismatchError(AttestationReportError)	It is raised when the VBIOS version in the attestation report does not match the VBIOS version that was fetched from the driver.

Table 7-6. SpdmMeasurementRequestMessage API

Method	Description
`get_spdm_version(self)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference)	Fetches the SPDM version field from the object of the SPDM measurement request message.

Method	Description
Returns [bytes]: The SPDM version.	

`set_spdm_version(self, value)`
Input Parameters
• self (SpdmMeasurementRequestMessage class object reference)
• value (bytes): Returns the SPDM version.
Returns
`N/A`
	Sets the SPDM version field of the object that represnets the SPDM GET_MEASUREMENT request.

`get_request_response_code(self)` **Input Parameters** • N/A **Returns** [bytes]: The RequestResponseCode.	Fetches the RequestResponseCode field of the object representing the SPDM GET_MEASUREMENT request.

`set_request_response_code(self, value)`
Input Parameters
• self (SpdmMeasurementRequestMessage class object reference)
• value (bytes): the RequestResponse value.
Returns
`N/A`
	Sets the RequestResponseCode field of the object that represents the SPDM GET_MEASUREMENT request.

`get_param1(self)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) **Returns** `[bytes]`: The Param1 value.	Fetches the Param1 field of the object that represents the SPDM GET_MEASUREMENT request.
`set_param1(self, value)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) • value (bytes): the Param1 value. **Returns** N/A	Sets the Param1 field of the object that represents the SPDM GET_MEASUREMENT request.
`get_param2(self)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) **Returns** [bytes]: The Param2 value	Fetches the Param2 field of the object that represents the SPDM GET_MEASUREMENT request.

Method	Description
`set_param2(self, value)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) • value (bytes): the Param2 value. **Returns** N/A	Sets the Param2 field of the object that represents the SPDM GET_MEASUREMENT request.
`get_nonce(self)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) **Returns** [bytes]: The nonce value.	Fetches the Nonce field of the object that represents the SPDM GET_MEASUREMENT request.
`set_nonce(self, value)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) • value (bytes): the nonce value. **Returns** N/A	Sets the Nonce field value of the object that represents the SPDM GET_MEASUREMENT request.
`get_slot_id_param(self)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) **Returns** [bytes]: The SlotIDParam value.	Fetches the SlotIDParam field value of the object that represents the SPDM GET_MEASUREMENT request.
`parse(self, request_data)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) • request_data (bytes): the raw message data. **Returns** N/A	Parses the raw SPDM GET_MEASUREMENT request message.
`print_obj(self, logger)` **Input Parameters** • self (SpdmMeasurementRequestMessage class object reference) • logger (logging.Logger): the logger object. **Returns**	Prints the field values of the object that represents the SPDM GET_MEASUREMENT request.

Method	Description
N/A	

Table 7-7. SpdmMeasurementResponseMessage API

Method	Description
`get_spdm_version(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) **Returns** [bytes]: The SPDM version.	Fetches the SPDM version field of the SPDM measurement response message object.
`set_spdm_version(self, value)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • value (bytes): the SPDM version. • Returns N/A	Sets the SPDM version field of the object that represents the SPDM GET_MEASUREMENT response.
`get_request_response_code(self)` **Input Parameters** • N/A **Returns** [bytes]: The RequestResponseCode.	Fetches the RequestResponseCode field of the object that represents the SPDM GET_MEASUREMENT response.
`set_request_response_code(self, value)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • value (bytes): the RequestResponseCode value. **Returns** N/A	Sets the RequestResponseCode field of the object that represents the SPDM GET_MEASUREMENT response.
`get_param1(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) **Returns**	Fetches the Param1 field of the object that represents the SPDM GET_MEASUREMENT response.

Method	Description
[bytes]: The Param1 value.	
`set_param1(self, value)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • value (bytes): the Param1 value. **Returns** N/A	Sets the Param1 field of the object that represents the SPDM GET_MEASUREMENT response.
`get_param2(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) **Returns** [bytes]: The Param2 value.	Fetches the Param2 field of the object that represents the SPDM GET_MEASUREMENT response.
`set_param2(self, value)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • value (bytes): the Param2 value. **Returns** N/A	Sets the Param2 field of the object that represents the SPDM GET_MEASUREMENT response.
`get_number_of_blocks(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) **Returns** [int]: The number of blocks.	Fetches the number of measurement blocks field in the object that represents the SPDM GET_MEASUREMENT response.
`set_number_of_blocks(self, value)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • value (int): the number of blocks. **Returns** N/A	Sets the number of measurement blocks field in the object that represents the SPDM GET_MEASUREMENT response.

