TCG PC Client Platform
Reset Attack Mitigation Specification

Family "2.0"

Version 1.10 Revision 17

January 21, 2019

Published

Contact: admin@trustedcomputinggroup.org

TCG Published

TCG

Disclaimers, Notices, and License Terms

Change History

Revision	Date	Description
1.0	January 5, 2008	Initial Revision
1.10	January 21, 2019	This Revision

Contents

List of Figures

List of Tables

Corrections and Comments

Comments may be sent to: techquestions@trustedcomputinggroup.org

TPM Dependency and Requirements

1. The TPM used for Host Platforms claiming adherence to this specification SHALL be compliant with the TCG PC Client Platform TPM Profile for TPM 2.0 Version 1.00, Revision 1.00 or later.
2. The Platform Class for platforms claiming adherence to this specification SHALL be registered with the TCG administrator.
3. Host Platforms claiming adherence to this specification SHALL be compliant with the TCG ACPI Specification Family 1.2 and 2.0, Revision 00.37 or later.

1 Introduction and Concepts

Theory of Operation

When a platform reboots or shuts down, the contents of volatile memory (RAM) are not immediately lost. Without an electric charge to maintain the data in memory, the data will begin to decay. During this period, there is a short timeframe during which an attacker can turn the platform back on to boot into a program that dumps the contents of memory. Encryption keys and other secrets can be easily compromised through this method.

Host Platform Reset threats to the S-CRTM can be mitigated by a Platform Firmware-initiated system memory operation that overwrites system memory on the next platform reboot. The Platform Firmware must overwrite memory with information unrelated to the secrets in memory that may be exposed to an attacker after a Host Platform reset; zeroing memory is one example of an effective memory overwrite operation.

The Platform Firmware is not required to initiate and complete the memory overwrite operation on every platform reboot, but is required to initiate and complete a memory overwrite operation every time it is signaled to do so by the OS. In this specification, a bit setting in Host Platform non-volatile memory, which persists across all types of Host Platform Resets, is called the Memory Overwrite Request (MOR) bit. Figure 1 and Figure 2, which are part of this Informative comment, show how Platform Firmware, the Bootloader, and the OS use the MOR bit to communicate with each other across all types of Host Platform Reset events.

A general description of the scheme is that after any type of Host Platform Reset event (except for a CPU-only reset that is used by some chipsets to turn off a CPU feature without re-setting other Host Platform components), if signaled to do so by the OS, the POST Platform Firmware must, prior to executing any non-Platform Firmware code, overwrite system memory.

Figure 1 shows the sequence where Platform Firmware reads the MOR bit before Platform Firmware executes any Option ROM code, the Bootloader code sets the MOR bit before the Bootloader code puts any secrets in the clear in system memory, and the OS clears the MOR bit across a Host Platform Reset event that includes a controlled OS shutdown; in this case the Platform Firmware code does not initiate a memory overwrite operation during the next Host Platform boot operation.

In Figure 2, a controlled OS shutdown is not part of the Host Platform Reset event, which is a potential attack. Comparing Figure 2 with Figure 1 shows that because the OS shutdown code was not executed, the OS did not clear the non-volatile MOR bit. When Platform Firmware code executes following a reset without OS shutdown or an incomplete shutdown, Platform Firmware code reads a '1' from the MOR bit and initiates a vendor-specific method that overwrites all of system memory and the processor caches. When that memory clear operation completes successfully, Platform Firmware clears the MOR bit. It then continues the boot process as in the orderly shutdown case, because all secrets have been cleared from memory.

Note that in Figure 2, although the box labeled “Memory Overwrite method” is shown outside the Platform Firmware code execution arrow, this does not mean execution of this method affects PCR contents; the code that either initiates a hardware-assisted method of overwriting memory or the code that overwrites memory without hardware assistance is a block of code in Platform Firmware, which is already measured into a PCR.

