# Secure Launch Specification

**Version:** 0.5.0

**Authors:**

**Daniel P. Smith** (Apertus Solutions) **Ross Philipson** (Oracle) **Krystian Hebel** (3mdeb)

# Table of Contents

# 1 Introduction

Like many cross-platform capabilities implemented by different manufacturers, each vendor may have their own unique way of implementing any one of these capabilities. Dynamic Launch is no different in this manner. To handle this, the TrenchBoot project is striving to provide a unified approach to using Dynamic Launch across different platforms. One aspect of Dynamic Launch that varies across platforms is the launch related data. For example, on Intel there is a series of Intel defined configuration data structures that must be present, as well as a user-defined data structure. Whereas on AMD, only the layout of an AMD Secure Loader Block (SLB) is defined and thus, any configuration structures are SLB implementation specific.

To provide a unified experience in passing configuration and other meta-data to a TrenchBoot Secure Launch entry point for a kernel, this specification provides the platform-agnostic Secure Launch Resource Table (SLRT) along with details on how it will be implemented for each platform supported.

# 2 Terminology

## 2.1 Requirements Language

The key words **"MUST"**, **"MUST NOT"**, **"REQUIRED"**, **"SHALL"**, **"SHALL NOT"**, **"SHOULD"**, **"SHOULD NOT"**, **"RECOMMENDED"**, **"MAY"**, and **"OPTIONAL"** in this document are to be interpreted as described in RFC 2119.

## 2.2 Acronyms

**DCE:** Dynamic Configuration Environment (*eg. Intel ACM/AMD SLB*)
**DLE:** Dynamic Launch Event (*eg. Intel GETSEC[SENTER]/AMD SKINIT*)
**DLME:** Dynamic Launch Measured Environment (*eg. Operating System/Hypervisor*)
**DRTM:** Dynamic Root of Trust Measurement

# 3 Secure Launch Architecture

## 3.1 Secure Launch aware Bootloader

A bootloader is Secure Launch aware if it understands how to load or provide a DLE (Dynamic Launch Event) Handler and configure an SLRT.

> ***Note***
>
> Throughout this specification the term bootloader will be used to generically refer to the sofware responsible for loading the OS. This may be an x86 bootloader such as GRUB, an embedded system bootloader such as Das U-Boot, or an arbitrary UEFI OS Loader.

### *3.1.1 Bootloader Types*

A Secure Launch aware bootloader can be categorized as one of three types,

**monolithic:** This is a bootloader that contains the DLE Handler.

**initializing:** This is a bootloader that loads an external DLE Handler and executes a supplemental bootloader.

**supplemental:** This is a bootloader that is capable of calling a DLE Handler.

## 3.2 Dynamic Launch Event Handler

The DLE Handler consumes the SLRT and invokes the platform's DLE. The Secure Launch specification seeks to standardize the invocation interface for the DLE Handler to be implemented by each platform supported by TrenchBoot. The specification provides a well-defined interface for DLE Handler and bootloader implementors to follow to ensure interoperability between implementations.

