TRUSTED® COMPUTING GROUP

S P E C I F I C A T I O N

Canonical Event Log Format

Version: 1.0
Revision: 0.30
December 11, 2020

Contact: admin@trustedcomputinggroup.org

PUBLIC REVIEW

Work in Progress

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

DISCLAIMERS, NOTICES, AND LICENSE TERMS

CHANGE HISTORY

REVISION	DATE	DESCRIPTION
REVISION	DATE	• DESCRIPTION
1.00/.10	April 3, 2018	• Initial Release of Draft Version 1.00 Revision 0.10.
1.00/.11	June 10, 2018	• To add new rows to the table, hit the TAB key from this last cell.
1.00/.12	Oct 16, 2018	• Change document to an Information Model. Move original information into back of document for future reference.
1.00/.13	July 16, 2019	• Merged comments and changed from difference files
1.00/.14	July 17, 2019	• Merged in edits and comments from 2019-06-13 Members Meeting (comments merged in are tagged with 2019-06-13 MM)
1.00/.15	September 03.2019	• Incorporated comments from previous WG Reviews
1.00/.16	December 12, 2019	• Introduce table base data model
1.00/.17	April 21, 2020	• Incorporate TLV as an available encoding, use terms consistently.
1.00/.18	April 29, 2020	• Completed and tested IMA-TLV example
1.00/19	May 6, 2020	• Added IMA_LEGACY example, and moved examples to appendix A.
1.00/20	May 20, 2020	• Clarified NV_index and State Transition fields
1.00/21	June 3, 2020	• Added PCCLIENT_STD example
1.00/22	June 10, 2020	• Added CEL_MGT and PCCLIENT_NG
1.00/23	July 22, 2020	• Added CEL-CBOR CDDL
1.00/24	July 29, 2020	• Minor cleanups for consistency
1.00/25	October 7, 2020	• Final cleanup
1.00/26	October 28, 2020	• Reorganize for clarity and address all comments
1.00/27	October 28, 2020	• Added algorithm reference
1.00/28	November 17, 2020	• Usage of term Extend and Quote consistent • Update references to references numbers
1.00/29	December 9, 2020	• Copy for public review
1.00/30	December 11,2020	• Minor edits prior to public review

Contents

1 Background and Scope

Attestation is the process of reporting a platform's operational state. During boot, and optionally during continued operation, the platform executes components that may impact the trustworthiness of its operational state. To support Attestation, the identity or state of each of these components is recorded such that those identities or states can be reported as authentic (i.e., they are from the expected platform and are integrity protected). This process is called Measurement. When using the TPM [1], the measurement process uses the TPM's Extend[1] operation. DICE [6] has similar operations[2].

A Measurement at a minimum performs an Event operation. This results in the Attestation of an accumulated platform state at a particular point, but there is no visibility into the individual components comprising that point or state. A particular Measurement, may, therefore, record information about the component being measured. This type of measurement is called an Event Log Record. The collection of the Event Log Records is an Event Log.

The Extend operation provides evidence of the integrity of a measurement. The Event Log may be stored in insecure locations. Authenticity of the Event Log is performed using a Quote operation[3].

1.1 Background

Different platform classes and even different processes with a single platform class may create Event Log Records in formats specific and optimized to a particular environment. The PC Client, for example, uses a format [2] defined by TCG that is optimized for that platform's firmware. Rather than put serialization information within each record, this Event Log Record format is optimized for space by requiring Event Log Records to be sequential in memory in order to preserve the sequence of the measurements performed using the Extend operations. This is acceptable for firmware because there are a finite number of measurements during the boot process. Once the platform's Operating System starts, the firmware stops execution and ceases to create measurements. These limited number of Event Log Records can be captured leaving storage, transmission, and serialization, if needed, to higher later layers.

Even within the same platform class, when the firmware completes and stops performing measurements, the Operating System may measure runtime events such as the execution of applications. These Event Log Records may be in historical formats or formats more optimized for the run-time environment. Another significant difference from a firmware measurement is that the Operating System measurement may be continuous, creating more Event Log Records than can be efficiently stored on the platform. The Event Log Records may, therefore, need to be exported from the originating platform to an external storage location. As there is no reliance on the transmission and storage to maintain the Event Log Record sequence, each Event Log Record must contain a unique sequential index number. Further, embedding sequence information within each Event Log Record enables tracking of Event Log Records that have been moved or deleted from their originating platform, and enables efficient transmission from host to host.

1.2 Scope

This specification for the Canonical Event Log covers three layers: a single top-level Information Model, possibly multiple encoding layers, and multiple content layers. This specification provides a single information model for representing a Canonical Event Log Record that can encapsulate native Event Log Records from various sources. This specification also provides a simple Type-Length-Value (TLV) encoding layer and a Concise Binary Object Representation (CBOR) encoding layer. This specification covers TLV and CBOR encapsulation of some content layers, including CEL Management, PCCLIENT [2] and IMA [5] content. While a content layer implementation may

[1] Several TPM commands support the Extend operation. Examples are TPM2_PCR_Extend, TPM2_PCR_Event, the TPM2_hash_* sequence, and even the TPM 1.2 TPM_Extend. For the purpose of this specification, these are equivalent and are collectively called the Extend operation (Note the use of capitalization to distinguish this specific operation.)

[2] DICE performs a similar operation to Extend by performing a series of hashes over its boot components. This operation is also included in the general term Extend used in this specification.

[3] This specification will use the term Quote operation as a general term for the proof of an Attestation such as a series of measurements. A Quote may be performed using the TPM2_PCR_Quote command but other methods are available such as a TPM2_PCR_Read within an Audit Session. Unless otherwise stated, this specification will use the term Quote as a general operation to mean the general operation of proving a series of measurements that may use a key restricted by TPM Policies.

choose to create an exact binary mapping to this information model as its native Event Log Record, other implementations may choose to bind this information model to other formats, such as TLV or CBOR.

As an information model, this specification's normative statements pertain to the type of information which must be included in any encoding layer format. This specification also normatively states the TLV encoding layer and CBOR encoding layers which may be used.

2 Document Style

2.1 Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document’s normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

2.2 Statement Type

Please note an important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it can be considered a kind of normative statements.

EXAMPLE: *Start of informative comment*

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCG specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment

3 References and Terms

3.1 References

[1] "TPM 2.0 Library Specification"; Family "2.0" Level 00 Revision 01.38 September 29, 2016, https://trustedcomputinggroup.org/resource/tpm-library-specification/

[2] "TCG PC Client Platform Firmware Profile Specification"; Family "2.0" Level 00 Revision 1.04 June 3, 2019, https://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/

[3] "TCG Algorithm Registry, Family "2.0", Level 00 Revision 01.24 January 25, 2017 https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/

[4] "TCG PC Client Platform Firmware Integrity Measurement Specification"; Version 0.15 March 31, 2020, https://trustedcomputinggroup.org/wp-content/uploads/TCG_PC_Client_FIM_v1_r40_02dec2020.pdf

[5] Integrity Measurement Architecture (IMA), Part of the official Linux Kernel, http://kernel.org

[6] Dice Architecture: https://trustedcomputinggroup.org/work-groups/dice-architectures/

3.2 Terms

Start of informative comment

3.2.1 Native Event Log Record (NELR)

This is an Event Log formatted by the native environment that produced the measurement. This may be a TCG standard (e.g., as defined by the PC Client Platform Specification [2]) or may be a de-facto standard, such as produced by bootloaders and operating systems.

3.2.2 Canonical Event Log (CEL)

The CEL consists of one or more Canonical Event Log Records.

3.2.3 Canonical Event Log Record (CELR)

An Event Log Record, irrespective of the data representation format, which meets the CEL mandatory requirements. A complete record containing required information for the Verifier to verify the integrity of the information in the Event Log Record and the information needed to assess a platform's trustworthiness. A reference to an Event Log Record does not imply any particular format. For example, a PC Client's Event Log [2] is a complete Event Log Record because it contains the information needed to verify the Event Log information, including the PCR number and the implicit sequence number, (that is indicated by the record's position within the allocated memory). A CELR also contains the critical data needed by the Verifier to determine a platform's trustworthiness

3.2.4 Canonical Event Log Information Model (CEL-IM)

A specification of the data that must be contained in a CELR, irrespective of Encoding.

3.2.5 Canonical Event Log Encoding (CEL-EN)

A specific data format to meet to the CEL requirements. This may be TLV, CBOR, or a simple binary format.