In Figure 1, and in the part of Figure 2 that shows a controlled OS shutdown, the Bootloader code writes a ‘1’ to the MOR flag before it has any secrets in system memory to protect. Figure 1 also shows that subsequently, as part of the controlled OS shutdown, the OS writes a ‘0’ to the MOR bit when there are no more secrets in memory to protect. Between these two OS-initiated write events to the MOR bit, the OS protects the secrets in system memory.

Because clearing memory may be required for reasons and platform functions not related to the MOR bit or purposes and threats associated with this specification or any other TCG operation, nothing in this specification prohibits any memory clear operation.

Scope, Security and Trust Assumptions

The scope of this specification applies only to the contents of memory under control of the Operating System within the Static RTM. However, it’s possible (and even likely) that memory under control of a D-RTM is also cleared as a result of the methods described in this specification.

The attacks mitigated by the methods in this specification are limited to simple rebooting of the system. An example of an attack to be mitigated is a stolen platform which is in a suspended state. After several unsuccessful attempts to guess the OS lock password, the attacker forces the platform to reboot without shutting down the OS and reboots to a CD containing an attack OS from which the attacker expects to read the contents of memory. The methods in this specification are not intended to protect against active physical attacks beyond the scope of the above scenario.

All secrets capable of being cleared by the methods in this specification are exposed to Ring 0, therefore, the protections to invoke and control these methods are also exposed to Ring 0. For this reason, this specification makes a fundamental assumption that the Operating System protects Ring 0, and any violation of those protections renders the methods discussed in this specification useless.

The security assurance level provided by the MOR bit mechanism is strengthened if the data to be protected is sealed using PCR [0].

Adding the functionality that is in this specification to the Platform Firmware does not open the Platform Firmware up to additional attacks.

If an attacker with physical access crashes the OS and if Platform Firmware has not been replaced since the last boot cycle, the secrets in system memory are still protected. If a Host Platform has an OS environment for Platform Firmware update, that platform may be at risk. This specification assumes the Host Platform manufacturer tightly controls Platform Firmware update. The requirements for protecting the Platform Firmware update process are stated in the relevant PC Client Specifications.

Functionality

On systems with conventional BIOS, the Bootloader code uses an INT 1Ah function offered by the Platform Firmware code to set the MOR bit before the Bootloader code puts any secrets in the clear in system memory. This INT 1Ah function is defined in Section 5.

For a UEFI boot, the EFI OS loader uses the MemoryOverwriteRequestControl EFI variable to set the MOR bit prior to the loader putting any secrets in the clear in system memory. This variable is defined in Section 4.

OS code uses a function in the ACPI _DSM control method in the TPM 1.2 ACPI device object to clear the MOR bit as part of a controlled OS shutdown. This _DSM control method function is defined in Section 5. The requirements in that section are based on the industry-standard ACPI 3.0 specification. The Platform Firmware persists the result of the OS calling this _DSM method function across Host Platform reboots by using a vendor-specific bit in a non-volatile storage location on the Host Platform.

On a UEFI system, the MemoryOverwriteRequestControl EFI variable described in Section 4 can be updated to clear the MOR bit after secrets have been removed from memory.