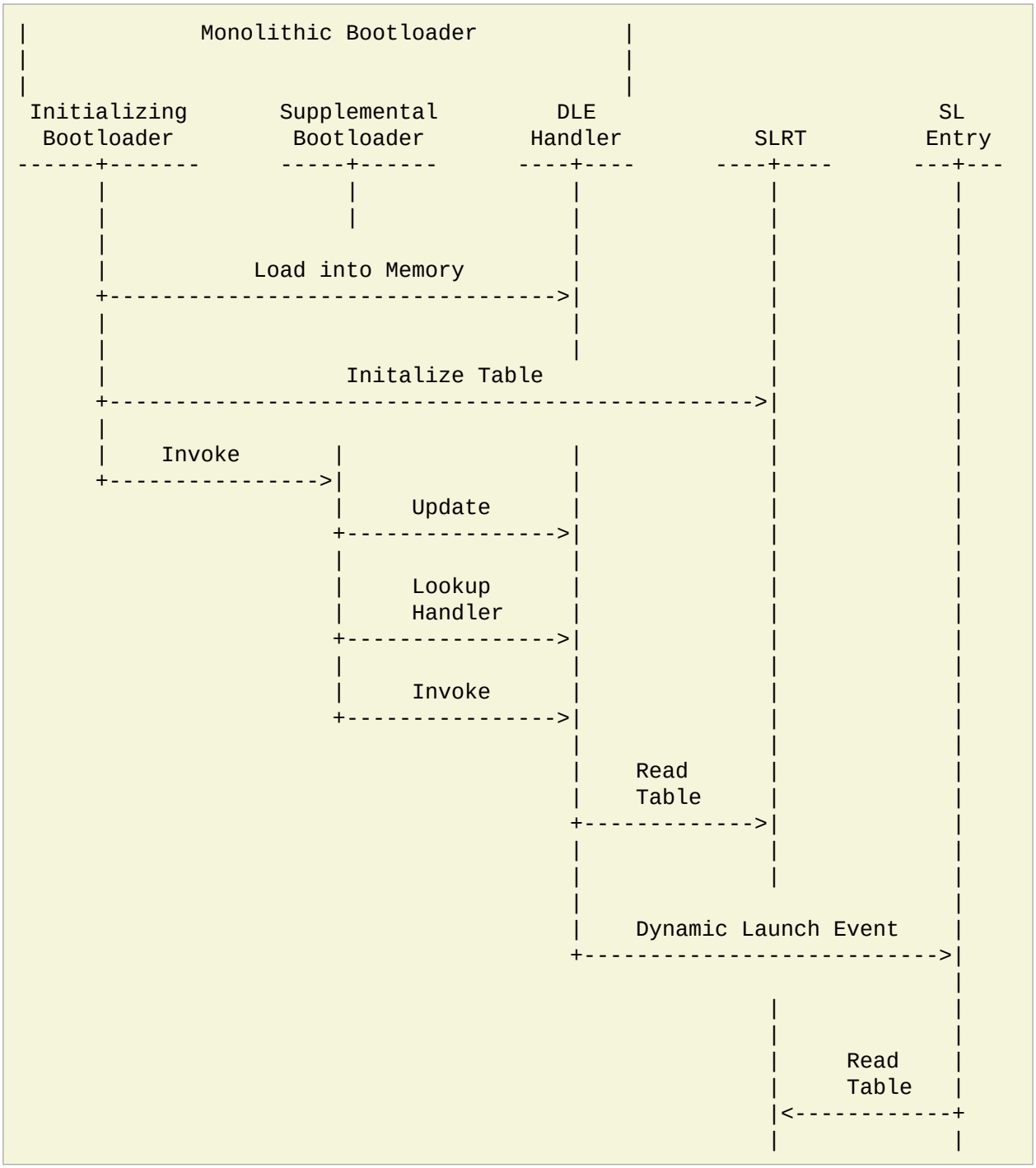
## 3.3 Secure Launch Resource Table

The SLRT is a platform-agnostic, standard format for providing information to the DLE Handler and for passing D-RTM relevant information across the DLE to be available for use by the DCE and the DLME. The table **SHALL** be initialized by a bootloader and any subsequent bootloaders in the boot chain **MAY** append entries to the table. While the table is designed to be implementation agnostic, the reality of D-RTM hardware will drive a difference in the construction and setup of the table.

## 3.4 Secure Launch Entry

The SL Entry (Secure Launch Entry) is a kernel entry point for the DLME that is aware of the intricacies of the platform's D-RTM implementation. Its interface is dictated by the platform's Dynamic Launch implementation. The SL Entry is responsible for bringing the system up from its Dynamic Launch state to a suitable state for transitioning control to the DLME kernel. Before transferring control to the standard code path, the SL Entry **SHALL** use the SLRT to establish an integrity assessment of the platform.

## 3.5 Sequence




# 4 Secure Launch Interfaces

There are two interfaces to be defined here, the DLE Handler Specifications and the SLRT Specification.

## 4.1 DLE Handler Specification

The DLE Handler Specification defines the invocation interface for the DLE Handler.