3.2.6 Event Log Critical Data (ELCD)

This is information required by the Verifier to either verify the integrity of the Event Log Record or information used by the Verifier to determine a platform's trustworthiness. A sequence number is an example of Critical Data required to verify to the integrity of the Event Log Record. The Verifier must process the series of Event Log Records in the order they were measured. Another example of ELCD is the PCR number.

3.2.7 Event Log Informative Data (ELID)

This is information that is helpful to the Verifier but is not measured. For example, while the actual hash of a firmware component that is Extended is Event Log Critical Data, the Event Log Record may contain other information such as a string representing firmware's source and version information that may assist in verification. In this example, if a Platform has had several firmware updates, the Verifier may use this Event Log Informative Data to lookup (or request) the expected measured value.

End of informative comment

4 Canonical Event Log Information Model (CEL-IM)

4.1 Canonical Event Log (CEL) Definition

A Canonical Event Log SHALL consist of a list of Canonical Event Log Records (CELR).

4.2 Canonical Event Log Record (CELR) Definition

A Canonical Event Log record SHALL consist of four fields: a record number, a PCR or NV Index number, a digest, and a content field.

4.2.1 CELR General Requirements

4.2.1.1 Maintenance of the Measurement Integrity

A Verifier MUST be able to verify the integrity of all Event Log Critical Data (ELCD) whether the ELCD was generated directly into CELR or whether a utility transformed a Native Event Log Record (NELR) into a CELR.

4.2.1.2 Event Log Critical Data

A Canonical Event Log Record (CELR) MUST contain all information from the original entity that created the measurement. If the CELR is created by transforming a Native Event Log Record (NELR), the transformation MUST allow the Verifier to verify the integrity of the resulting CELR using information from the TPM Quote Operation. Note that the conversion from NELR to CELR may produce more information in the CELR than was in the original NELR. For example, the CELR may contain an explicit sequence number when the NELR contained only an implicit one.

4.2.2 Record Number Field

This identifies the sequence of the CELR within a set of measurements. The Value Element is the sequence number represented as an unsigned integer. This field is referred to as the RECNUM.

The RECNUM value MUST:

1. Start at zero at TPM2_Startup (CLEAR).
2. Monotonically increment with each measurement.
3. Be maintained separately per index, in the case of a log with records for multiple different PCR or NV indices. (For example, IMA is authoritative only over its PCR (i.e. 10) and need not coordinate RECNUM with other PCRs.)
4. Increment regardless of whether the Event is measurement or unmeasured.

4.2.3 Record PCR or NV Index Field

This field indicates which PCR or NV Index was affected by a measurement. As a measurement can Extend only one PCR or NV index, there is no rationale for this field being represented as a bitmap (i.e., TPMS_PCR_SELECTION). The field, therefore, MUST be the integer representation of the PCR or NV index.

4.2.4 Record Digest Field

This field contains the list of the digest values Extended. The Extend method varies with TPM command, so there is no uniform meaning of TPM Extend in this instance, and separate descriptions are unavoidable. If using the TPM2_PCR_Extend command, this field is the data sent to the TPM (i.e., not the resulting value of the PCR after the TPM2_PCR_Extend command completes). If using the TPM2_PCR_Event command, this field contains the digest structure returned by the TPM2_PCR_Event command (that contains the digest(s) submitted to each PCR bank as the internal Extend operation). This field SHALL contain the information from the TPML_DIGEST_VALUES used in the Extend operation. (Exactly how this information is encoded will be described in the selected encoding layer).

4.2.5 Record Event Content Field

This field contains both the Event Log Critical Data (ELCD) and Event Log Informative Data (ELID) as defined by the Content Type Custodian. This specification (CEL) defines the values identifying the Content Type Custodian but the information within the Event Content Value field is entirely defined by the Content Type Custodian.

Note that there are one or more Content Types assigned to the CEL. These are typically management Events or determined to be common across all Content Type domains, so they are defined as part of this specification.

The Content Type Custodian defines what components of the Event Content are hashed to create the Extend value. These components are likely the entire set of Event Log Critical Data (ELCD), but Event Log Informative Data (ELID) MAY be included.

If multiple PCR Banks are Extended, the same method MUST be used for deriving each PCR Bank's digest.

4.3 CELR data type definitions

The data type in Table 1 defines the CELR data type TPMS_EVENT. If a numeric representation is required for the specific encoding (such as binary or TLV) then the values from the column "value" SHALL be used. (Again, this Information Model specifies the data content, not the encoded format.)

Table 1 TPMS_EVENT

DATA TYPE	FIELD NAME	VALUE	DESCRIPTION
Unsigned Integer	recnum	0	Unique Record Number
Unsigned Integer	pcr	1	PCR index
Unsigned Integer	nv_index	2	NV Index
TPML_DIGEST_VALUES	digests	3	Digests Extended
TPMU_EVENTCONTENT	content	CONTENT_TYPE	The event data for this CELR

Note that TPML_DIGEST_VALUES is a complex structure, including variable length arrays of structures. This information model specifies only that the contents and ordinals from TPML_DIGEST_VALUES be used and does not specify how they are encoded. Section 5.3 shows how this may be encoded using TLV.

The enumeration in Table 2 defines the supported content types. If a numeric representation is required for the specific encoding (such as binary or TLV) then the values from the column "value" SHALL be used.

Table 2 CONTENT_TYPE

NAME	VALUE	DESCRIPTION
CEL	4	CEL management; Content managed by TCG / CEL
PCCLIENT_STD	5	PC Client WG defined encapsulated structure
IMA_TEMPLATE	7	Linux-IMA TEMPLATE format
IMA_TLV	8	Linux-IMA TLV format

4.4 CEL Management Event Types

A CEL Management Event consists of a type and depending on the type it might contain additional information. Some of these Events are measured into a PCR whilst others are not. This is denoted accordingly in each part of section 4.4.1.

Table 3 defines the content of a CEL Management Event TPMS_EVENT_CELMGT.

Table 3 TPMS_EVENT_CELMGT

TYPE	FIELD	DESCRIPTION

TPMI_CELMGTTYPE	type	The type of CEL event
TPMU_CELMGT	data	The data of this CEL event

The VALUE column in Table 4 defines the types of CEL Management Events TPMI_CELMGTTYPE.

Table 4 TPMI_CELMGTTYPE

NAME	VALUE	DESCRIPTION
CEL_VERSION	1	Identifies the CEL specification version (Not measured into PCR)
FIRMWARE_END	2	End of firmware events (Not measured into PCR)
CEL_TIMESTAMP	80	Provides a timestamp (Measured into PCR)
STATE_TRANS	81	Identifies a platform state transition (e.g. hibernation) (Measured into PCR)

Table 5 defines the CEL Management Event content TPMU_CELMGT. The referenced types stem from the TPM library specification part 2 (TPMS_EMPTY) or are defined in the following sections.

Table 5 TPMU_CELMGT

TYPE	FIELD	SELECTOR	DESCRIPTION
TPMS_CEL_VERSION	cel_version	CEL_VERSION	Identifies the CEL specification version
TPMS_EMPTY	firmware_end	FIRMWARE_END	End of firmware events This event contains not further data, thus TPMS_EMPTY is matched for the union.
UINT64	cel_timestamp	CEL_TIMESTAMP	Provides a timestamp as UTC time, in Linux seconds from the epoch format.
TPMS_STATE_TRANS	state_trans	STATE_TRANS	Identifies a platform state transition

4.4.1.1 CEL Version

The Event Content Value Element identifies the specification version this Event Log adheres to. Table 6 defines the corresponding data types for TPMS_CEL_VERSION.

Table 6 TPMS_CEL_VERSION

TYPE	FIELD	DESCRIPTION
UINT16	major	Major version (currently 1)
UINT16	minor	Minor version (currently 0)

This Event SHALL NOT be measured, but is included for informational purposes.

4.4.1.2 Firmware End

This Event Content Value Element marks the end of the device's firmware boot phase and the start of the OS / operational phase. It does not contain any further information.

4.4.1.3 CEL Timestamp

This Event Content Value Element contains a timestamp that was Extended.

The data SHALL encoded as UINT64 representing the milliseconds since time-origin (value = 0) according to the Coordinated Universal Time (UTC).

4.4.1.4 State Trans

This Event Content Value Element contains the device's state transition. For example, devices that enter sleep states may want to provide a measured event indicating this transition. Table 7 defines the structure of event data for this type of CEL Management Event TPMS_STATE_TRANS.

Table 7 TPMS_STATE_TRANS

NAME	FIELD	DESCRIPTION
Suspend	0	System suspending
Hibernate	1	System is hibernating
Kexec	2	System is kexec'ing a new kernel

5 Canonical Event Log Encodings (CEL-EN)

Canonical Event Logs MUST use the information model described in section 4. They MAY use TLV or CBOR as specified in this section. If they do use any of these encodings, they MUST follow the encoding specifications in this section.