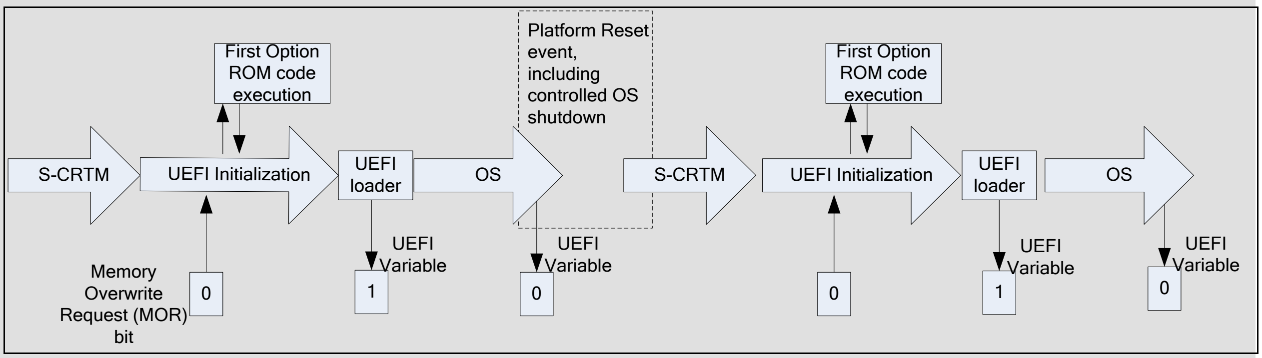

Figure 1 UEFI Platform Boot Cycles: With Complete OS Shutdown

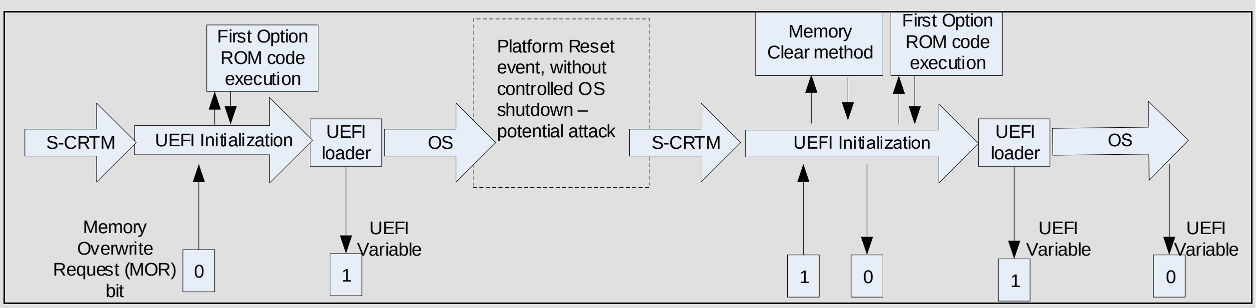

Figure 2 UEFI Platform Boot Cycles: Without Complete OS Shutdown

2 Requirements

This section contains all the mandatory requirements for this specification for clearing memory upon unexpected resets and reboots.

2.1 General Requirements

During a transition from S1 to S3, the operating system does not rely on protections provided by the MOR bit. The Platform Firmware, therefore, takes no action entering or leaving any of these operational states. When entering S4 and S5, the operating system depends on the protections provided by the MOR bit and therefore the Platform Firmware is expected to honor the MOR bit. Platform Firmware should detect and act on the MOR bit upon resuming from these operational states.

Item 3.b below requires the Platform Firmware to attempt to detect any potential tampering with the MOR bit. Tampering of the MOR bit could cause the Platform Firmware, upon reset, to ignore a necessary memory clear operation.

This specification defines platform behavior for scenarios where TPM protected secrets reside in memory. Examples for situations where clearing memory is not necessary are: the manufacturing floor, prior to OS installation, or if the OS does not make use of the TPM's boot time data protection capabilities.

1. Platform Firmware MUST support reading and writing the Memory Overwrite Request (MOR) bit to and from non-volatile storage on the Host Platform
2. To enable Bootloader code to communicate MOR bit settings to Platform Firmware, Platform Firmware MUST support the EFI Variables MemoryOverwriteRequestControl and MemoryOverwriteRequestControlLock.
3. If there is a TPM present and either of the following conditions occur, the Platform Firmware MUST initiate the process that clears all system memory and the processor caches:
    a. The Platform Firmware detects the MOR bit is set, or
    b. The Platform Firmware detects any reliability or integrity issue with NVM on the Host Platform.
4. The MOR request (i.e. checking of MOR bit and memory clear operation) SHOULD be performed before control is transferred outside of the S-CRTM, and MUST be performed before Bootloader, option ROM, DXE driver or any other 3rd party code can be executed.
5. The Platform Firmware MAY perform a memory clear operation for reasons unrelated to the MOR bit or for purposes and threats not associated with this specification.