Method	Description
`get_measurement_record_length(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) **Returns** [int]: The length of measurement record in bytes.	Fetches the length of the measurement record length field in the object that represents the SPDM GET_MEASUREMENT response.
`set_measurement_record_length(self, value)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • value (int): the length of measurement records in bytes. **Returns** N/A	Sets the length of the measurement record length field in the object that represents the SPDM GET_MEASUREMENT response.
`get_measurement_record(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) **Returns** [MeasurementRecord]: The object that represents the measurement record.	Fetches the MeasurementRecord object that represents the measurement record of the SPDM GET_MEASUREMENT response.
`set_measurement_record(self)` **Input Parameters** • self (SpdmMeasurementResponseMessage class object reference) • [MeasurementRecord]: the object representing the measurement record. **Returns** N/A	Assigns the MeasurementRecord object to the MeasurementRecord field in the SpdmMeasurementResponseMessage class.

7.2.4 RIM

This module handles the RIM parsing and authentication operations.

Table 7-8. RIM APIs

Method	Description
`get_element(parent_element, name of element)` **Input Parameters** • parent_element (lxml.etree._Element): the parent of the required element • name_of_element (str): the name of the required element **Returns** [lxml.etree._Element]:]:The required child element with the given name.	Static method that returns the child element of the parent_element with the given name.
`get_all_elements(parent_element, name_of_element)` **Input Parameters** • parent_element (lxml.etree._Element): the parent of the required element • name_of_element (str): the name of the required element **Returns** [list]: the list of all the elements with given name which is the child of the parent element	A static method that gives all the child elements of the parent_element with the given name.
`read(base_RIM_path)` **Input Parameters** • base_RIM_path (str): the path to the signed base RIM **Returns** root (lxml.etree._Element): The root element of the base RIM.	Reads the signed base RIM from the disk.
`validate_schema(self, schema_path)` **Input Parameters** • self (RIM class object reference) • schema_path (str): the path to the swidtag schema xsd file. **Returns**	Performs the schema validation of the base RIM against a given swid schema.

Method	Description
Returns True if the schema validation is successful, and otherwise, False.	
`_print_base_RIM (self)` **Input Parameters** • self (RIM class object reference) **Returns** N/A	Prints the name and attributes of the software Identity child tags in the base RIM.
`get_colloquial_version (self)` **Input Parameters** • self (RIM class object reference) **Returns** Returns the colloquialVersion attribute of the Meta element.	Parses RIM to return the driver version, which is in the RIM as a colloquial version.
`Extract_certificate (self)` **Input Parameters** • self (RIM class object reference) **Returns** Returns the X509 certificate as bytes.	Extracts the x509 certificate in PEM format from the base RIM.
`verify_signature (self, settings)` **Input Parameters** • self (RIM class object reference) • settings (config.HopperSettings): the object containing the various config info. **Returns** [bool]: Returns True if signature verification is successful, and otherwise, raises the RIMSignatureVerificationError.	Performs the signature verification of the base RIM that contains the golden measurements.
`get_measurements (self)` **Input Parameters** • self (RIM class object reference) **Returns** [dict]: The dictionary that contains the golden measurement.	Returns the dictionary object that contains the golden measurement.
`parse_measurements (self, settings)` **Input Parameters** • self (RIM class object reference)	Lists the measurements of the Resource tags in the base RIM.

Method	Description
• settings (config.HopperSettings): the object containing the various config info. **Returns** N/A	
`verify (self, version, settings, schema_path)` **Input Parameters** • self (RIM class object reference) • version (str): the driver/vbios version of the required RIM. • settings (config.HopperSettings): the object containing the various config info. • schema_path (str): the path to the swidtag schema xsd file. Default value is "./rim/swid_schema_2015.xsd" **Returns** Returns True if schema validation and signature verification passes, and otherwise, False.	Performs the schema validation. If the validation is successful, the signature verification is completed. If both tests pass, it returns True, and otherwise, False.

Table 7-9. RIM Exceptions

Exception	Description
RIMFetchError(RIMError)	It is raised when the required RIM file cannot be fetched.
ElementNotFoundError(RIMError)	It is raised when the required element is not found in the RIM file.
EmptyElementError(RIMError)	It is raised when the content of an element in the RIM file is empty.
RIMSignatureVerificationError(RIMError)	It is raised when the signature verification of the RIM file fails.
InvalidCertificateError(RIMError)	It is raised when there is a problem extracting the X509 certificate from the RIM file.
NoRIMMeasurementsError(RIMError)	It is raised when there are no measurement values in the RIM file.
FileNotFoundError(RIMError)	It is raised when the required file is not found.
RIMVerificationFailureError(RIMError)	It is raised when the verification of RIM fails.
RIMSchemaValidationError(RIMError)	It is raised when the RIM schema validation fails.
InvalidRIMNameError(RIMError)	It is raised when the name assigned to the RIM class is not *driver* or *vbios*.