1) Encodings MUST maintain the coherence of the CEL Fields within a CEL Record (CELR)
2) Encodings SHALL either use key value maps to represent TPMS_ structures or ordered lists (arrays) if key value maps are not supported. The order of these lists SHALL follow the order of the rows in the tables in section 4.
3) Encodings SHALL use the tag values defined in the enumeration defining tables (Tables 1 – 7 above) if the encodings do not support text-based enumeration values.
4) If encodings support several byte orders the encodings SHOULD use Network byte order.
5) The data type definitions in this specification use the data types from the TPM library specification [1]. Encodings SHALL include the specified data types and their referenced TPM data types accordingly, down to the base data type definitions of the TPM library specification.

5.1 Canonical Event Log Record Encoding – TLV (CEL-TLV)

In the TLV encoding, every element of a log entry is a Type-Length-Value (TLV) triple (3-tuple) that is defined in table 8. Figure 1 shows the format of a complete CELR in TLV format, with the four required fields in TLV format.

Table 8 TLV Encoding

<T,L,V> Triple	Size / Type
T	8-bit
L	32-bit, unsigned integer, Network Byte Order = Big-Endian
V	L bytes/octets

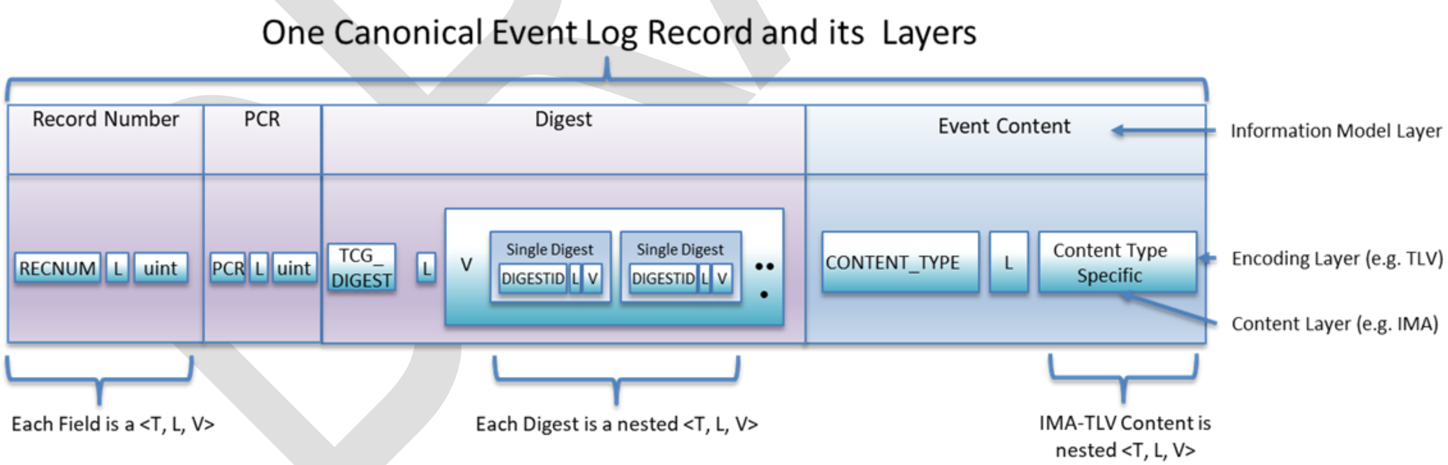

Figure 1 A Canonical Event Log Record in TLV Format

5.1.1 CEL_RECNUM TLV

The CEL record number field is encoded in a TLV of type RECNUM (Table 9).

Table 9 Record Number

<RECNUM> TLV		
T	recnum = 0	From Information Model table 1
L	The length in bytes/octets of the value (V). As the value (V) is a sequential number and since the format is TLV, the length (L) could vary while the log advances. For instance, with a length of 1 (= 1 byte), after the 255th entry, the length (L) could be increased to 2, giving enough space for storing 65535 values/numbers. After that the length increases to 3, and so on and so forth. This specification fixes the size of L to 32 bits.	
V	The value is the sequential number (an unsigned integer) with the length (L), in network byte order.	

5.1.2 CEL_PCR_NVindex TLV

The CEL PCR or NV Index field is encoded in a TLV as shown in Table 10.

Table 10 PCR or NV Index Field

<PCR> TLV		
T	pcr = 1, nv_index = 2	Indicates a PCR or NV index
L	The length in bytes/octets of the value (V). The length (L) could vary, e.g. with a length of 1 (= 1 byte) 255 PCRs can be differentiated. Larger numbers need a greater length (L). This specification fixes the size of L to 32 bits.	
V	The PCR number (unsigned integer) in network byte order	

5.1.3 CEL_DIGESTS TLV

The CEL digest field TLV is nested, with one or more sub-TLV for each bank's digest.

Table 11 Digests encoding

<DIGEST> TLV				
T		Digest = 3	From Information Model table 1	
L		Length of the entire digest content value 'V'	Network byte order	
V		One or more following digest TLV		
	T	SHA1	4	From TPML_DIGEST_VALUES
		SHA256	11	
		SHA384	12	
		SHA512	13	

		SM2	27	
		SM3	18	
	L		The length of the digest indicated by T	
	V		The digest of the type indicated by T and the length indicated by L	

5.1.4 CEL_CONTENT TLV

The CEL Event Content field is encoded in a TLV as shown in Table 12.

Table 12 Content encoding

<CONTENT> TLV			
T	CEL	4	Management Content Type from Information Model Table 2
	PCCLIENT_STD	5	PC Client WG defined encapsulated structure [2]
	IMA_TEMPLATE	7	Linux IMA_TEMPLATE format [5]
	IMA_TLV	8	IMA directly stored in CEL-TLV format
L			The length of the value (V) network byte order
V			The value, according to the type (T)

5.1.5 IMA_TLV Content Layer

IMA_TLV is a Linux kernel measurement content layer, in which IMA [5] records are stored directly in CEL format with TLV encoding using existing fields of the TCG Canonical Event Log Record Information Model (CEL_IM) and TLV Encoding. In IMA_TLV, the digest field values are hashes across the entire content field TLV data, thus simplifying the verification, while protecting all the type and length and value information in the field's TLV and nested TLV's. The overall format is shown in figure 2.

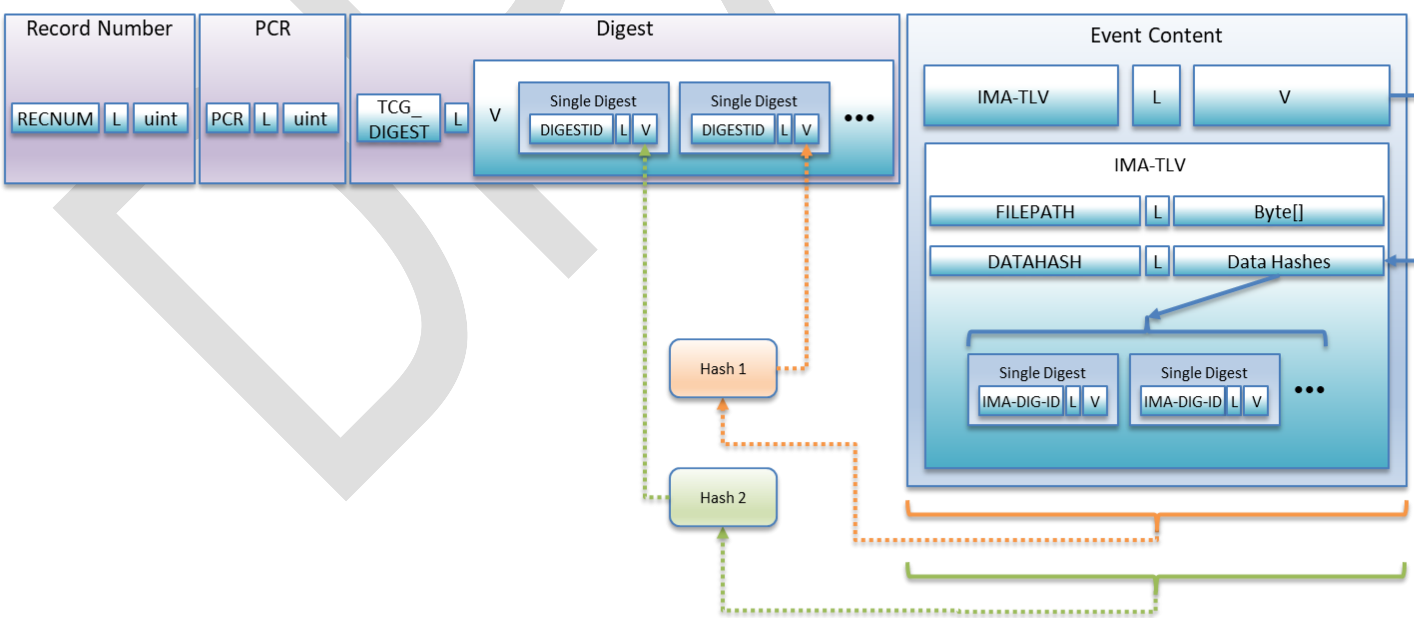

Figure 2: IMA_TLV encoding

A Linux CEL-TLV-IMA content field is a nested TLV with the top-level type IMA_TLV from table 2, containing one or more of the following CEL-TLV-IMA content fields.