2.2 Memory Overwrite Request Optimizations

To ensure that the memory overwrite process is performed as efficiently as possible, system builders should be aware of additional design considerations. If the MOR request is performed too early in the boot process, the system may not be able to take advantage of the full speed of memory. This may greatly increase the time required to overwrite memory and can result in slower boot times and increased user confusion. Ideally, the MOR request should be performed as soon as memory has been initialized and can be overwritten with minimal clock cycles per byte.

UEFI platform firmware can use known art to ensure that flash wear-leveling occurs in the UEFI variable store since this MemoryOverwriteRequestControl variable will be written twice per platform boot.

2.3 Auto Detection of Clean Static RTM Shutdown

Some Operating Systems may not clear the MOR bit prior to shutting down. This may cause the Platform Firmware to always perform a memory clear operation. While not a security concern, this will cause unnecessary delays in the platform's boot process. Operating Systems that do not clear the MOR bit upon a clean shutdown should provide an option to allow the platform owner to opt-out of the protections provided by the MOR. These operating systems may also indicate this potential behavior by clearing the DisableAutoDetect bit in the MemoryOverwriteAction_BitValue parameter. When this bit is zero (clear), the Platform Firmware can detect a clean shutdown of the Operating System and clear the flag itself.

There are various target shutdown operational states and certain conventional steps an Operating System takes when transitioning to those states. If allowed by the DisableAutoDetect bit, the Platform Firmware may detect an orderly Operating System shutdown. Known operational state transitions and their notifications are identified in the normative sections below. The most common method for clearing the MOR upon detecting one of these notifications is the use of SMI but no particular method is mandated by this specification.

Operating Systems that always clear the MOR bit upon a clean shutdown (i.e., they will always call the MOR interface) will set the DisableAutoDetect bit to the value 1 indicating to the Platform Firmware that it should not automatically detect the Operating System's shutdown.

Operating Systems that allow the Platform Firmware to autodetect a clean shutdown must ensure that secrets are cleared from memory prior to any notification event listed below.

The descriptions below are provided as example implementation to auto-detect an orderly shutdown of the Operating System.

It is permissible for system firmware to be implemented such that it automatically clears MOR on detection of an orderly shutdown of the OS. Determination of an orderly shutdown of the OS is OS and firmware specific.

3 MemoryOverwriteAction_BitValue Values

Values for MemoryOverwriteAction_BitValue Parameter which are common to both 215 Conventional Platform Firmware and UEFI functions are defined in Table 1 below.

Table 1 Variable Layout

Mnemonic	Bit Offset	Bit Length	Description
ClearMemory	0	1	0 = Firmware MUST clear the MOR bit 1 = Firmware MUST set the MOR bit Note: Based on the MOR bit value Firmware performs actions on the next reboot.
Reserved	1	3	Reserved (currently unused, Caller MUST set all to 0s)
DisableAutoDetect	4	1	0 = Firmware MAY autodetect a clean shutdown of the Static RTM OS. See Section 2.3 for details. 1 = Firmware MUST NOT autodetect a clean shutdown of the Static RTM OS
Reserved	5	3	Reserved (currently unused). Caller MUST set all to 0.

4 UEFI Interface

UEFI uses variables rather than a callable interface to set and clear the MOR bit. The generic UEFI interfaces to set and get these variables are used. Familiarity with these UEFI APIs is assumed.

4.1 MemoryOverwriteRequestControl Variable

The MemoryOverwriteRequestControl UEFI variable gives users (e.g., OS, loader) the ability to indicate to the platform that secrets are present in memory and that the platform firmware must clear memory upon a restart.

The OS loader does not create the variable. Rather, the firmware is required to create it and supports the semantics described here.