### *4.1.1 Platform Requirements*

**1** - x86 Platforms
**1.1** - The DLE Handler **MAY** be invoked with the CPU in either 32bit protected mode or 64bit long mode
**1.2** - The SLRT **SHALL** be passed to the DLE Handler in the EDI/RDI CPU register
**1.3** - All other registers besides EDI/RDI are not guarenteed
**1.4** - The invoking code **SHALL** use a long jump to the DLE Handler
**1.5** - The DLE Handler **SHALL NOT** return control on error

**2** - Arm Platforms
**2.1** - *Reserved*

## 4.2 SLRT Specification

This specification details the construction of the SLRT, and how that table will be passed to a Secure Launch entry point for each supported hardware platform.

The SLRT **SHALL** be initialized by a bootloader and any subsequent bootloaders in the boot chain **MAY** append entries to the table. While the table is designed to be agnostic, the reality of DRTM hardware will drive differences in the construction and populating the table. Outlined here are a set of common, general requirements that all platforms should be able to meet. The supplemental sections will cover any idiosyncrasies for the various platforms and environments supported.

## 4.3 Platform Requirements

**1** - General Requirements
**1.1** - The SLRT **MUST** begin with the magic value *0x4452544d*.
**1.2** - A properly formatted SLRT **SHALL** consist of a table header, zero or more table entries, and an end entry.
**1.3** - The SLRT **SHOULD** be in contiguous physical memory.
**1.3.1** - A preallocated, fixed size table is **OPTIONAL** through the use of the *max_size* field.
**1.4** - The SLRT **SHALL** be aligned to a four-byte boundary.

**2** - UEFI Environmnents
**2.1** - The SLRT **SHALL** be registed in the UEFI SystemTable.
**2.1.2** - The GUID for registering the table **MUST BE** *877a9b2a-0385-45d1-a034-9dac9c9e565f*.
**2.2** - All UEFI OS Loaders **SHALL** record an EFI Config record with an UEFI Config Entry for each measureable configuration action or information provide to the DLME kernel.
**2.3** - The UEFI OS Loader is responsible for calling ExitBootServices() **SHALL** be responsible for calling the DLE Handler.
**2.3.1** - The address for the SLRT **MUST** be passed via an architecture specific register.

**3** - x86 Platforms
**3.1** - The SLRT **MUST** be constructed within the 4G boundary.
**3.1.1** - On Intel TXT platforms the location of the SLRT **SHALL** be stored in the OS2MLE structure, see Appendix B
**3.1.2** - On AMD platforms the location of the SLRT **SHALL** be stored in the Secure Kernel Loader (SKL) configuration table

**4** - Arm Platforms
**4.1** - *Reserved*

## 4.4 SLRT Structure

The SLRT is constructed from a header at the beginning, followed by a list of Tag-Length-Value (TLV) entries. Provided below is a description of the header and the possible entry types that may be found in the table. For general portability, the definitions of each entry contain a representative C structure.

### *4.4.1 Versioning Scheme*

The SLRT supports revisioning at two depths, at the table level and at the individual entry level. The table level revision is found in the SLRT header and conveys the format of the SLRT header as well as what SLRT entries may be appear in the table. The entry level revisioning is for those that are expected may need future expansion.

#### *4.4.1.1 Revision Table*

This table contains the list of versions for this revision of the specification.

| | Revision |
|---|---|
| SLR Table | 0x01 |
| D-RTM Policy Entry | 0x01 |
| UEFI Config Entry | 0x01 |

### *4.4.2 Core Entries*

The core table entries are the set of entries that are platform and architecture agnostic.

#### *4.4.2.1 SLRT Header*

The SLRT Header **SHALL** be located at the beginning of the table and provides the core information regarding the construction of the table.

**magic:** SLRT identifier, expected value is *0x4452544d*.

**revision:** Indentifies the SLRT specification revision used.

**architecture:** Identifies the platform architecture and DL implementation.

**size:** Total size of the table

**max_size:** The maximum size of the memory block.

```
1 struct slr_table {
2     u32 magic;
3     u16 revision;
4     u16 architecture;
5     u32 size;
6     u32 max_size;
7     /* entries[] */
8 };
```

#### *4.4.2.2 SLRT Entry Header*

The SLRT is a TLV list requiring every entry to have an identifying tag and the size of the entry. The SLRT Entry Header provides that representation and is present at the beginning of every entry.