Table 13 IMA_TLV Content Encoding

IMA_TLV		
T	IMA_TLV_CONTENT_PATH (0)	full pathname of the file
	IMA_TLV_CONTENT_DATAHASH (1)	The hash over the file content.
	IMA_TLV_CONTENT_DATASIG (2)	A file signature in IMA format, which includes type and keyid.
	IMA_TLV_CONTENT_OWNER (3)	File owner (uid)
	IMA_TLV_CONTENT_GROUP (4)	File group (gid)
	IMA_TLV_CONTENT_MODE (5)	Linux uint16 bitmask of the file permission bits
	IMA_TLV_CONTENT_TIMESTAMP (6)	Linux time_t file creation/modification time
	IMA_TLV_CONTENT_LABEL (7)	LSM security label on the file (e.g. selinux label)
L	The length of the value (V).	
V	The value (V) containing data of the type (T).	

Start of informative comment

The following is a hex/ascii dump of one CELR in the CEL-IMA-TLV format. The dump is repeated, with each TLV formatted CEL field highlighted and explained.

```
00000000  00 00 00 00 04 00 00 00  01 01 00 00 00 04 00 00  |................|
00000010  00 0a 03 00 00 00 19 04  00 00 00 14 4b 47 65 fa  |............KGe.|
00000020  62 21 a2 c6 d6 d2 a2 7c  5c e9 22 df b8 93 e9 3f  |b!.....|\."....?|
00000030  08 00 00 00 26 00 00 00  00 08 2f 62 69 6e 2f 66  |....&...../bin/f|
00000040  6f 01 00 00 00 14 00  01 02 03 04 05 06 07 08  |oo..............|
00000050  09 0a 0b 0c 0d 0e 0f 10  11 12 13                 |...........|
Field: '0' (SEQNUM), Length 4, value 00000001


00000000  00 00 00 00 04 00 00 00  01 01 00 00 00 04 00 00  |................|
00000010  00 0a 03 00 00 00 19 04  00 00 00 14 4b 47 65 fa  |............KGe.|
00000020  62 21 a2 c6 d6 d2 a2 7c  5c e9 22 df b8 93 e9 3f  |b!.....|\."....?|
00000030  08 00 00 00 26 00 00 00  00 08 2f 62 69 6e 2f 66  |....&...../bin/f|
00000040  6f 01 00 00 00 14 00  01 02 03 04 05 06 07 08  |oo..............|
00000050  09 0a 0b 0c 0d 0e 0f 10  11 12 13                 |...........|
Field '1' (PCRNUM) length 4, value 10 (decimal)


00000000  00 00 00 00 04 00 00 00  01 01 00 00 00 04 00 00  |................|
00000010  00 0a 03 00 00 00 19 04  00 00 00 14 4b 47 65 fa  |............KGe.|
00000020  62 21 a2 c6 d6 d2 a2 7c  5c e9 22 df b8 93 e9 3f  |b!.....|\."....?|
00000030  08 00 00 00 26 00 00 00  00 08 2f 62 69 6e 2f 66  |....&...../bin/f|
00000040  6f 01 00 00 00 14 00  01 02 03 04 05 06 07 08  |oo..............|
00000050  09 0a 0b 0c 0d 0e 0f 10  11 12 13                 |...........|
Field '3' (Digest), Length 25,
Nested TLV, type '04' (SHA1) 20 bytes, 4B4765FA6221A2C6D6D2A27C5CE922DFB893E93F
```



```
(This is the digest of the entire following content TLV (bytes 07..13)

00000000  00 00 00 00 04 00 00 00  01 01 00 00 00 04 00 00  |................|
00000010  00 0a 03 00 00 00 19 04  00 00 00 14 4b 47 65 fa  |............KGe.|
00000020  62 21 a2 c6 d6 d2 a2 7c  5c e9 22 df b8 93 e9 3f  |b!.....|\."....?|
00000030  08 00 00 00 26 00 00 00  00 08 2f 62 69 6e 2f 66  |....&...../bin/f|
00000040  6f 01 00 00 00 14 00  01 02 03 04 05 06 07 08  |oo...............|
00000050  09 0a 0b 0c 0d 0e 0f 10  11 12 13                 |...........|
Field '8' (IMA-TLV content), length 38 bytes. This has two nested IMA-TLV contents:
IMA Type '00' (Path) Length 8, value "/bin/foo" and
IMA Type '01' (datahash), length 20, value bytes 00 01…12 13.
```

End of informative comment

5.1.6 IMA_TEMPLATE Content Layer

The existing IMA_TEMPLATE log [5] uses a named "template" to specify the content format. This section specifies CEL-TLV encapsulation of records from the "ima-ng" template native format, which includes "d-ng" (file-hash) and "n-ng" (filename) fields.

Start of informative comment

```
"ima-ng" is a binary format with records of:

      [uint32 little endian pcr]
      [20 digest]
      [uint32 little endian template name len = 6]
            [template name = "ima-ng" not null terminated]
      [uint32 little endian rec len]
            d-ng: [uint32 little endian hash len]*
                  [hash id - e.g. "sha1:" null terminated]*
                  [file hash]*
            n-ng: [uint32 little endian path len]*
                  [path - null terminated]*

For example, here is a hexdump of the first two records in an actual native log:

00000000  0a 00 00 00 2d 92 56 f5  92 9d 55 13 16 09 ff 7c  |....-.V...U....||
00000010  3f 44 b9 ab b6 8a 30 ee  06 00 00 00 69 6d 61 2d  |?D....0.....ima-|
00000020  6e 67 31 00 00 00 1a 00  00 00 73 68 61 31 3a 00  |ng1.......sha1:.|
00000030  5b e8 d5 1b fe af 79 f2  ff 71 41 17 1a b7 a5 d3  |[.....y..qA.....|
00000040  3c 93 8c fc 0f 00 00 00  62 6f 74 5f 61 67 67  |<.......boot_agg|
00000050  72 65 67 61 74 65 00 0a  00 00 00 46 80 a2 18 f5  |regate.....F....|
00000060  20 ce b0 9a c5 2e 8b 61  c8 12 c2 50 5e 2f 67 06  | ......a...P^/g.|
00000070  00 00 00 69 6d 61 2d 6e  67 49 00 00 00 28 00 00  |...ima-ngI...(..|
00000080  00 73 68 61 32 35 36 3a  00 64 a9 81 99 bc 62 58  |.sha256:.d....bX|
00000090  82 15 81 2b 55 c1 24 34  e7 a2 61 f7 b6 ed 93 ea  |...+U.$4..a.....|
000000a0  58 0d 5c 9a ea eb 2d  9c 19 00 00 00 2f 75 73  |X..\...-...../us|
000000b0  72 2f 6c 69 62 2f 73 79  73 74 65 6d 64 2f 73 79  |r/lib/systemd/sy|
000000c0  73 74 65 6d 64 00

This log contains two records: the first is the special "boot_aggregate" event, in which
IMA hashes together PCRS 0-7. The second record, highlighted in yellow, can be parsed as:
```

```
[uint32 little endian pcr]                  0x0a
[20 digest]                                 46 80 ... 2f 67          <-----------|
[uint32 little endian template name len]    0x06                                 |
    [template name]                         "ima-ng"                             |
[uint32 little endian rec len]              0x49                                 |
    d-ng:[uint32 little endian hash len]*   0x28                               ->|
         ["id:"]*                           "sha256:\0"                        ->|
         [file hash]*                       64 a9 ... 2d 9c                    ->|
    n-ng:[uint32 little endian path len]*   0x19                               ->|
         [path]*                            "/usr/lib/systemd/systemd\0'       ->|


The digest is of the record data (marked with '*' above):
28 00 00 00 73 68 61 32 35 36 3a 00 64 a9 81 99 bc 62 58 82 15 81 2b 55 c1 24 34 e7
a2 61 f7 b6 ed 93 ea 58 0d 5c 9a ea eb 2d 9c 19 00 00 00 2f 75 73 72 2f 6c 69 62
2f 73 79 73 74 65 6d 64 2f 73 79 73 74 65 6d 64 00

The sha1 of this binary data is:
      4680a218f520ceb09ac52e8b61c812c2505e2f67
which matches the given digest.

Note that, since the template name, template name length, and record length fields are
not hashed, the parser should use care in using the given values.
```

End of informative comment

Encapsulating IMA_TEMPLATE in CEL-TLV format:

In order to encapsulate IMA_TEMPLATE formatted Event Log Records in CEL_TLV format, add a record number, encapsulate the existing PCR number in TLV, encapsulate the digest(s) (nested), and then nest the template name TLV, and hashed content TLV, into a content TLV. This makes it simpler to hash the content sub-TLV for verification, because it is only slightly different from the nominal hashing of the entire content TLV. This encapsulation is shown in figure 3. The content TLVs SHALL use the type names or numbers as shown in the figure 3. The encapsulation of IMA_TEMPLATE in CEL-TLV SHALL use the field names and values shown in figure 3.

