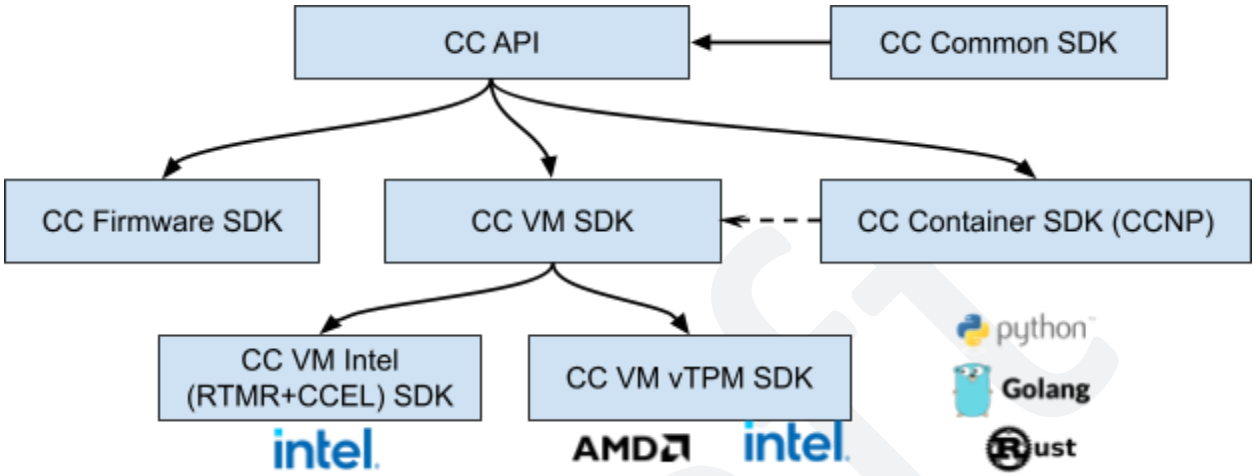
Confidential Computing API Specification

Nov 15, 2023

1. Overview

CC API provides vendor agnostic and TCG compliance APIs supporting access and produce confidential primitives (measurement, event log, quote) for zero-trust design by supporting multiple deploy environments (firmware/VM/cloud native cluster) and multiple programming languages within CVM (confidential virtual machine) guest.



Name	Description
CC API	unified APIs to consume the confidential primitives: eventlog, measurement and quote etc
CC Common SDK	Common functionalities or utilities to process the confidential primitives
CC Firmware SDK	Firmware implementation for different vendors like Intel td-shim/TDVF, AMD SVSM etc
CC VM SDK	VM implementation via /dev/cc-guest, /dev/tpm0, ACPI table
CC Container SDK	Container's implementation for measurement per node, namespace or container
CC VM Intel SDK	VM implementation for Intel via CCEL+RTMR
CC VM vTPM SDK	VM implementation for standard vTPM

2. Definitions

2.1. Vendor Type

Vendor Type pertains to the hardware version and vendor designation of TEE (Trusted Execution Environment) firmware/hardware.

For example, Intel TDX v1.0-1.5, AMD SEV/SEV ES/SEV SNP, ARM CCA v9.4

It can be compatible with the EFI_CC_TYPE from CC_MEASUREMENT_PROTOCOL at here

```
//
// EFI_CC Type/SubType definition
//
#define EFI_CC_TYPE_NONE  0
#define EFI_CC_TYPE_SEV   1
#define EFI_CC_TYPE_TDX   2
#define EFI_CC_TYPE_CCA   3
```

2.2. Event Log ACPI Table

TPM2 ACPI table is defined in TCG ACPI Specification as follows. The log area address in physical memory is given via the field of Lasa (Log Area Start Address), and the length is given via the field of Laml (Log Area Minimum Length)

```
typedef struct {
  EFI_ACPI_DESCRIPTION_HEADER    Header;
  // Flags field is replaced in version 4 and above
  //    BIT0~15:  PlatformClass      This field is only valid for version 4 and above
  //    BIT16~31: Reserved
  UINT32                         Flags;
  UINT64                         AddressOfControlArea;
  UINT32                         StartMethod;
  // UINT8                       PlatformSpecificParameters[];  // size up to 12
  // UINT32                      Laml;                          // Optional
  // UINT64                      Lasa;                          // Optional
} EFI_TPM2_ACPI_TABLE;
```

There are similar structure for CCEL ACPI table (ACPI spec 6.5) at here

```
typedef struct {
  EFI_ACPI_DESCRIPTION_HEADER    Header;
  EFI_CC_TYPE                    CcType;
  UINT16                         Rsvd;
  UINT64                         Laml;
```

```
  UINT64                        Lasa;
} EFI_CC_EVENTLOG_ACPI_TABLE;
```

2.3. Event Log Entry

Refer TCG EFI Protocol Specification, the TPM2 compatible event log format is as follow:

```
typedef struct tdTCG_PCR_EVENT2 {
 TCG_PCRINDEX PCRIndex; //PCRIndex event extended to
 TCG_EVENTTYPE EventType; //Type of event (see [2])
 TPML_DIGEST_VALUES Digests; //List of digests extended to PCRIndex
 UINT32 EventSize; //Size of the event data
 UINT8 Event[EventSize]; //The event data
} TCG_PCR_EVENT2; //Structure to be added to the Event Log
```

Field	Offset	Size	Content
PCRIndex	0x00	4	2
EventType	0x04	4	EV_SEPARATOR (4)
Digests	0x08		
Digests.Count	0x08	4	1
Digests.Digests	0x0C		
Digests.Digests[0].AlgorithmID	0x0C	2	SHA384(0xC)
Digests.Digests[0].Digest	0x0E	48	D6 07 C0 EF B4 1C 0D 75 7D 69 BC A0 61 5C 3A 9A C0 B1 DB 06 C5 57 D9 92 E9 06 C6 B7 DE E4 0E 0E 03 16 40 C7 BF D7 BC D3 58 44 EF 9E DE AD C6 F9
EventSize	0x3E	4	4
Event	0x42	4	0x00, 0x00, 0x00, 0x00

Note: SHA384 is referring to the specification of TCG Algorithms Registry

The event log for CC MEASUREMENT protocol is here

```
//
// Crypto Agile Log Entry Format.
// It is similar with TCG_PCR_EVENT2 except the field of MrIndex and PCRIndex.
```

```
//
typedef struct {
  EFI_CC_MR_INDEX       MrIndex;
  UINT32                EventType;
  TPML_DIGEST_VALUES    Digests;
  UINT32                EventSize;
  UINT8                 Event[1];
} CC_EVENT;
```