4.1.1 GUID

```
#define MEMORY_ONLY_RESET_CONTROL_GUID \
   { 0xe20939be, 0x32d4, 0x41be, 0xa1, 0x50, 0x89, 0x7f, 0x85, 0xd4,
   0x98, 0x29 }
```

4.1.2 Description

The name of the UEFI variable SHALL be defined as "**MemoryOverwriteRequestControl**" and is a 1-byte unsigned value. The attributes SHALL be:

```
   EFI_VARIABLE_NON_VOLATILE |
   EFI_VARIABLE_BOOTSERVICE_ACCESS |
   EFI_VARIABLE_RUNTIME_ACCESS
```

since the variable needs to be maintained across all types of reboots, and be available at boot time and run time.

The layout of the variable is described in Table 1.

4.1.3 Usage

Variable creation: Upon each reboot, the platform firmware must check for the existence and correct attributes of the MemoryOverwriteRequestControl variable. If the variable does not exist as defined above, the platform firmware must create the variable as defined above. Upon creation, the platform firmware should set the initial value of the MemoryOverwriteRequestControl variable to 0.

Upon each reboot, the platform firmware must check the value of bit 0 of the MemoryOverwriteAction_BitValues variable. If bit 0 is set, the platform firmware MUST clear all memory prior to continuing with the boot process. Once the memory is cleared, bit 0 SHOULD be cleared since any secrets have been removed.

The MemoryOverwriteRequestControl variable is protected by MemoryOverwriteRequestControlLock variable.

The OS is expected to purge secrets from memory and clear the MemoryOverwriteRequest variable in the event of a normal shutdown so that Platform Firmware will not normally be required to clear memory.

If a call to **SetVariable** with the correct GUID and variable name (MemoryOverwriteRequestControl) either has *DataSize* set to 0, or the *Attributes* value does not match the *Attributes* value returned from GetVariable, the service must return EFI_INVALID_PARAMETER without changing the state of the variable.

4.2 MemoryOverwriteRequestControlLock Variable

Previous versions of this specification defined system Platform Firmware security mitigation using the MemoryOverwriteRequestControl UEFI variable. To prevent and defend against advanced memory attacks, this specification augments MemoryOverwriteRequestControl to support locking with the MemoryOverwriteRequestControlLock variable.

4.2.1 GUID

```
#define MEMORY_OVERWRITE_REQUEST_CONTROL_LOCK_GUID \
   { 0xBB983CCF, 0x151D, 0x40E1, 0xA0, 0x7B, 0x4A, 0x17, 0xBE, 0x16,
   0x82, 0x92 }
```

4.2.2 Description

The name of the UEFI variable will be "**MemoryOverwriteRequestControlLock**" and it is a 1 byte unsigned value. The attribute must be:

```
EFI_VARIABLE_NON_VOLATILE |
EFI_VARIABLE_BOOTSERVICE_ACCESS |
EFI_VARIABLE_RUNTIME_ACCESS
```

since the variable needs to hold across all types of reboots, and be available at boot time and run time.

The definition of the value returned by **GetVariable** is described in Table 2. Any other value is treated as undefined and should not be returned by a proper implementation.

Table 2 MemoryOverwriteRequestControlLock GetVariable Definition

Lock State	Size in Bytes	Output Value	Description
Unlocked	1	0	MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl are unlocked. They can be updated.
Locked without key	1	1	MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl are locked and read only. They cannot be updated until next boot.
Locked with key	1	2	MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl are locked and read only. They can be unlocked with a key specified in SetVariable().

The definition of the value set by **SetVariable** is described in Table 3. Any other value is treated as invalid input and should be rejected by a proper implementation.

For a call to **SetVariable** (MemoryOverwriteRequestControlLock) with *DataSize* set to 0, or *Data* set to NULL, or the *Attributes* set to 0, platform firmware must return EFI_WRITE_PROTECTED without changing the state of the variable (see Figure 4).

For a call to **SetVariable** (MemoryOverwriteRequestControlLock) with other unexpected *DataSize*, or other unexpected *Attributes*, platform firmware must return EFI_INVALID_PARAMETER without changing the state of the variable (see Figure 4).