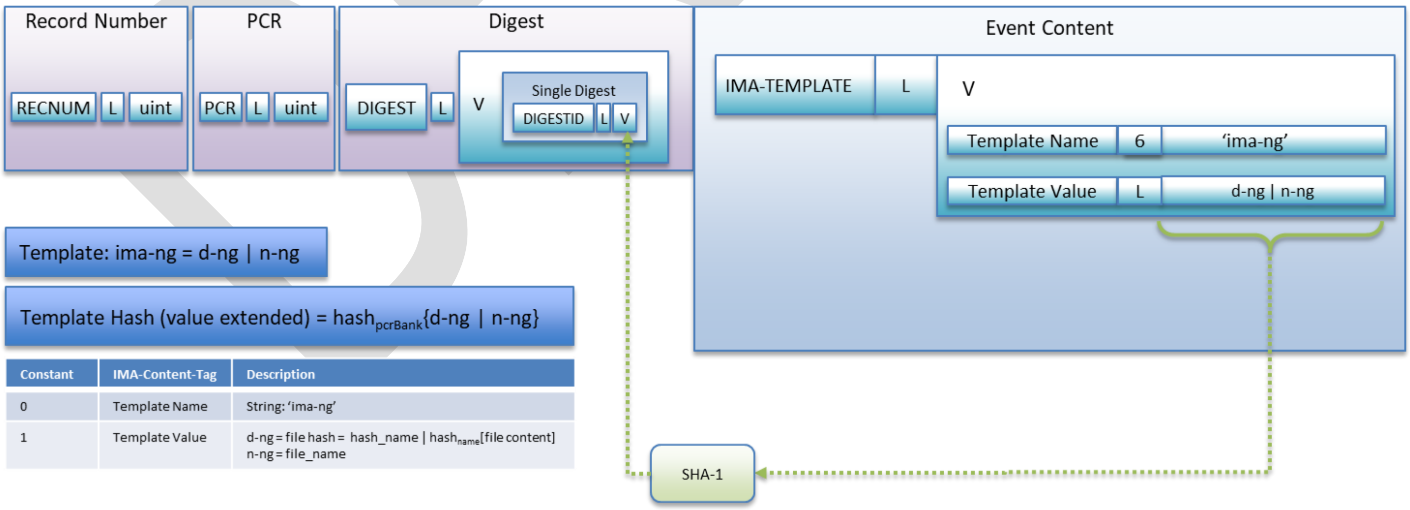

Figure 3 Encapsulating IMA_TEMPLATE in CEL-TLV

Start of informative comment

```
A program was written with these definitions, to translate binary ima-ng formatted logs
into CEL-TLV-IMA-TEMPLATE formatted ones. CEL-TLV encoded ima-ng. Translating the above
ima-ng example, yields the following, with all TYPE bytes in yellow:

00000000  00 00 00 00 04 00 00 00  00 01 00 00 00 04 00 00  |................|
00000010  00 0a 03 00 00 00 19 04  00 00 00 14 2d 92 56 f5  |............-.V.|
00000020  92 9d 55 13 16 09 ff 7c  3f 44 b9 ab b6 8a 30 ee  |..U....|?D....0.|
00000030  07 00 00 00 41 00 00 00  00 06 69 6d 61 2d 6e 67  |....A.....ima-ng|
00000040  01 00 00 00 31 1a 00 00  00 73 68 61 31 3a 00 5b  |....1....sha1:.[|
00000050  e8 d5 1b fe af 79 f2 ff  71 41 17 1a b7 a5 d3 3c  |.....y..qA.....<|
00000060  93 8c fc 0f 00 00 00 62  6f 74 5f 61 67 67 72  |.......boot_aggr|
00000070  65 67 61 74 65 00 00 00  00 00 04 00 00 00 01 01  |egate...........|
00000080  00 00 00 04 00 00 00 0a  03 00 00 00 19 04 00 00  |................|
00000090  00 14 46 80 a2 18 f5 20  ce b0 9a c5 2e 8b 61 c8  |..F.... ......a.|
000000a0  12 c2 50 5e 2f 67 07 00  00 00 59 00 00 00 00 06  |..P^/g....Y.....|
000000b0  69 6d 61 2d 6e 67 01 00  00 00 49 28 00 00 00 73  |ima-ng....I(...s|
000000c0  68 61 32 35 36 3a 00 64  a9 81 99 bc 62 58 82 15  |ha256:.d....bX..|
000000d0  81 2b 55 c1 24 34 e7 a2  61 f7 b6 ed 93 ea 58 0d  |.+U.$4..a.....X.|
000000e0  0d 5c 9a ea eb 2d 9c 19  00 00 00 2f 75 73 72 2f  |.\...-...../usr/|
000000f0  6c 69 62 2f 73 79 73 74  65 6d 64 2f 73 79 73 74  |lib/systemd/syst|
00000100  65 6d 64 00

tlv_dump parses and verifies this tlv encapsulated IMA TEMPLATE as:

SEQNUM 00000000
PCRNUM 10
DIGEST SHA1 2D9256F5929D55131609FF7C3F44B9ABB68A30EE
TEMPLATE NAME ima-ng
filesha1: 5BE8D51BFEAF79F2FF7141171AB7A5D33C938CFC
path: boot_aggregate
Digest matches content.

SEQNUM 00000001
PCRNUM 10
DIGEST SHA1 4680A218F520CEB09AC52E8B61C812C2505E2F67
TEMPLATE NAME ima-ng
filesha256: 64A98199BC62588215812B55C12434EA261F7B6ED93EA580D0D5C9AEAEB2D9C
path: /usr/lib/systemd/systemd
Digest matches content.
```

End of informative comment

5.1.7 PCCLIENT_STD Content Layer

Existing PCCLIENT records [2] are converted to TLV format in much the same way as IMA_TEMPLATE records were specified in section 5.1.6. The overall content is encapsulated in a TLV of type PCCLIENT_STD, which contains two nested TLV, one containing the uint32_t EventType, and one containing the EventContent, with the field values shown in figure 4.

Start of informative comment

The only difference is that in the IMA_TEMPLATE case, all digests were produced by hashing the EventContent value directly. In the case of PCCLIENT_STD, this is true of some of the EventContents, but not all. For example, EV_NO_ACTION events are advisory only, and the digests (of all zeros) are not Extended. In EV_POST_CODE, the digests are of the code or data to be measured, and the digests are Extended into the specified PCR, but the event content contains unverified hints or related information, and not the original data or hash. The TCG PC Client Platform Firmware Profile Specification [2] contains the details of what is in the contents for all Event Types. While an attacker cannot change any of the record number, PCR, and Digest fields without detection by comparison to the actual PCR values, the attacker can freely change all of the event type fields and some of the event content fields. Verifiers must not trust any unverified fields, which are included merely as hints.