**tag:** Entry identifier.

**size:** Size of the entry.

```
1 struct slr_entry_hdr {
2     u16 tag;
3     u16 size;
4 };
```

#### 4.4.2.2.1 SLRT Entry Tag Types

The list of valid entry tags.

```
 1 #define SLR_ENTRY_INVALID           0x0000
 2 #define SLR_ENTRY_DL_INFO           0x0001
 3 #define SLR_ENTRY_LOG_INFO          0x0002
 4 #define SLR_ENTRY_DRTM_POLICY       0x0003
 5 #define SLR_ENTRY_INTEL_INFO        0x0004
 6 #define SLR_ENTRY_AMD_INFO          0x0005
 7 #define SLR_ENTRY_ARM_INFO          0x0006
 8 #define SLR_ENTRY_UEFI_INFO         0x0007
 9 #define SLR_ENTRY_UEFI_CONFIG       0x0008
10 #define SLR_ENTRY_END               0xffff
```

### 4.4.2.3 Dynamic Launch Configuration

**REQUIRED:** This entry **MUST** be present.

This is the main configuration entry, providing the necessary information to invoke the DLE Handler and for the DLE Handler to invoke the DL.

| | |
|---|---|
| **tag:** | SLR_ENTRY_DL_INFO |
| **bl_context:** | Allows the bootloader to provide a reference to a context object. |
| **dl_handler:** | The address to the entry point for the DLE Handler. |
| **dce_base:** | The base address where the DCE is located. |
| **dce_size:** | The size of the DCE. |
| **dlme_entry:** | The address for the entry point of the DLME. |

```
1 struct slr_entry_dl_info {
2     struct slr_entry_hdr hdr;
3     struct slr_bl_context bl_context;
4     u64 dl_handler;
5     u64 dce_base;
6     u32 dce_size;
7     u64 dlme_entry;
8 };
```

#### 4.4.2.3.1 Boot Loader Context

There may be a situation where the bootloader will need to leave a context object containing platform specific information or helper callbacks for the DLE Handler to use.

> ***Note***
>
> It is out-of-scope for this specification, but it is advised that if a platform or a bootloader requires the use of a context object, they should standardize their context object to enable independent DLE Handlers

**bootloader:** An identifier the bootloader should use for ident and version.

**context:** Address of the context object.

```
1 struct slr_bl_context {
2     u16 bootloader;
3     u16 reserved;
4     u64 context;
5 };
```

#### *4.4.2.4 DRTM TPM Event Log*

**REQUIRED:** This entry **MUST** be present.

This entry describes where and what type of TPM event log should be used.

> ***Note***
>
> On most platforms, the TCG UEFI TPM event log format should be used. The ability to specify alternatives is to support older platforms that are not aware of the modern event log format or can support multiple formats.

**tag:** SLR_ENTRY_LOG_INFO

**format:** The type of TPM event log format to use.

**addr:** The base address where the log should reside.

**size:** The size allocated for the log.

```
1 struct slr_entry_log_info {
2     struct slr_entry_hdr hdr;
3     u16 format;
4     u16 reserved;
5     u64 addr;
6     u32 size;
7 };
```

#### *4.4.2.5 D-RTM Measurement Policy*

**REQUIRED:** This entry **MUST** be present and have the required entries.

The measurement policy is for conveying to the SL Entry on what it should measure, where that entity is located, which PCR the measurement should be stored, and how the event should be identified in the TPM event log.

> ***Warning***
>
> The SL Entry **SHALL** fail if it determines an invalid policy is present.

**tag:** SLR_ENTRY_ENTRY_POLICY

**revision:** A revision field to identify the version of policy being used.

**nr_entries:** The total number of policy entries available.

```
1 struct slr_entry_policy {
2     struct slr_entry_hdr hdr;
3     u16 revision;
4     u16 nr_entries;
5     /* policy_entries[] */
6 };
```

#### 4.4.2.5.1 DRTM Policy Entry

A policy entry represents an entity that the SL Entry is being requested to measure. As an SL Entry is able to measure an attribute of the launch environment, that attribute will be published as an entity type. A generic "unspecified" entity type is also available for measuring a range of memory.