The example output is

```
==== TDX Event Log Entry - 69 [0x7CBA8D13] ====
RTMR              : 2
Type              : 0xD (EV_IPL)
Length            : 220
Algorithms ID     : 12 (TPM_ALG_SHA384)
Digest[0] :
00000000  CF 85 4B 75 16 E8 3E 28 8F DA 96 24 37 14 63 DE  ..Ku..>(...$7.c.
00000010  5C 54 59 CC 20 90 12 34 1B 65 B5 6F 1C 9D DB 7D  \TY. ..4.e.o...}
00000020  F3 64 BD 6B E9 46 18 95 D3 E3 D0 A5 E7 E7 91 39  .d.k.F.........9
RAW DATA: ---------------------------------------------
7CBA8D13  03 00 00 00 0D 00 00 00 01 00 00 00 0C 00 CF 85  ................
7CBA8D23  4B 75 16 E8 3E 28 8F DA 96 24 37 14 63 DE 5C 54  Ku..>(...$7.c.\T
7CBA8D33  59 CC 20 90 12 34 1B 65 B5 6F 1C 9D DB 7D F3 64  Y. ..4.e.o...}.d
7CBA8D43  BD 6B E9 46 18 95 D3 E3 D0 A5 E7 E7 91 39 9A 00  .k.F.........9..
7CBA8D53  00 00 67 72 75 62 5F 63 6D 64 20 6C 69 6E 75 78  ..grub_cmd linux
7CBA8D63  20 28 68 64 30 2C 6D 73 64 6F 73 33 29 2F 62 6F   (hd0,msdos3)/bo
7CBA8D73  6F 74 2F 76 6D 6C 69 6E 75 7A 2D 35 2E 31 39 2E  ot/vmlinuz-5.19.
7CBA8D83  30 2D 74 64 78 2E 76 31 2E 34 2E 6D 76 70 37 2E  0-tdx.v1.4.mvp7.
7CBA8D93  65 6C 38 2E 78 38 36 5F 36 34 20 72 6F 6F 74 3D  el8.x86_64 root=
7CBA8DA3  55 55 49 44 3D 64 63 34 61 62 31 61 38 2D 31 33  UUID=dc4ab1a8-13
7CBA8DB3  37 63 2D 34 30 64 35 2D 38 63 66 63 2D 61 63 36  7c-40d5-8cfc-ac6
7CBA8DC3  65 33 32 63 39 65 35 62 31 20 72 6F 20 72 6F 6F  e32c9e5b1 ro roo
7CBA8DD3  74 3D 2F 64 65 76 2F 76 64 61 33 20 72 77 20 63  t=/dev/vda3 rw c
7CBA8DE3  6F 6E 73 6F 6C 65 3D 68 76 63 30 00              onsole=hvc0.
RAW DATA: ---------------------------------------------
```

2.4. Quote Structure

2.4.1. TPM

TPM2 structures are defined in [Trusted Platform Module Library Part 2: Structures], the TPM2 Quote definition is as follow:

```
//
// Definition of TPMS_QUOTE_INFO Structure <OUT>
//
typedef struct {
```

```
  TPML_PCR_SELECTION pcrSelect; // Information on algID PCR selected and digest

  TPM2B_DIGEST pcrDigest; // Digest of the selected PCR using the hash of the signing key
} TPMS_QUOTE_INFO;
```

2.4.2. Intel TDX

Intel TDX attestation is intended to be used in two phases, the TD VM use TDCALL(TDG.MR.REPORT) function to request the Intel TDX module to generate an integrity-protected TDREPORT structure, then verify and sign the report in SGX Quoting Enclave. Refer [Intel TDX Module Architecture Specification] and [TDX Quoting Library API].

```
//
// Definition of TD_QUOTE Structure
//
typedef struct {
  SGX_QUOTE_HEADER header; //< 0:   The quote header.
  uint16_t type; //< 48: Determines type of Quote body (TEE report)
  //      Architecturally supported values of type:
  //      1 (SGX Enclave Report)
  //      2 (TD Report for TDX 1.0)
  //      3 (TD Report for TDX 1.5)
  uint32_t size; //< 50: Size of Quote Body field.
  uint8_t body[]; //< 54: Data conveyed as Quote Body. Its content depends on the value of
Quote Body Type
  //     1 Byte array that contains SGX Enclave Report.
  //     sgx_report_body_t + (uint32_t)signature_data_len + signature
  //     2 Byte array that contains TD Report for TDX 1.0.
  //     sgx_report2_body_t + (uint32_t)signature_data_len + signature
  //     3 Byte array that contains TD Report for TDX 1.5.
  //     sgx_report2_body_v1_5_t + (uint32_t)signature_data_len + signature
} TD_QUOTE;
```

2.4.3. AMD SEV

Please refer [SEV Secure Nested Paging Firmware ABI Specification (amd.com)]

110h	383:0	AUTHOR_KEY_DIGEST	SHA-384 digest of the Author public key that certified the ID key, if provided in SNP_LAUNCH_FINISH. Zeroes if AUTHOR_KEY_EN is 1.
140h	255:0	REPORT_ID	Report ID of this guest.
160h	255:0	REPORT_ID_MA	Report ID of this guest's migration agent.
180h	63:0	REPORTED_TCB	Reported TCB version used to derive the VCEK that signed this report.
188h – 19Fh		-	Reserved.
1A0h–1DFh	511:0	CHIP_ID	If MaskChipId is set to 0, Identifier unique to the chip as output by GET_ID. Otherwise, set to 0h.
1E0h	63:0	COMMITTED_TCB	CommittedTcb.
1E8h	7:0	CURRENT_BUILD	The build number of CurrentVersion.
1E9h	7:0	CURRENT_MINOR	The minor number of CurrentVersion.
1Eah	7:0	CURRENT_MAJOR	The major number of CurrentVersion.
1Ebh	7:0	-	Reserved.
1Ech	7:0	COMMITTED_BUILD	The build number of CommittedVersion.
1Edh	7:0	COMMITTED_MINOR	The minor version of CommittedVersion.
1Eeh	7:0	COMMITTED_MAJOR	The major version of CommittedVersion.
1Efh	7:0	-	Reserved.
1F0h	63:0	LAUNCH_TCB	The CurrentTcb at the time the guest was launched or imported.
1F8h–29Fh		-	Reserved.
2A0h–49Fh		SIGNATURE	Signature of bytes 0h to 29Fh inclusive of this report. The format of the signature is described in Chapter 10.

Byte Offset	Bits	Name	Description
00h	31:0	VERSION	Version number of this attestation report. Set to 2h for this specification.
04h	31:0	GUEST_SVN	The guest SVN.
08h	63:0	POLICY	The guest policy. See Table 9 for a description of the guest policy structure.
10h	127:0	FAMILY_ID	The family ID provided at launch.
20h	127:0	IMAGE_ID	The image ID provided at launch.
30h	31:0	VMPL	The request VMPL for the attestation report.
34h	31:0	SIGNATURE_ALGO	The signature algorithm used to sign this report. See Chapter 10 for encodings.
38h	63:0	CURRENT_TCB	CurrentTcb.
40h	63:0	PLATFORM_INFO	Information about the platform. See Table 23.
48h	31:5	-	Reserved. Must be zero.
	4:2	SIGNING_KEY	Encodes the key used to sign this report. 0: VCEK. 1: VLEK. 2–6: Reserved. 7: None.
	1	MASK_CHIP_KEY	The value of MaskChipKey.
	0	AUTHOR_KEY_EN	Indicates that the digest of the author key is present in AUTHOR_KEY_DIGEST. Set to the value of GCTX.AuthorKeyEn.
4Ch	31:0	-	Reserved. Must be zero.
50h	511:0	REPORT_DATA	Guest-provided data.
90h	383:0	MEASUREMENT	The measurement calculated at launch.
C0h	255:0	HOST_DATA	Data provided by the hypervisor at launch.
E0h	383:0	ID_KEY_DIGEST	SHA-384 digest of the ID public key that signed the ID block provided in SNP_LAUNCH_FINISH.

2.4.4. ARM CCA

Please refer to ARM website for more details. Format of attestation result EAR is currently adopted in Veraison project (https://github.com/veraison/docs/blob/main/project-overview.md), more information can be found here: https://github.com/veraison/rust-ear. Event log support is still in development in Firmware.

The verified components column of table below shows the minimal set of verified components. If a component comprises multiple sub-images, each image should be verified separately.