End of informative comment

Figure 4 illustrates this encapsulation of the data from the log's native EVENT2 structure into a CEL-TLV encoding:

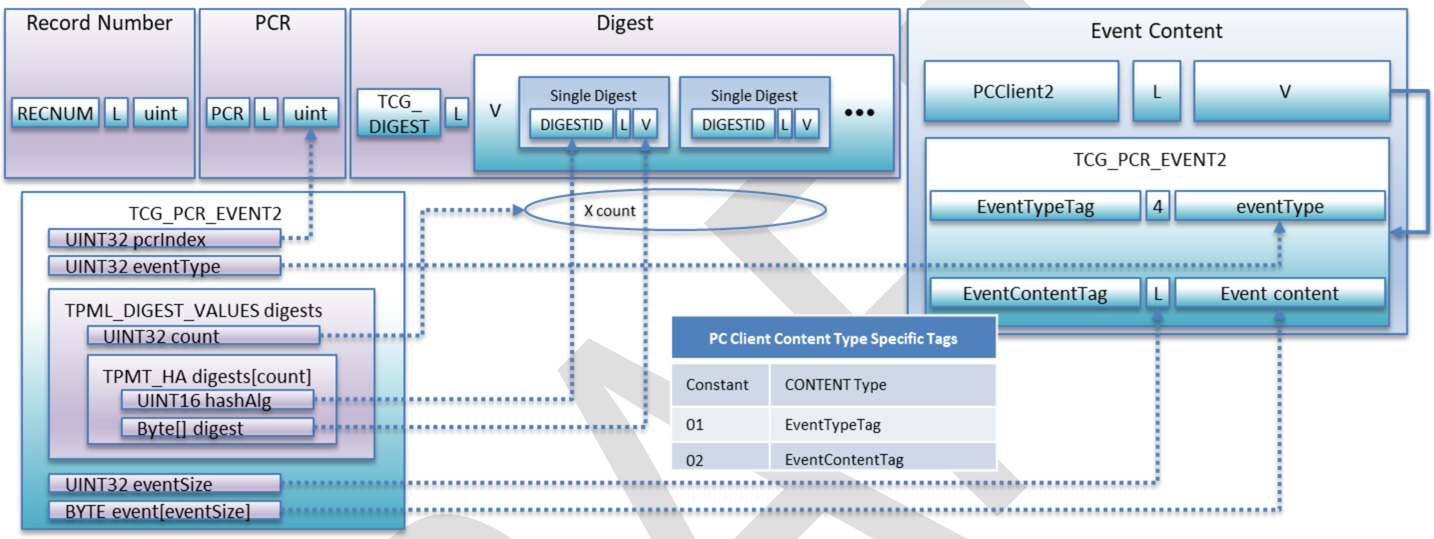

Figure 4 PCCLIENT encapsulation in CEL_TLV format

Start of informative comment

In more detail, the PCCLIENT event log consists of one EVENT-1 structure, followed by one or more Event-2 structures. The EVENT-1 structure is similar to the EVENT-2 shown but has a single fixed SHA-1 digest of 20 bytes, without the count or HashAlg fields.

```
   event-1
      uint32_t    PCR_Index         //little-endian
      uint32_t    EventType         //little-endian
      uint8_t     sha1_digest[20]
      uint32_t    event_data_size   //little-endian
      uint8_t     data[]
   event-2
      uint32_t    PCR_Index         //little-endian
      uint32_t    EventType         //little-endian
      uint32_t    digest_count
      uint16_t    digest1_alg       //little-endian
      uint8_t     digest1[]
      ...
      uint32_t    event_data_size   //little-endian
      uint8_t     data[]
Here is a hexdump of the first two PCCLIENT records from an actual system:

00000000  00 00 00 00 03 00 00 00  00 00 00 00 00 00 00 00  |................|
```



```
00000010  00 00 00 00 00 00 00 00  00 00 00 00 25 00 00 00  |............%...|
00000020  53 70 65 63 20 49 44 20  45 76 65 6e 74 30 33 00  |Spec ID Event03.|
00000030  00 00 00 00 00 02 00 02  02 00 00 00 04 00 14 00  |................|
00000040  0b 00 20 00 00 00 00 00  00 08 00 00 00 02 00 00  |.. .............|
00000050  00 04 00 c4 2f ed ad 26  82 00 cb 1d 15 f9 78 41  |..../..&......xA|
00000060  c3 44 e7 9d ae 33 20 0b  00 d4 72 0b 40 09 43 82  |.D...3 ...r.@.C.|
00000070  13 b8 03 56 80 17 f9 03  09 3f 6b ea 8a b4 7d 28  |...V.....?k...}(|
00000080  3d b3 2b 6e ab ed bb f1  55 10 00 00 00 1e fb 6b  |=.+n....U......k|
00000090  54 0c 1d 55 40 a4 ad 4e  f4 bf 17 b8 3a

Event-1 (yellow):
         PCR:      00 00 00 00        // PCR0
         EventType 03 00 00 00        // 3 - EV_NO_ACTION
         Sha-1     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
         Data size 25 00 00 00        // 37 bytes
         Data      53 70 … 20 00 00

Similarly, Event-2 is:
         PCR       00 00 00 00         // PCR 0
         EventType 08 00 00 00         // 8 - EV_S_CRTM_VERSION
         Dgstcount 02 00 00 00         // two digests
         Dgst1Alg  04 00               // 4 - sha1
         Dgst1     c4 2f ed … ae 33 20
         Dgst2Alg  0b 00               // sha256
         Dgst2     d4 72 0b …bb f1 55
         Datasize  10 00 00 00         // 16 bytes
         Data      1e fb 6b … 17 b8 3a

When translated to PCCLIENT_STD format these two example events become:

00000000  00 00 00 00 04 00 00 00  00 01 00 00 00 04 00 00  |................|
00000010  00 00 03 00 00 00 19 04  00 00 00 14 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  05 00 00 00 33 00 00 00  00 04 00 00 00 03 01 00  |....3...........|
00000040  00 00 25 53 70 65 63 20  49 44 20 45 76 65 6e 74  |..%Spec ID Event|
00000050  30 33 00 00 00 00 00 00  02 00 02 02 00 00 00 04  |03..............|
00000060  00 14 00 0b 00 20 00 00  00 00 00 00 04 00 00 00  |..... ..........|
00000070  01 01 00 00 00 04 00 00  00 00 03 00 00 00 3e 04  |..............>.|
00000080  00 00 00 14 c4 2f ed ad  26 82 00 cb 1d 15 f9 78  |...../..&......x|
00000090  41 c3 44 e7 9d ae 33 20  0b 00 00 00 20 d4 72 0b  |A.D...3 .... .r.|
000000a0  40 09 43 82 13 b8 03 56  80 17 f9 03 09 3f 6b ea  |@.C....V.....?k.|
000000b0  8a b4 7d 28 3d b3 2b 6e  ab ed bb f1 55 05 00 00  |..}(=.+n....U...|
000000c0  00 1e 00 00 00 00 04 00  00 00 08 01 00 00 00 10  |................|
000000d0  1e fb 6b 54 0c 1d 55 40  a4 ad 4e f4 bf 17 b8 3a

Event 1 (yellow):
         T: 00  L: 00 00 00 04 V: 00 00 00 00          // SEQNUM 0
         T: 01  L: 00 00 00 04 V: 00 00 00 00          // PCRNUM 0
         T: 03  L: 00 00 00 19 V: nested               // Digest header
                 T: 04 L: 00 00 00 14 V: 00 … 00       // sha-1, 16 zero bytes
         T: 05  L 00 00 00 33 V   nested
                 T: 00 L: 00 00 00 04 V: 00 00 00 03   // Event Type 3
                 T: 01 L: 00 00 00 25 V: 53 70 … 00 00 // Event Content 37 bytes
```

```
Tlv_dump parses and reports these as:

SEQNUM 0 PCRNUM 0 DIGESTS SHA1 0000000000000000000000000000000000000000 PCCLIENT
 TYPE EV_NO_ACTION CONTENT 5370656332049442045766 56E7430330000000000000020002020 00
000040014000B00200000
SEQNUM 1 PCRNUM 0 DIGESTS SHA1 C42FEDAD268200CB1D15F97841C344E79DAE3320 SHA256 D
4720B4009438213B803568017F903093F6BEA8AB47D283DB32B6EABEDBBF155 PCCLIENT TYPE EV
_S_CRTM_VERSION CONTENT 1EFB6B540C1D5540A4AD4EF4BF17B83A SHA1 Matches content.
```

End of informative comment

5.1.8 CEL_MGT Content Layer

CEL_MGT is a simple record, defined directly in native TLV format. Figure 5 shows the format of the TIMESTAMP record.