**pcr:** PCR to store the measurement.

**entity_type:** Identifies the entity type of the entry.

**flags:** Flag field to store state for this entry.

**entity:** The address to measure.

**size:** The size of entity if not flagged as implicit.

**evt_info:** Label to be recorded in TPM Event Log.

```
1 struct slr_policy_entry {
2     u16 pcr;
3     u16 entity_type;
4     u16 flags;
5     u16 reserved;
6     u64 entity;
7     u64 size;
8     char evt_info[TPM_EVENT_INFO_LENGTH];
9 };
```

##### 4.4.2.5.1.1 D-RTM Policy Entry Entity Types

The list of valid entity types for D-RTM Policy entries.

```
1 #define SLR_ET_UNSPECIFIED  0x0000
2 #define SLR_ET_SLRT         0x0001
3 #define SLR_ET_BOOT_PARAMS  0x0002
4 #define SLR_ET_SETUP_DATA   0x0003
5 #define SLR_ET_CMDLINE      0x0004
6 #define SLR_ET_UEFI_MEMMAP  0x0005
7 #define SLR_ET_RAMDISK      0x0006
8 #define SLR_ET_TXT_OS2MLE   0x0010
9 #define SLR_ET_UNUSED       0xffff
```

##### 4.4.2.5.1.2 D-RTM Policy Entry Flags

The list of valid flags for D-RTM Policy entries.

```
1 #define SLR_POLICY_FLAG_MEASURED    0x1
2 #define SLR_POLICY_IMPLICIT_SIZE    0x2
```

### 4.4.3 Intel TXT Platforms

When on Intel platforms specific information needs to be conveyed to Secure Launch.

#### 4.4.3.1 Intel TXT Info

**REQUIRED:** This entry **MUST** be present on Intel platforms.

Intel TXT requires for the pre-launch environment to pass MSR and MTRR state across to the post-launch environment.

**tag:** SLR_ENTRY_INTEL_INFO

**saved_misc_enable_msr:** Saved MSR values

**saved_bsp_mtrrs:** Saved BSP MTRRs

```
1 struct slr_entry_intel_info {
2     struct slr_entry_hdr hdr;
3     u64 saved_misc_enable_msr;
4     struct txt_mtrr_state saved_bsp_mtrrs;
5 };
```

##### 4.4.3.1.1 Saved MTRR State

`struct slr_txt_mtrr_state`

**default_mem_type:** The default memory type for regions not covered by an MTRR

**mtrr_vcnt:** Number of variable MTRR pairs in the mtrr_vcnt array

**mtrr_pair:** Array of variable MTRR pairs to restore post launch

`struct slr_txt_mtrr_pair`

**mtrr_physbase:** Physical base address for variable MTRR

**mtrr_physmask:** Physical mask for the variable MTRR

```
 1 struct slr_txt_mtrr_pair {
 2     u64 mtrr_physbase;
 3     u64 mtrr_physmask;
 4 };
 5
 6 struct slr_txt_mtrr_state {
 7     u64 default_mem_type;
 8     u64 mtrr_vcnt;
 9     struct txt_mtrr_pair mtrr_pair[TXT_VARIABLE_MTRRS_LENGTH];
10 };
```

### 4.4.4 AMD Secure Launch Platforms

#### 4.4.4.1 AMD SKINIT Info

A placeholder for info specific to AMD SKINIT.

```
1 struct slr_entry_amd_info {
2     struct slr_entry_hdr hdr;
3 };
```

### 4.4.5 ARM DRTM Environments

#### 4.4.5.1 ARM Info

A placeholder for info specific to ARM D-RTM environments.

```
1 struct slr_entry_arm_info {
2     struct slr_entry_hdr hdr;
3 };
```

### 4.4.6 UEFI Environments

To support UEFI bootloaders that may do additional configuration of the Secure Launch kernel, the UEFI SLRT entries provide a means to convey any operational configurations they may have done.