Component	Verified components		Version checked components	
	Secure boot verification	Hash recorded for attestation	Anti-rollback check	Version recorded for attestation
Boot ROM	N/A	N/A	N/A	Yes
Main Bootloader	Yes	Yes	Yes	Yes
PSA Root of Trust	Yes	Yes	Yes	Yes
Trusted Subsystems	Yes	Yes	Yes	Yes
Application Root of Trust	Yes	Yes	Yes	Yes
Application Loader	Yes	Yes	Yes	Yes

```
//
// Definition of platform security architecture (psa) token
//

psa-token = {
    psa-nonce
    psa-instance-id
    psa-verification-service-indicator
    psa-profile
    psa-implementation-id
    psa-client-id
    psa-lifecycle
    psa-certification-reference
    ? psa-boot-seed
    psa-software-components
}

psa-client-id-key = 2394
psa-lifecycle-key = 2395
psa-implementation-id-key = 2396
psa-boot-seed-key = 2397
psa-certification-reference-key = 2398
psa-software-components-key = 2399
```

```
psa-verification-service-indicator-key = 2400

nonce-label = 10
ueid-label = 256
profile-label = 265

psa-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64

psa-boot-seed-type = bytes .size (8..32)

psa-boot-seed = (
    psa-boot-seed-key => psa-boot-seed-type
)

psa-client-id-nspe-type = -2147483648...0
psa-client-id-spe-type = 1..2147483647

psa-client-id-type = psa-client-id-nspe-type / psa-client-id-spe-type

psa-client-id = (
    psa-client-id-key => psa-client-id-type
)

psa-certification-reference-type = text .regexp "[0-9]{13}-[0-9]{5}"

psa-certification-reference = (
    ? psa-certification-reference-key =>
        psa-certification-reference-type
)

psa-implementation-id-type = bytes .size 32

psa-implementation-id = (
    psa-implementation-id-key => psa-implementation-id-type
)

psa-instance-id-type = bytes .size 33

psa-instance-id = (
    ueid-label => psa-instance-id-type
)

psa-nonce = (
    nonce-label => psa-hash-type
)

psa-profile-type = "http://arm.com/psa/2.0.0"

psa-profile = (
    profile-label => psa-profile-type
)
```

```
psa-lifecycle-unknown-type = 0x0000..0x00ff
psa-lifecycle-assembly-and-test-type = 0x1000..0x10ff
psa-lifecycle-psa-rot-provisioning-type = 0x2000..0x20ff
psa-lifecycle-secured-type = 0x3000..0x30ff
psa-lifecycle-non-psa-rot-debug-type = 0x4000..0x40ff
psa-lifecycle-recoverable-psa-rot-debug-type = 0x5000..0x50ff
psa-lifecycle-decommissioned-type = 0x6000..0x60ff

psa-lifecycle-type =
    psa-lifecycle-unknown-type /
    psa-lifecycle-assembly-and-test-type /
    psa-lifecycle-psa-rot-provisioning-type /
    psa-lifecycle-secured-type /
    psa-lifecycle-non-psa-rot-debug-type /
    psa-lifecycle-recoverable-psa-rot-debug-type /
    psa-lifecycle-decommissioned-type

psa-lifecycle = (
    psa-lifecycle-key => psa-lifecycle-type
)

psa-software-component = {
  ? &(measurement-type: 1) => text
    &(measurement-value: 2) => psa-hash-type
  ? &(version: 4) => text
    &(signer-id: 5) => psa-hash-type
  ? &(measurement-desc: 6) => text
}

psa-software-components = (
    psa-software-components-key => [ + psa-software-component ]
)

psa-verification-service-indicator-type = text

psa-verification-service-indicator = (
    ? psa-verification-service-indicator-key =>
        psa-verification-service-indicator-type
)
```

Example

The following example shows a PSA attestation token for an hypothetical system comprising two measured software components (a boot loader and a trusted RTOS). The attesting device is in a lifecycle state Section 4.3.1 of SECURED. The attestation has been requested from a client residing in the SPE:

```
//
```

```
// Example of a PSA attestation token
//
{
  / eat_profile /             265: "http://arm.com/psa/2.0.0",
  / psa-client-id /          2394: 2147483647,
  / psa-security-lifecycle / 2395: 12288,
  / psa-implementation-id /  2396: h'0000000000000000000000000000000000000000000000000000000000000000',
  / psa-boot-seed /          2397: h'0000000000000000',
  / psa-certification-reference / 2398: "1234567890123-12345",
  / psa-software-components / 2399: [
    {
      / measurement value / 2: h'0303030303030303030303030303030303030303030303030303030303030303',
      / signer ID /         5: h'0404040404040404040404040404040404040404040404040404040404040404'
    }
  ],
  / nonce /     10: h'0101010101010101010101010101010101010101010101010101010101010101',
  / ueid /     256: h'0102020202020202020202020202020202020202020202020202020202020202',
  / psa-vsi / 2400: "https://veraison.example/v1/challenge-response"
}
```

The JWK representation of the IAK used for creating the COSE Sign1 signature over the PSA token is:

```
//
// Example of JWK representation of the IAK
//
{

  "kty": "EC",

  "crv": "P-256",

  "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",

  "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",

  "d": "870MB6gfuTJ4HtUnUvYMyJpr5eUZNP4Bk43bVdj3eAE"

}
```

The resulting Concise Binary Object Representation (CBOR) Object Signing and Encryption (COSE) object is:

```
//
// Example of the resulting COSE object
```

```
//
18(

  [

    / protected /   h'A10126',

    / unprotected / {},

    / payload /
h'AA19010978186874747 03A2F2F61726D2E636F6D2F7073612F322E302E3019095A1A7FFFFFFF19095B193000190
95C582000000000000000000000000000000000000000000000000000000000000000019095D4800000000000000
0019095E733132333435363738393031323 32D313233343519095F81A20258200303030303030303030303030303
303030303030303030303030303030303030303055820040404040404040404040404040404040404040404040404
0404040404040A582001010101010101010101010101010101010101010101010101010101010101010119010058210
102020202020202020202020202020202020202020202020202020202020202020219096078 2E68747470733A2F2F76
65726169736F6E2E6578616D706C652F76312F6368616C6C656E67652D726573706F6E7365',

    / signature /
h'56F50D131FA83979AE064E76E70DC75C070B6D991AEC08ADF9F41CAB7F1B7E2C47F67DACA8BB49E3119B7BAE77A
EC6C89162713E0CC6D0E7327831E67F32841A'

  ]

)
```

which has the following base16 encoding:

```
//
// Example of the resulting COSE object in base16 encoding
//
18(d28443a10126a059013baa1901097818687474703a2f2f61726d2e636f6d2f7073612f322e302e3019095a1a7f
ffffff19095b19300019095c582000000000000000000000000000000000000000000000000000000000000000019
9095d4800000000000000019095e7331323334353637383930313233 2d313233343519095f81a202582003030303
030303030303030303030303030303030303030303030303030303030558200404040404040404040404040404040
4040404040404040404040404040404040a582001010101010101010101010101010101010101010101010101010101
0101010101190100582101020202020202020202020202020202020202020202020202020202020202020219096078
2e68747470733a2f2f7665726169736f6e2e6578616d706c652f76312f6368616c6c656e67652d726573706f6e73
65584056f50d131fa83979ae064e76e70dc75c070b6d991aec08adf9f41cab7f1b7e2c47f67daca8bb49e3119b7ba
e77aec6c89162713e0cc6d0e7327831e67f32841a
```

2.5. CC Attestation Report

Different with traditional TPM, the CC hardware vendor will provide an attestation report as the base to generate CC quote.