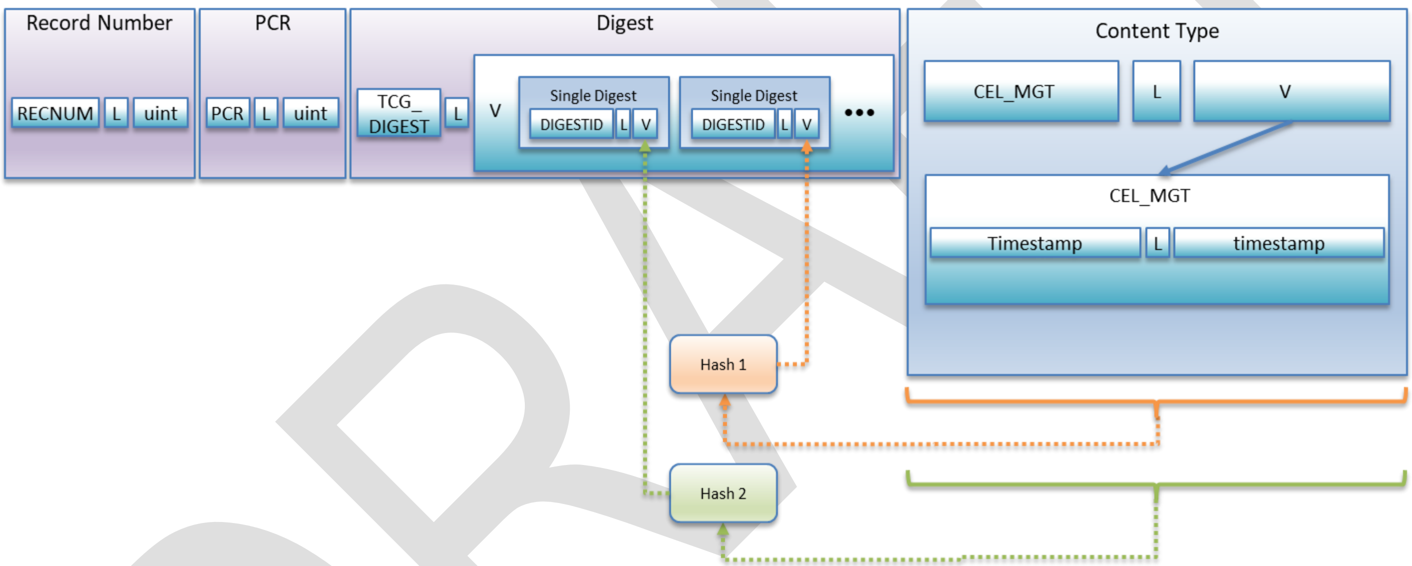

Figure 5 CEL Management content in CEL_TLV

The CEL_CONTENT_MGT tag is ‘4’, and the CEL_CONTENT_MGT_TIMESTAMP tag is ‘80’, as specified in section 4. The timestamp is in Unix/Linux epoch format (uint64 seconds from the epoch in big endian format.)

5.2 Canonical Event Log Encoding - Binary (CEL-CBOR)

Start of informative comment

RFC 7049 defines a Concise Binary Object Representation (CBOR) format. RFC 8610 defines Concise Data Definition Language (CDDL), which can be used to specify CBOR encodings. Section 5.2.2 includes a CDDL specification for the CEL-CBOR encoding. This encoding defines raw binary format for a Canonical Event Log that is compliant with this CEL specification, with content encapsulation using the same defined content types as in the TLV encodings in section 5.1. As the CDDL specification in 5.2.2 can be compiled into automated processing tools, informative examples and diagrams of the CEL-CBOR format are not included in this document.

Note that Section 5.2.2 contains a complete specification for CEL-CBOR and for the encapsulated PCCLIENT and IMA content layers. This specification is not authoritative over the TCG PC Client Platform Firmware Profile [2] and TPM Algorithm Registry [3] definitions, and the relevant values are included in this CEL-CBOR specification for convenience only. Section 5.2.2 contains comments on which sections are taken from which reference, and the indicated references should be consulted for authoritative values.

End of informative comment

5.2.1 Canonical Event Log Record Encoding for JSON (CEL-JSON)

The CDDL specification in Appendix A can also be used to generate a JSON encoding for CEL.