#### 4.4.6.1 UEFI Info

A placeholder for info specific to UEFI environments.

```
1 struct slr_entry_uefi_info {
2     struct slr_entry_hdr hdr;
3 };
```

#### 4.4.6.2 UEFI Config

**OPTIONAL:** This entry **SHOULD** be present on UEFI systems.

This entry can be considered a D-RTM measurement policy for UEFI. It will declare that the UEFI bootloader has made configurations changes that should be measured.

**tag:** SLR_ENTRY_UEFI_CONFIG
**revision:** A revision field to identify the version of config being used.
**nr_entries:** The total number of configuration entries present.

```
1 struct slr_entry_uefi_config {
2     struct slr_entry_hdr hdr;
3     u16 revision;
4     u16 nr_entries;
5     /* slr_uefi_cfg_entries[] */
6 };
```

##### 4.4.6.2.1 UEFI Config Entry

**OPTIONAL:** This entry **SHOULD** be present on UEFI systems.

A config entry represents an entity that the UEFI bootloader is requesting to be measured.

**pcr:** PCR to store the measurement.
**cfg:** The address or value to measure.
**size:** The size to measure.

**evt_info:** Label to be recorded in TPM Event Log.

```
1 struct slr_uefi_cfg_entry {
2     u16 pcr;
3     u16 reserved;
4     u64 cfg; /* address or value */
5     u32 size;
6     char evt_info[TPM_EVENT_INFO_LENGTH];
7 } __packed;
```

# 5 Appendix A: Measuring the DRTM Policy

While the D-RTM TPM event log is itself proof of the D-RTM policy used by Secure Launch, there may be motivation for the policy itself to be incorporated into the measurement chain. While this section does not address the possible motivations or the validity of those motivations, a possible use of the policy measurement can be used as the value to cap the D-RTM PCRs. Regardless of motivations, this appendix is to provide guidance on how the policy might be measured in a meaningful way.

## 5.1 TPM Extend Operation

For clarity, the extend operation, denoted here on out as E(), is an order preserving, recursive, mapping function. Consider any hash function, denoted as H(), the extend operation, is defined as:

Given,
    0 = sizeof(H) bytes of 0
    Objs = [ Obj_0, ..., Obj_n ]
Then,
    E(Objs) = {
        E_0 = H( 0 | H(Obj_0) )
        E_n = H( E_(n-1) | H(Obj_n) )
    }

## 5.2 Measuring the Policy

Measuring the policy is not as simple as hashing the block of memory containing the policy. This will not work as the policy may contain memory addresses that have the potential to change on the next launch of the system. As a result there is a potential for the next launch not to have the same memory addresses in the policy, which would in turn render a different measurement.

To make a meaningful measurement of the policy, the measurement must capture what entities were to be measured and what order they were to be measured. To capture the first half, the measurement must contain enough of a DRTM policy entry to capture its uniqueness. Considering DRTM Policy Entry above, a combination of PCR, Entity Type, and Event Info yields a unique identity for the entry. To capture ordering of the policy events, the extend operation can be used to render a unique value that reflects the order of the policy events.

Using this logic, the resulting operation to measure the policy would be as:

Given,
    Entry_n = PCR_n | EntityType_n | EventInfo_n
    Policy = [ Entry_0, ..., Entry_n ]
Then,
    M_policy = E(Policy)

The result, *M_policy*, will be a hash of the policy that can then be extended into one, or more if using as a cap value, PCR(s).

# 6 Appendix B: Intel TXT OS2MLE

The Intel TXT specification[1] provides a provision for the pre-launch environment to pass information to the post-launch environment. The specification does not define this structure, leaving that to the implementation, but provides an allocation for it in the TXT Heap definition. This area is referred to as the OS2MLE structure.

The OS2MLE structure for Secure Launch is defined as follows,

**version:** Revision of the os2mle table

**slrt:** Pointer to the SLRT

**mle_scratch:** Scratch area for use by SL Entry early code

```
1 struct os2mle {
2     u32 version;
3     struct slr_table *slrt;
4     u8 mle_scratch[64];
5 }
```

[1] https://www.intel.com/content/www/us/en/content-details/315168/intel-trusted-execution-technology-intel-txt-software-development-guide.html?wapkw=txt