2.5.1. Intel TDX

Intel TDX report is generated by Intel TDX module, the definition is in [Intel TDX Module Architecture Specification].

```
//
// Definition of TD_REPORT Structure
//
typedef struct {
  tee_tcb_svn_t       tee_tcb_svn; //<  0:  TEE_TCB_SVN Array
  tee_measurement_t   mr_seam; //< 16:  Measurement of the SEAM module
  tee_measurement_t   mrsigner_seam; //< 64:  Measurement of a 3rd party
SEAM module’s signer (SHA384 hash). The value is 0’ed for Intel SEAM module
  tee_attributes_t    seam_attributes; //< 112: MBZ: TDX 1.0
  tee_attributes_t    td_attributes; //< 120: TD's attributes
  tee_attributes_t    xfam; //< 128: TD's XFAM
  tee_measurement_t   mr_td; //< 136: Measurement of the initial contents
of the TD
  tee_measurement_t   mr_config_id; //< 184: Software defined ID for
non-owner-defined configuration on the guest TD. e.g., runtime or OS
configuration
  tee_measurement_t   mr_owner; //< 232: Software defined ID for the guest
TD's owner
  tee_measurement_t   mr_owner_config; //< 280: Software defined ID for
owner-defined configuration of the guest TD, e.g., specific to the workload
rather than the runtime or OS
  tee_measurement_t   rt_mr[4]; //< 328: Array of 4(TDX1: NUM_RTMRS is 4)
runtime extendable measurement registers
  tee_report_data_t   report_data; //< 520: Additional report data
  tee_tcb_svn_t       tee_tcb_svn2; //< 584: Array of TEE TCB SVNs (for TD
preserving).
  tee_measurement_t   mr_servicetd; //< 600: If is one or more bound or
pre-bound service TDs, SERVTD_HASH is the SHA384 hash of the TDINFO_STRUCTs
of those service TDs bound. Else, SERVTD_HASH is 0..
} TD_REPORT;
```

2.5.2. AMD SEV

AMD SEV-SNP attestation is also in two phases, the VM requests the firmware to construct an attestation report and signing by Versioned ChipEndorsement Key (VCEK). Refer SEV Secure Nested Paging Firmware ABI Specification and AMD SEV-SNP Attestation.

```
//
// Definition of ATTESTATION_REPORT Structure
//
typedef struct {
       uint32_t          version;                /* 0x000 */
       uint32_t          guest_svn;              /* 0x004 */
       uint64_t          policy;                 /* 0x008 */
       uint8_t           family_id[16];          /* 0x010 */
       uint8_t           image_id[16];           /* 0x020 */
       uint32_t          vmpl;                          /* 0x030 */
       uint32_t          signature_algo;         /* 0x034 */
       union tcb_version platform_version;       /* 0x038 */
       uint64_t          platform_info;          /* 0x040 */
       uint32_t          flags;                  /* 0x048 */
       uint32_t          reserved0;              /* 0x04C */
       uint8_t           report_data[64];        /* 0x050 */
       uint8_t           measurement[48];        /* 0x090 */
       uint8_t           host_data[32];          /* 0x0C0 */
       uint8_t           id_key_digest[48];      /* 0x0E0 */
       uint8_t           author_key_digest[48];  /* 0x110 */
       uint8_t           report_id[32];          /* 0x140 */
       uint8_t           report_id_ma[32];       /* 0x160 */
       union tcb_version reported_tcb;                  /* 0x180 */
       uint8_t           reserved1[24];          /* 0x188 */
       uint8_t           chip_id[64];                   /* 0x1A0 */
       uint8_t           reserved2[192];         /* 0x1E0 */
       struct signature  signature;              /* 0x2A0 */
} ATTESTATION_REPORT;
```

2.5.3. ARM CCA

RMM spec in https://developer.arm.com/documentation/den0137/latest shows the detailed description of Attestation report structure and RSI interface details. Event log support is still in development in Firmware.

A CCA attestation token is a collection of claims about the state of a Realm and of the CCA platform on which the Realm is running. The size of a CCA attestation token may be greater than 4KB. A CCA attestation token consists of two parts: Realm token and CCA platform token.

Realm token contains attributes of the Realm, including, Realm Initial Measurement and Realm Extensible Measurements.
CCA platform token Contains attributes of the CCA platform on which the Realm is running, including, CCA platform identity, CCA platform lifecycle state, and CCA platform software component measurement.

The process for a Realm to obtain an attestation token is get through call RSI_ATTESTATION_TOKEN_INIT once, then call all RSI_ATTESTATION_TOKEN_CONTINUE in a loop, until the result is not RSI_INCOMPLETE Each call to RSI_ATTESTATION_TOKEN_CONTINUE retrieves up to one Granule of the attestation token.

```
int get_attestation_token(...)
{
    int ret;
    uint64_t size, max_size;
    uint64_t buf, granule;

    ret = RSI_ATTESTATION_TOKEN_INIT(challenge, &max_size);
    if (ret) {
        return ret;
    }

    buf = alloc(max_size);
    granule = buf;

    do { // Retrieve one Granule of data per loop iteration
        uint64_t offset = 0;

        do { // Retrieve sub-Granule chunk of data per loop iteration
            size = GRANULE_SIZE - offset;
            ret = RSI_ATTESTATION_TOKEN_CONTINUE(granule, offset, size, &len);
            offset += len;
        } while (ret == RSI_INCOMPLETE && offset < GRANULE_SIZE);

        // "offset" bytes of data are now ready for consumption from "granule"

        if (ret == RSI_INCOMPLETE) {
            granule += GRANULE_SIZE;
        }
    } while ((ret == RSI_INCOMPLETE) && (granule < buf + max_size));

    return ret;
}
```

The CCA attestation token is defined as follows:

```
//
// Example of CCA attestation token
//
cca-token = #6.399(cca-token-collection) ; EAT token-collection extension
```

```
cca-platform-token = bstr .cbor COSE_Sign1_Tagged

cca-realm-delegated-token = bstr .cbor COSE_Sign1_Tagged


cca-token-collection = {

    44234 => cca-platform-token ;

    44234 = 0xACCA 44241 => cca-realm-delegated-token }

; EAT standard definitions

COSE_Sign1_Tagged = #6.18(COSE_Sign1)


; Deliberately shortcut these definitions until EAT is finalised and able to

; pull in the full set of definitions

COSE_Sign1 = "COSE-Sign1 placeholder"
```

The format of the Realm token claim map is defined as follows:

```
//
// Definition of Realm token
//
cca-realm-claims = (cca-realm-claim-map)

cca-realm-claim-map = {

    cca-realm-challenge

    cca-realm-personalization-value

    cca-realm-initial-measurement

    cca-realm-extensible-measurements

    cca-realm-hash-algo-id

    cca-realm-public-key

    cca-realm-public-key-hash-algo-id

}

cca-realm-challenge-label = 10

cca-realm-challenge-type = bytes .size 64

cca-realm-challenge = ( cca-realm-challenge-label => cca-realm-challenge-type )
```

```
cca-realm-personalization-value-label = 44235

cca-realm-personalization-value-type = bytes .size 64

cca-realm-personalization-value = ( cca-realm-personalization-value-label =>
cca-realm-personalization-value-type )

cca-realm-measurement-type = bytes .size 32 / bytes .size 48 / bytes .size 64

cca-realm-initial-measurement-label = 44238

cca-realm-initial-measurement = ( cca-realm-initial-measurement-label =>
cca-realm-measurement-type )
cca-realm-extensible-measurements-label = 44239

cca-realm-extensible-measurements = ( cca-realm-extensible-measurements-label => [ 4*4
cca-realm-measurement-type ] )

cca-realm-hash-algo-id-label = 44236

cca-realm-hash-algo-id = ( cca-realm-hash-algo-id-label => text )

cca-realm-public-key-label = 44237 ; TODO: support public key sizes other than ECC-P384

cca-realm-public-key-type = bytes .size 97

cca-realm-public-key = ( cca-realm-public-key-label => cca-realm-public-key-type )
cca-realm-public-key-hash-algo-id-label = 44240

cca-realm-public-key-hash-algo-id = ( cca-realm-public-key-hash-algo-id-label => text )
```