Exception	Description
RIMCertChainVerificationError(RIMError)	It is raised in case of the RIM certificate chain verification fails.
RIMCertChainOCSPVerificationError(RIMError)	It is raised in case the RIM certificate chain OCSP status verification fails.

7.2.5 Verifier

This module compares the runtime measurements from the attestation report to the reference measurements from RIM.

Table 7-10. Verifier APIs

Method	Description
`__init__(self, attestation_report_obj, driver_rim_obj, vbios_rim_obj, settings)` **Input Parameters** • attestation_report_obj (AttestationReport): the attestation report. • driver_rim_obj (rim.RIM): the driver RIM object containing the the driver golden measurements. • vbios_rim_obj (rim.RIM): the vbios RIM object containing the vbios golden measurement. • settings (config.HopperSettings): the object containing the various config info. **Returns** [Verifier]: The Verifier class object reference.	The constructor method for the Verifier class.
`verify(self, settings)` **Input Parameters** • Self (Verifer class object reference) • settings (config.HopperSettings): the object containing the various config info. **Returns** True if all the golden measurements match the GPU runtime measurements, and otherwise, False	Compares the measurements in the list of GPU measurements with the corresponding golden measurements.
`generate_golden_measurement_list(self,driver_golden_measurements,`	This method takes the driver and VBIOS golden measurements and combines them into a dictionary

Method	Description
`vbios_golden_measurements, settings)` **Input Parameters** • Self (Verifer class object reference) • driver_golden_measurements (dict): the dictionary containing the driver golden measurements • vbios_golden_measurements (dict): the dictionary containing the vbios golden measurements • settings (config.HopperSettings): the object containing the various config info. **Returns** N/A	with the measurement index as the key and the golden measurement object as the value.

Table 7-11. Verifier Exceptions

Exception	Description
MeasurementMismatchError(VerifierError)	It is raised when a runtime measurement does not matches the golden value.
InvalidMeasurementIndexError(VerifierError)	It is raised when the same measurement value index is active in the driver and VBIOS RIM file.

7.3 List of Used Python Libraries

7.3.1 Standard Libraries

Table 7-12. Standard Libraries

Library	Link	Description/Usage	License
Python3	https://docs.python.org/3/	A language interpreter .	Python Software Foundation License (https://docs.python.org/3/license.html)
Os	Standard Python library https://docs.python.org/3/library/os.html	To get the path of the files to be read/written.	
Sys	standard python library	To exit in case of exception	

Library	Link	Description/Usage	License
	https://docs.python.org/3/library/sys.html		
Io	Standard Python library https://docs.python.org/3/library/io.html	To read files	
Argparse	standard python library https://docs.python.org/3/library/argparse.html	To parse the command line arguments	
Hashlib	standard python library https://docs.python.org/3/library/hashlib.html	To calculate the hash values.	
Copy	standard python library https://docs.python.org/3/library/copy.html	For deepcopy	
subprocess	standard python library https://docs.python.org/3/library/subprocess.html	To create the verifier process from cc_admin tool.	
secrets	standard python library https://docs.python.org/3/library/secrets.html	To generate random numbers.	
json	standard python library https://docs.python.org/3/library/json.html	To parse and print Attestation SDK JWT token in logs	
re	standard python library https://docs.python.org/3/library/re.html	To search the PEM certificate end delimiter in the certificate chain.	

7.3.2 Additional Libraries

Table 7-13. Standard Libraries

Library	Link	Description / Usage	License
Cryptography	https://pypi.org/project/cryptography/	For the x509 certificate in RIM and to generate the keys to test the	Apache Software License, BSD License (BSD or Apache License, Version 2.0)

Library	Link	Description / Usage	License
		signature verification.	
Lxml	https://pypi.org/project/lxml/	For parsing the RIM.	BSD License (BSD)
Signxml	https://pypi.org/project/signxml/	For signature verification of the RIM.	Apache Software License (Apache Software License)
Xmlschema	https://pypi.org/project/xmlschema/	For validating the RIM against the swidtag schema.	MIT License (MIT)
nvidia-ml-py	https://pypi.org/project/nvidia-ml-py/	For the using pynvml, which is a Python binding, to use the nvml APIs and communicating with the guest RM.	BSD License (BSD)
Ecdsa	https://pypi.org/project/ecdsa/	For the ECC digital signature verification of the measurements in the attestation report.	MIT
pyOpenSSL	https://pypi.org/project/pyOpenSSL/	For the GPU certificate chain verification.	Apache Software License (Apache License, Version 2.0)
PyJWT	https://pypi.org/project/PyJWT/	For creating JWT token required for Attestation SDK	MIT

7.4 Tool Usage

Refer to the README file in the package for more information about the set up and usage.

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 (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes 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, Hopper are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. 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