5.2.2 CEL-CBOR Encoding Layer CDDL Specification

```
tcg-canonical-event-log = [ + TPMS_EVENT: cel-record ]
recnum = 0
pcr = 1
nv_index = 2
digests = 3

cel-record = {
  recnum => uint,
  pcr-xor-nv-index,
  digests => TPML_DIGEST_VALUES,
  $$event-content, ; TPMU_EVENTCONTENT group choice
}

pcr-xor-nv-index //= ( pcr => uint .size 1 )
pcr-xor-nv-index //= ( nv_index => uint .size 1 )

TPML_DIGEST_VALUES = [ + digest: TPMT_HA ]
TPMT_HA = [
  TPMU_HA-hashAlg: digest-type,
  TPMU_HA-digest: digest-value,
]

digest-type = $TPM_ALG_ID ; manually added below for completeness
digest-value = bytes

CEL = 4
PCCLIENT_STD = 5
IMA_TEMPLATE = 7
IMA_TLV = 8

$$event-content //= ( CEL => $CEL-content )
$$event-content //= ( PCCLIENT_STD => PCCLIENT_STD-client-content )
$$event-content //= ( IMA_TEMPLATE => IMA_Template-content )
```



```
$$event-content //= ( IMA_TLV => IMA_TLV-content )

CEL_VERSION = 1
FIRMWARE_END = 2
CEL_TIMESTAMP = 80
STATE_TRANS = 81

TPMI_CELMGTTYPE = 0
TPMU_CELMGT = 1

$CEL-content /= {
  TPMI_CELMGTTYPE => CEL_VERSION,
  TPMU_CELMGT => [
    major: uint .size 2,
    minor: uint .size 2,
  ],
}

$CEL-content /= {
  TPMI_CELMGTTYPE => FIRMWARE_END,
}

$CEL-content /= {
  TPMI_CELMGTTYPE => CEL_TIMESTAMP,
  TPMU_CELMGT => uint .size 8,
}

$CEL-content /= {
  TPMI_CELMGTTYPE => STATE_TRANS,
  TPMU_CELMGT => { $$STATE_TRANS-content },
}

PCCLIENT_STD-client-content = [
  content-type: PCCLIENT_STD,
  event-type: $event-type,
  event-content: bytes,
]

IMA_Template-content = [
  content-type: IMA_TEMPLATE,
  template-name: text,
  template-value: bytes,
]

IMA_TLV_CONTENT_PATH = 0
IMA_TLV_CONTENT_DATAHASH = 1
IMA_TLV_CONTENT_DATASIG = 2
IMA_TLV_CONTENT_OWNER = 3
IMA_TLV_CONTENT_GROUP = 4
IMA_TLV_CONTENT_NODE = 5
IMA_TLV_CONTENT_TIMESTAMP = 6
IMA_TLV_CONTENT_LABEL = 7

IMA_TLV-content = [
  content-type: IMA_TLV,
  IMA-TLV: {
    ? IMA_TLV_CONTENT_PATH => bytes,
```



```
    ? IMA_TLV_CONTENT_DATAHASH => [ + bytes ],
    ? IMA_TLV_CONTENT_DATASIG => bytes,
    ? IMA_TLV_CONTENT_OWNER => bytes,
    ? IMA_TLV_CONTENT_GROUP => bytes,
    ? IMA_TLV_CONTENT_NODE => bytes,
    ? IMA_TLV_CONTENT_TIMESTAMP => bytes,
    ? IMA_TLV_CONTENT_LABEL => bytes,
  },
]

;; The following are from the PCCLIENT specification [2]

$event-type /= EV_PREBOOT_CERT
$event-type /= EV_POST_CODE
$event-type /= EV_UNUSED
$event-type /= EV_NO_ACTION
$event-type /= EV_SEPARATOR
$event-type /= EV_ACTION
$event-type /= EV_EVENT_TAG
$event-type /= EV_S_CRTM_CONTENTS
$event-type /= EV_S_CRTM_VERSION
$event-type /= EV_CPU_MICROCODE
$event-type /= EV_PLATFORM_CONFIG_FLAGS
$event-type /= EV_TABLE_OF_DEVICES
$event-type /= EV_COMPACT_HASH
$event-type /= EV_IPL
$event-type /= EV_IPL_PARTITION_DATA
$event-type /= EV_NONHOST_CODE
$event-type /= EV_NONHOST_CONFIG
$event-type /= EV_NONHOST_INFO
$event-type /= EV_OMIT_BOOT_DEVICE_EVENTS
$event-type /= EV_EFI_EVENT_BASE
$event-type /= EV_EFI_VARIABLE_DRIVER_CONFIG
$event-type /= EV_EFI_VARIABLE_BOOT
$event-type /= EV_EFI_BOOT_SERVICES_APPLICATION
$event-type /= EV_EFI_BOOT_SERVICES_DRIVER
$event-type /= EV_EFI_RUNTIME_SERVICES_DRIVER
$event-type /= EV_EFI_GPT_EVENT
$event-type /= EV_EFI_ACTION
$event-type /= EV_EFI_PLATFORM_FIRMWARE_BLOB
$event-type /= EV_EFI_HANDOFF_TABLES
$event-type /= EV_EFI_HCRTM_EVENT
$event-type /= EV_EFI_VARIABLE_AUTHORITY

EV_PREBOOT_CERT = 0
EV_POST_CODE = 1
EV_UNUSED = 2
EV_NO_ACTION = 3
EV_SEPARATOR = 4
EV_ACTION = 5
EV_EVENT_TAG = 6
EV_S_CRTM_CONTENTS = 7
EV_S_CRTM_VERSION = 8
EV_CPU_MICROCODE = 9
EV_PLATFORM_CONFIG_FLAGS = 10
EV_TABLE_OF_DEVICES = 11
EV_COMPACT_HASH = 12
```

```
EV_IPL = 13
EV_IPL_PARTITION_DATA = 14
EV_NONHOST_CODE = 15
EV_NONHOST_CONFIG = 16
EV_NONHOST_INFO = 17
EV_OMIT_BOOT_DEVICE_EVENTS = 18
EV_EFI_EVENT_BASE = 0x80000000
EV_EFI_VARIABLE_DRIVER_CONFIG = 0x80000001
EV_EFI_VARIABLE_BOOT = 0x80000002
EV_EFI_BOOT_SERVICES_APPLICATION = 0x80000003
EV_EFI_BOOT_SERVICES_DRIVER = 0x80000004
EV_EFI_RUNTIME_SERVICES_DRIVER = 0x80000005
EV_EFI_GPT_EVENT = 0x80000006
EV_EFI_ACTION = 0x80000007
EV_EFI_PLATFORM_FIRMWARE_BLOB = 0x80000008
EV_EFI_HANDOFF_TABLES = 0x80000009
EV_EFI_HCRTM_EVENT = 0x80000010
EV_EFI_VARIABLE_AUTHORITY = 0x800000E0

;; The following are from the TCG Algorithm Registry [3]
;; For convenience, the section below is manually copied from
;; tcg-algorithm-registry.cddl.  This document should be used
;; in conjunction with tcg-algorithm-registry.cddl instead.

$TPM_ALG_ID /= TPM_ALG_ERROR
$TPM_ALG_ID /= TPM_ALG_RSA
$TPM_ALG_ID /= TPM_ALG_TDES
$TPM_ALG_ID /= TPM_ALG_SHA
$TPM_ALG_ID /= TPM_ALG_SHA1
$TPM_ALG_ID /= TPM_ALG_HMAC
$TPM_ALG_ID /= TPM_ALG_AES
$TPM_ALG_ID /= TPM_ALG_MGF1
$TPM_ALG_ID /= TPM_ALG_KEYEDHASH
$TPM_ALG_ID /= TPM_ALG_XOR
$TPM_ALG_ID /= TPM_ALG_SHA256
$TPM_ALG_ID /= TPM_ALG_SHA384
$TPM_ALG_ID /= TPM_ALG_SHA512
$TPM_ALG_ID /= TPM_ALG_NULL
$TPM_ALG_ID /= TPM_ALG_SM3_256
$TPM_ALG_ID /= TPM_ALG_SM4
$TPM_ALG_ID /= TPM_ALG_RSASSA
$TPM_ALG_ID /= TPM_ALG_RSAES
$TPM_ALG_ID /= TPM_ALG_RSAPSS
$TPM_ALG_ID /= TPM_ALG_OAEP
$TPM_ALG_ID /= TPM_ALG_ECDSA
$TPM_ALG_ID /= TPM_ALG_ECDH
$TPM_ALG_ID /= TPM_ALG_ECDAA
$TPM_ALG_ID /= TPM_ALG_SM2
$TPM_ALG_ID /= TPM_ALG_ECSCHNORR
$TPM_ALG_ID /= TPM_ALG_ECMQV
$TPM_ALG_ID /= TPM_ALG_KDF1_SP800_56A
$TPM_ALG_ID /= TPM_ALG_KDF2
$TPM_ALG_ID /= TPM_ALG_KDF1_SP800_108
$TPM_ALG_ID /= TPM_ALG_ECC
$TPM_ALG_ID /= TPM_ALG_SYMCIPHER
$TPM_ALG_ID /= TPM_ALG_CAMELLIA
$TPM_ALG_ID /= TPM_ALG_SHA3_256
```

```
$TPM_ALG_ID /= TPM_ALG_SHA3_384
$TPM_ALG_ID /= TPM_ALG_SHA3_512
$TPM_ALG_ID /= TPM_ALG_CTR
$TPM_ALG_ID /= TPM_ALG_OFB
$TPM_ALG_ID /= TPM_ALG_CBC
$TPM_ALG_ID /= TPM_ALG_CFB
$TPM_ALG_ID /= TPM_ALG_ECB

TPM_ALG_ERROR = 0x0000
TPM_ALG_RSA = 0x0001
TPM_ALG_TDES = 0x0003
TPM_ALG_SHA = 0x0004
TPM_ALG_SHA1 = 0x0004
TPM_ALG_HMAC = 0x0005
TPM_ALG_AES = 0x0006
TPM_ALG_MGF1 = 0x0007
TPM_ALG_KEYEDHASH = 0x0008
TPM_ALG_XOR = 0x000A
TPM_ALG_SHA256 = 0x000B
TPM_ALG_SHA384 = 0x000C
TPM_ALG_SHA512 = 0x000D
TPM_ALG_NULL = 0x0010
TPM_ALG_SM3_256 = 0x0012
TPM_ALG_SM4 = 0x0013
TPM_ALG_RSASSA = 0x0014
TPM_ALG_RSAES = 0x0015
TPM_ALG_RSAPSS = 0x0016
TPM_ALG_OAEP = 0x0017
TPM_ALG_ECDSA = 0x0018
TPM_ALG_ECDH = 0x0019
TPM_ALG_ECDAA = 0x001A
TPM_ALG_SM2 = 0x001B
TPM_ALG_ECSCHNORR = 0x001C
TPM_ALG_ECMQV = 0x001D
TPM_ALG_KDF1_SP800_56A = 0x0020
TPM_ALG_KDF2 = 0x0021
TPM_ALG_KDF1_SP800_108 = 0x0022
TPM_ALG_ECC = 0x0023
TPM_ALG_SYMCIPHER = 0x0025
TPM_ALG_CAMELLIA = 0x0026
TPM_ALG_SHA3_256 = 0x0027
TPM_ALG_SHA3_384 = 0x0028
TPM_ALG_SHA3_512 = 0x0029
TPM_ALG_CTR = 0x0040
TPM_ALG_OFB = 0x0041
TPM_ALG_CBC = 0x0042
TPM_ALG_CFB = 0x0043
TPM_ALG_ECB = 0x0044
```


6 Defined types for all examples.

Start of informative comment

The following are C code definitions for the CEL-TLV and CEL-CBOR types used in the Examples in section 5.

```
/* TCG CEL Top Level Event Types */
#define CEL_SEQNUM                    0
#define CEL_PCR                       1
#define CEL_NV_INDEX                  2
#define CEL_DIGESTS                   3
#define CEL_CONTENT_MGT               4
#define CEL_CONTENT_PCCLIENT_STD      5
#define CEL_CONTENT_IMA_TEMPLATE      7
#define CEL_CONTENT_IMA_TLV           8

/* TCG TPM Digest Types */
#define TPM_ALG_SHA1                  4
#define TPM_ALT_SHA256                11

/* CEL_MGT types
#define CEL_CONTENT_MGT_TIMESTAMP     80
#define CEL_CONTENT_MGT_KEXEC         81

/* IMA-TLV Specific Content Types */
#define IMA_TLV_CONTENT_PATH          0
#define IMA_TLV_CONTENT_DATAHASH      1
#define IMA_TLV_CONTENT_DATASIG       2
#define IMA_TLV_CONTENT_OWNER         3
#define IMA_TLV_CONTENT_GROUP         4
#define IMA_TLV_CONTENT_MODE          5
#define IMA_TLV_CONTENT_TIMESTAMP     6
#define IMA_TLV_CONTENT_LABEL         7

/* IMA_TEMPLATE Specific Content Types */
#define IMA_TEMPLATE_CONTENT_NAME     0
#define IMA_TEMPLATE_CONTENT_VALUE    1

/* PCCLIENT_STD content types */
#define PCCLIENT_EVENT_TYPE           0
#define PCCLIENT_EVENT_CONTENT        1
```

End of informative comment