An example Realm claim map is shown below in Concise Binary Object Representation (CBOR) Object Signing and Encryption (COSE)-DIAG format:

```
//
// Example of Realm claim token
//
cca-token = #6.399(cca-token-collection) ; EAT token-collection extension

/ Realm claim map /

{

/ cca-realm-challenge /

10: h'ABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB',

/ cca-realm-personalization-value /
```

```
44235: h'ABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB',

/ cca-realm-initial-measurement /

44238: h'000000000000000000000000000000000000000000000000000000000000000000',

/ cca-realm-extensible-measurements /

44239: [ h'000000000000000000000000000000000000000000000000000000000000000000',
h'000000000000000000000000000000000000000000000000000000000000000000',
h'000000000000000000000000000000000000000000000000000000000000000000',
h'000000000000000000000000000000000000000000000000000000000000000000' ],

/ cca-realm-hash-algo-id /

44236: "sha-256",

/ cca-realm-public-key /

44237: h'0476F988091BE585ED41801AECFAB858548C63057E16B0E676120BBD0D2F9C29
E056C5D41A0130EB9C21517899DC23146B28E1B062BD3EA4B315FD219F1CBB52
8CB6E74CA49BE16773734F61A1CA61031B2BBF3D918F2F94FFC4228E50919544 AE',

/ cca-realm-public-key-hash-algo-id /

44240: "sha-256"

}
```

The format of the CCA platform token claim map is defined as follows:

```
//
// Definition of CCA platform token claim map
//
cca-platform-claims = (cca-platform-claim-map)
cca-platform-claim-map = {
      cca-platform-profile
      cca-platform-challenge
      cca-platform-implementation-id
      cca-platform-instance-id
      cca-platform-config
      cca-platform-lifecycle
      cca-platform-sw-components
      ? cca-platform-verification-service
      cca-platform-hash-algo-id }

cca-platform-profile-label = 265 ; EAT
profile cca-profile-type = "http://arm.com/CCA-SSD/1.0.0"
```

```
cca-platform-profile = ( cca-platform-profile-label => cca-profile-type )
cca-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-platform-challenge-label = 10
cca-platform-challenge = ( cca-platform-challenge-label => cca-hash-type )
cca-platform-implementation-id-label = 2396 ; PSA implementation ID
cca-platform-implementation-id-type = bytes .size 32
cca-platform-implementation-id = ( cca-platform-implementation-id-label =>
cca-platform-implementation-id-type )
cca-platform-instance-id-label = 256 ; EAT ueid ;

// TODO: require that the first byte of cca-platform-instance-id-type is 0x01 ;
// EAT UEIDs need to be 7 - 33 bytes
cca-platform-instance-id-type = bytes .size 33

cca-platform-instance-id = ( cca-platform-instance-id-label => cca-platform-instance-id-type
)

cca-platform-config-label = 2401 ; // PSA platform range; TBD: add to IANA registration
cca-platform-config-type = bytes
cca-platform-config = ( cca-platform-config-label => cca-platform-config-type )
cca-platform-lifecycle-label = 2395 ; PSA lifecycle
cca-platform-lifecycle-unknown-type = 0x0000..0x00ff
cca-platform-lifecycle-assembly-and-test-type = 0x1000..0x10ff
cca-platform-lifecycle-cca-platform-rot-provisioning-type = 0x2000..0x20ff
cca-platform-lifecycle-secured-type = 0x3000..0x30ff
cca-platform-lifecycle-non-cca-platform-rot-debug-type = 0x4000..0x40ff
cca-platform-lifecycle-recoverable-cca-platform-rot-debug-type = 0x5000..0x50ff
cca-platform-lifecycle-decommissioned-type = 0x6000..0x60ff

cca-platform-lifecycle-type = cca-platform-lifecycle-unknown-type /
    cca-platform-lifecycle-assembly-and-test-type /
    cca-platform-lifecycle-cca-platform-rot-provisioning-type /
    cca-platform-lifecycle-secured-type /
    cca-platform-lifecycle-non-cca-platform-rot-debug-type /
    cca-platform-lifecycle-recoverable-cca-platform-rot-debug-type /
    cca-platform-lifecycle-decommissioned-type

cca-platform-lifecycle = ( cca-platform-lifecycle-label => cca-platform-lifecycle-type )
cca-platform-sw-components-label = 2399 ; PSA software components
cca-platform-sw-component = {
    ? 1 => text, ; component type
    2 => cca-hash-type, ; measurement value
    ? 4 => text, ; version
    5 => cca-hash-type, ; signer id
    ? 6 => text, ; hash algorithm identifier
}
cca-platform-sw-components = ( cca-platform-sw-components-label => [ +
cca-platform-sw-component ] )

cca-platform-verification-service-label = 2400 ; PSA verification service
cca-platform-verification-service-type = text
cca-platform-verification-service = ( cca-platform-verification-service-label =>
```

```
cca-platform-verification-service-type )
cca-platform-hash-algo-id-label = 2402 ; PSA platform range ; TBD: add to IANA registration
cca-platform-hash-algo-id = ( cca-platform-hash-algo-id-label => text )
```

An example CCA platform claim map is shown below in Concise Binary Object Representation (CBOR) Object Signing and Encryption (COSE)-DIAG format:

```
//
// Example of CCA platform claim map
//
/ CCA platform claim map /
{
/ cca-platform-profile /
265: "http://arm.com/CCA-SSD/1.0.0",
/ cca-platform-challenge /
10: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',
/ cca-platform-implementation-id /
2396: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',
/ cca-platform-instance-id /
256: h'010BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB BB',
/ cca-platform-config /
2401: h'CFCFCFCF',
/ cca-platform-lifecycle /
2395: 12288,
/ cca-platform-sw-components /
2399: [
    {
    / measurement value /
    2:h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',
    / signer id /
    5: h'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
    BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB',
    / version /
    4: "1.0.0",
    / hash algorithm identifier /
    6: "sha-256"
},
{
    / measurement value /
    2: h'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC',
    / signer id /
    5: h'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
```

```
    DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD',
    / version /
    4: "1.0.0",
    / hash algorithm identifier /
    6: "sha-256"
    }
],
  / cca-platform-verification-service /
  2400: "https://cca_verifier.org",
  / cca-platform-hash-algo-id /
  2402: "sha-256"
}
```

3. APIs

3.1. int get_cc_measure(int index, int type, typedef tcg_digest* digest);

- Description

  Get a digest hash of a specific measurement register according to the given index of the register.

  Note: With Intel TDX, when vTPM enabled, you might get an RTMR register or PCR register at same time.

- Parameters

Parameter	Type	Note
index	integer	The index for integrity measurement register
type	Integer: 0 - vtpm 1 - Intel TDX 2 - AMD SEV 3 - ARM CCA	Hardware version and vendor designation of TEE (Trusted Execution Environment) firmware/hardware.

digest	typedef struct tcg_digest {
	  TPM_ALG_ID algoid;
	  uint8 digest[0];
	};
		The measurement digest

- Return (To be Refined)

Ret Value	Description
0	Success
-1	Fail

3.2. int get_cc_eventlog(int start, int count, typedef event** buffer);

- Description

  Get the buffer list of the event log, which is compatible with TCG_PCR_EVENT defined in 2.3. Event Log Entry

- Parameters

Parameter	Type	Note
start	integer	The start index of event log to be returned
count	integer	The count of event logs to be returned
buffer	Record buffer list	The buffer includes the count of event log records.

- Return (To be Refined)

Ret Value	Description
0	Success
-1	Fail

3.3. int get_cc_quote(int nonce, byte* data, TPML_PCR_SELECTION* mr_select, byte** quote, byte** signature);

- Description

  Get quote based on the integrity measurement. For CC native quote, it will based on CC report which including all integrity measurement.

- Parameters

Parameter	Type	Note
nonce	integer	The start index of event log to be returned
data	bytes	Data for quoting
mr_select[count]	Measure register array	Selected measure register for quote generation
quote	bytes of quote	The count of event logs to be returned
signature	bytes	Signature digest

- Return (To be Refined)

Ret Value	Description
0	Success
-1	Fail

3.4. int get_cc_report(typedef report);

- Description

  Get CC vendor report. Please refer 2.5. CC Attestation Report

- Parameters

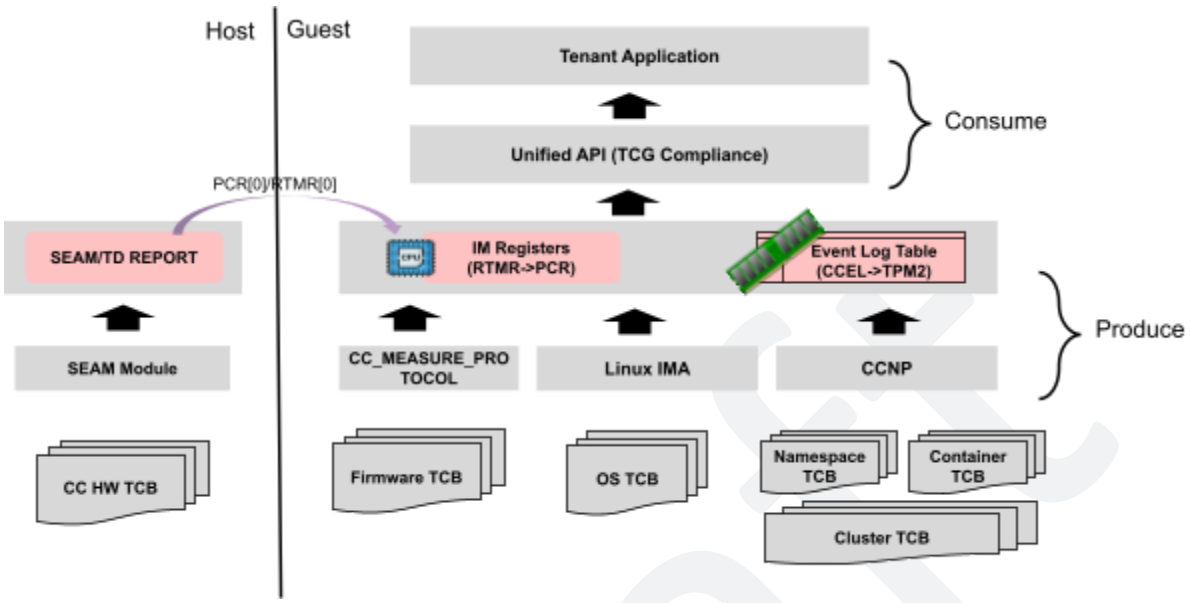
  TBD

- Return (To be Refined)

Ret Value	Description
0	Success
-1	Fail

4. Measurement Produce

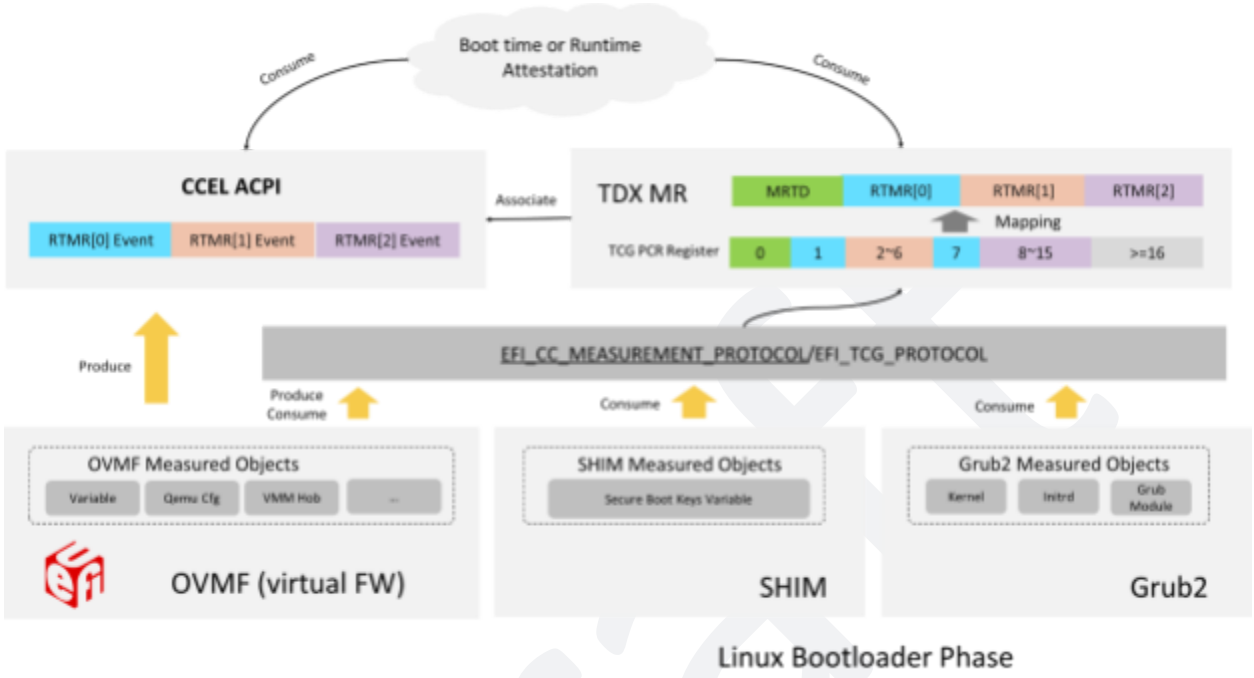
The TCB measurement will be produced via different frameworks/approaches in different phases via different frameworks/loaders:



TCB Measurement	Producer
Firmware TCB	CC_MEASUREMENT_PROTOCOL or TCG_TPM2 protocol
OS Launch Time	CC_MEASUREMENT_PROTOCOL or TCG_TPM2 protocol
OS Runtime	IMA over RTMR or vTPM
Kubernetes	Confidential Cloud Native Primitives for node level, namespace level and container level

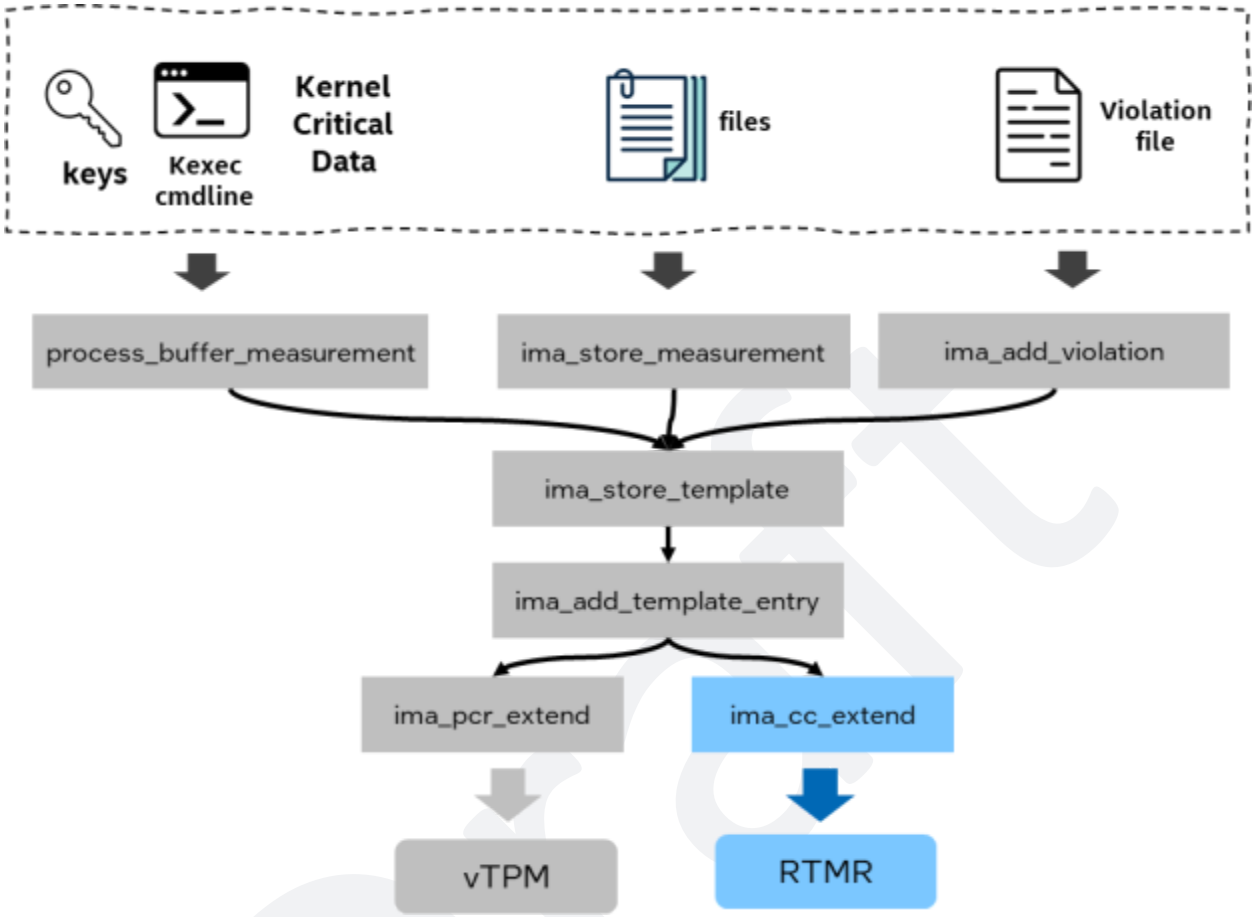
4.1. Firmware

In the firmware phase, the firmware framework will measure the objects and produce the RTMR via EFI_CC_MEASUREMENT_PROTOCOL/EFI_TCG_PROTOCOL to the measurement register.



4.2. VM OS

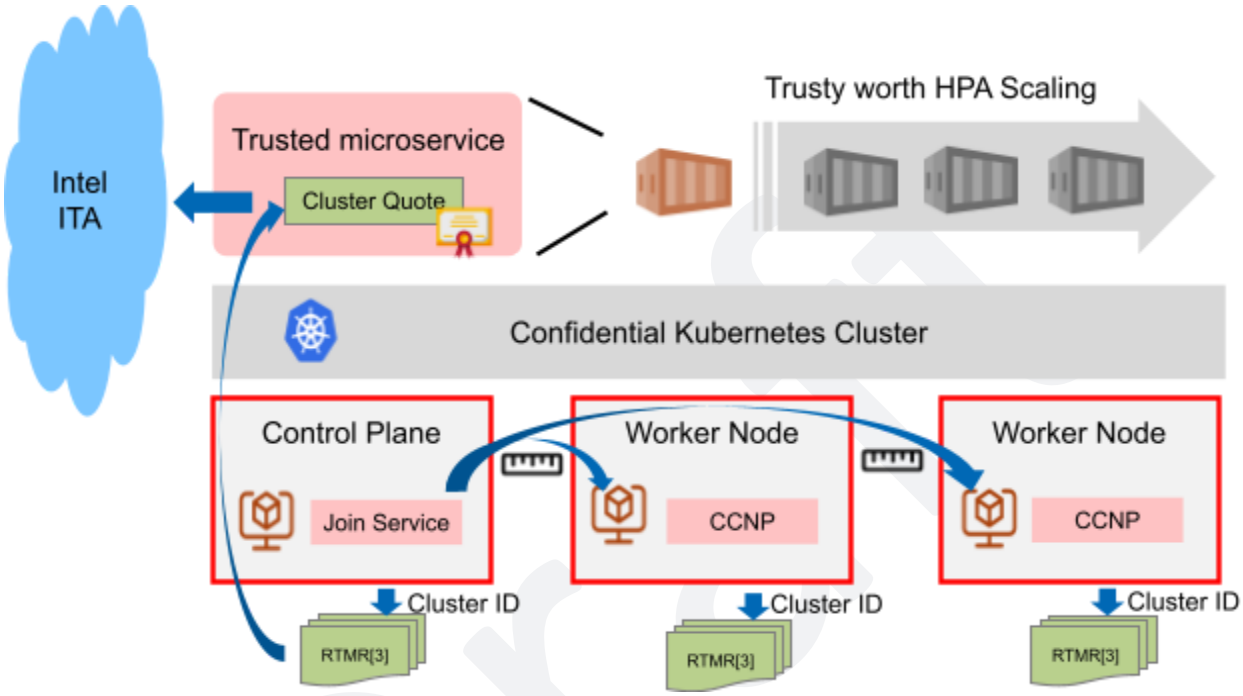
CVM OS measurement is established via Linux IMA (Integrity Measurement Architecture)



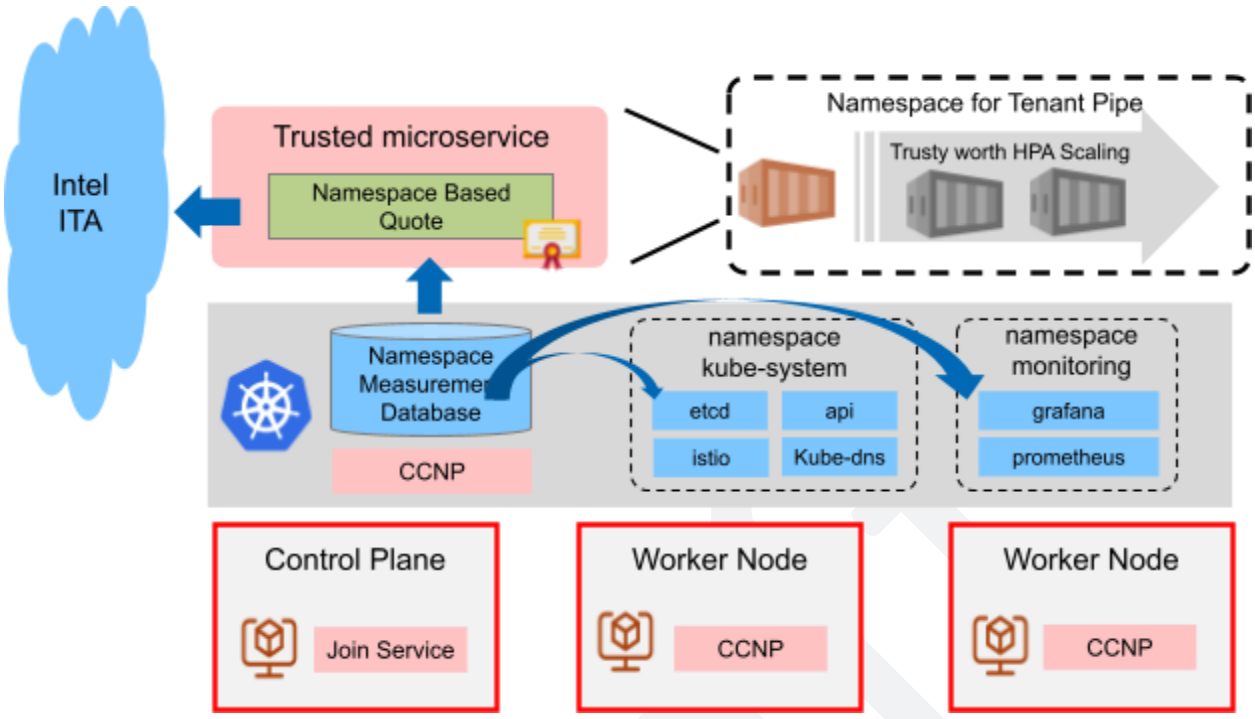
4.3. Cloud Native

4.3.1. Node Level Measurement

Node level measurement is achieved via Join Service when establishing the confidential cluster. It is from [Edgeless Constellation] design.

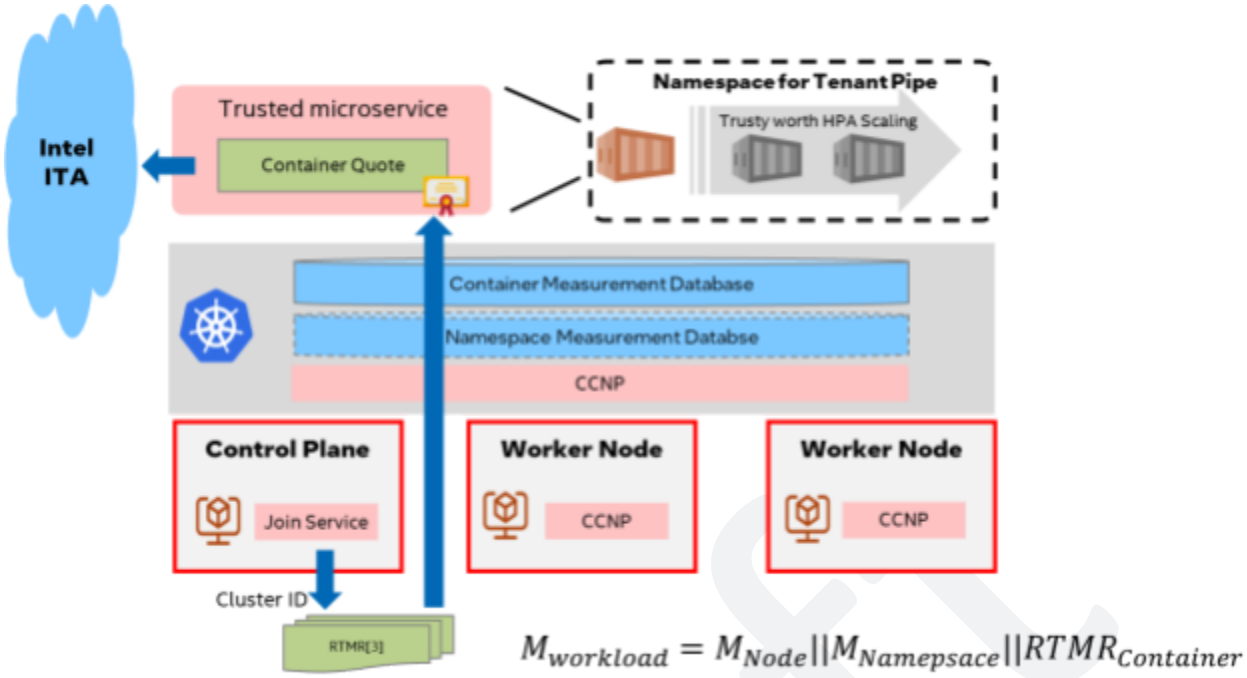


4.3.2. Namespace level measurement

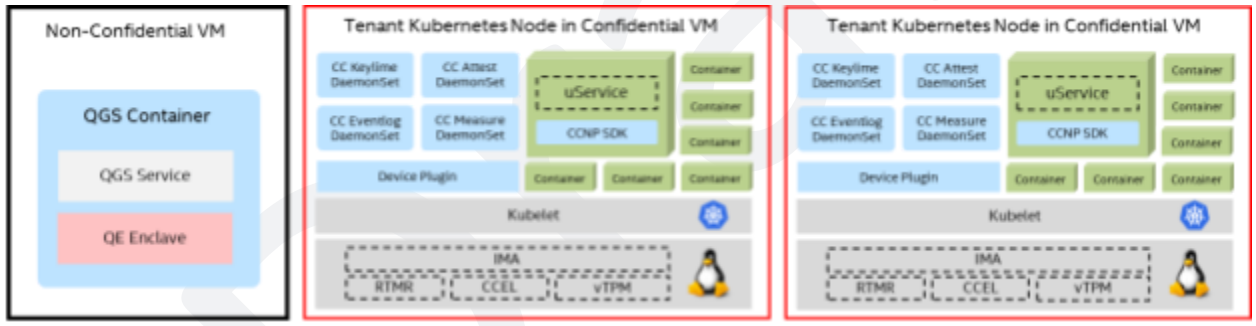


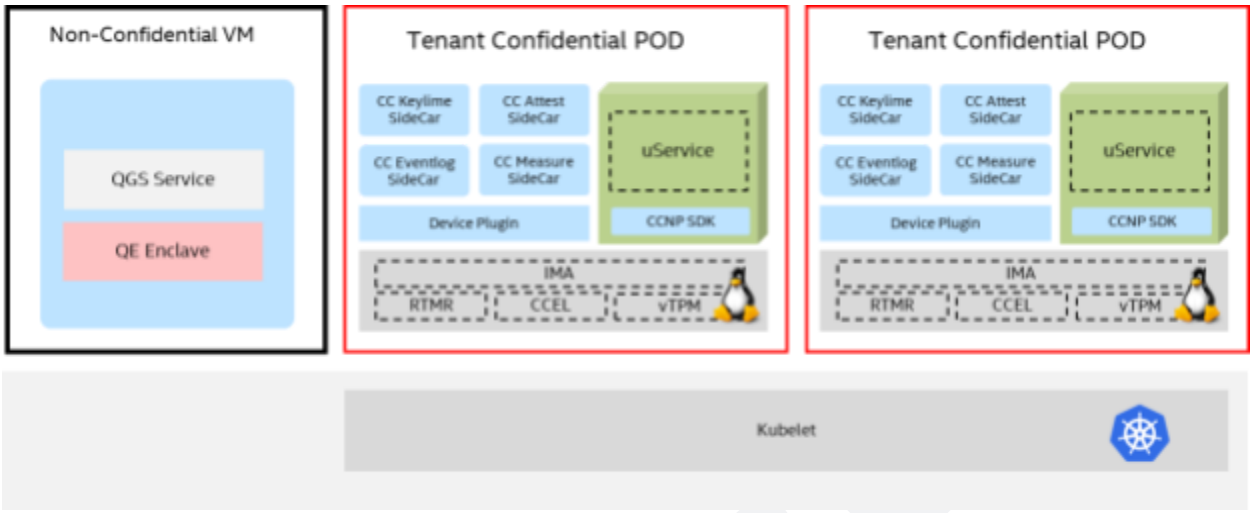
4.3.3. Node level measurement

Workload Container level measurement could be used as base for SPIRE see concept.



4.4. Deployment




Non-Confidential VM
QGS Service
QE Enclave
Tenant Confidential POD
CC Keylime SideCar
CC Attest SideCar
CC Eventlog SideCar
CC Measure SideCar
uService
Device Plugin
CCNP SDK
IMA
RTMR
CCEL
vTPM
Tenant Confidential POD
CC Keylime SideCar
CC Attest SideCar
CC Eventlog SideCar
CC Measure SideCar
uService
Device Plugin
CCNP SDK
IMA
RTMR
CCEL
vTPM
Kubelet