Table 3 MemoryOverwriteRequestControlLock SetVariable Meaning

Lock action	Size in Bytes	Input Value	Description
Unlock	1	0	Try to unlock MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl. **SetVariable SHALL** return one of the following: If current lock state is Unlocked then the lock state is unchanged and SetVariable returns **EFI_SUCCESS**. If current lock state is Locked without key or Locked with key then the lock state is unchanged and SetVariable returns **EFI_ACCESS_DENIED**. Note: An attempt to unlock a locked state will always fail and is listed for completeness in this table. The locked state is reset on reboot.
Lock without key	1	1	Try to lock MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl. **SetVariable SHALL** return one of the following: If current lock state is unlocked then the lock state is updated to Locked without key and SetVariable returns **EFI_SUCCESS**. If current lock state is Locked without key or Locked with key then the lock state is unchanged and SetVariable returns **EFI_ACCESS_DENIED**.
Lock/Unlock with key	8	8-byte value that represents a shared secret key	Try to lock MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl with key, if the current lock state is Unlocked. Try to unlock MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl with key, if the current lock state is Locked with key. **SetVariable SHALL** return one of the following: If current lock state is unlocked then the lock state is updated to Locked with key and SetVariable returns **EFI_SUCCESS**. If current lock state is locked with key and the input 8-byte shared secret key matches the 8-byte shared secret key in previous SetVariable() lock with key action then the lock state is updated to Unlocked and SetVariable returns **EFI_SUCCESS**. If current lock state is Locked without key then the lock state is unchanged and SetVariable returns **EFI_ACCESS_DENIED**. If current lock state is locked with key and the input 8-byte shared secret key does not match the 8-byte shared secret key in previous SetVariable() lock with key action then the lock state is updated to Locked without key to prevent dictionary attack and SetVariable returns **EFI_ACCESS_DENIED**.

4.2.3 Usage

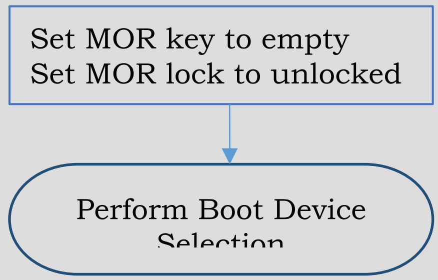

Figure 3 Initialization of MemoryOverwriteRequestControlLock Variable

On every boot, platform firmware shall initialize MemoryOverwriteRequestControlLock to a single-byte value of 0x00 (indicating unlocked) before the Boot Device Selection (BDS) phase (see **Error! Reference source not found.**). Platform Firmware shall prevent the deletion of the MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl variables and the modification of their attributes.

When **SetVariable** for MemoryOverwriteRequestControlLock is first called by passing a valid non-zero value in Data, the access mode for both MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl is changed to read-only, indicating that they are locked.

SetVariable (MemoryOverwriteRequestControlLock) is passed a single byte of 0x01 to lock MemoryOverwriteRequestControlLock (see Figure 8).

SetVariable (MemoryOverwriteRequestControlLock) also accepts an 8-byte value that represents a shared secret key. To generate that key, use a high-quality entropy source such as the Trusted Platform Module or a hardware random number generator. After setting a key, both the caller and firmware should save copies of this key in a read-protected location (see Figure 5).

If any other value is specified in **SetVariable** (MemoryOverwriteRequestControlLock), the call fails with status EFI_INVALID_PARAMETER.

SetVariable (MemoryOverwriteRequestControlLock) does not commit MemoryOverwriteRequestControlLock to flash (just changes the internal lock state). **GetVariable** (MemoryOverwriteRequestControlLock) returns the lock state and never exposes the key.

When the MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl variables are locked, invocations of **SetVariable** (MemoryOverwriteRequestControlLock) are first checked against the registered key using a constant-time algorithm, which is an algorithm that always takes the same time independent of the input or the registered key. If there is a registered key and the input is a key and both keys match, the variables transition back to an unlocked state. After this first attempt or if no key is registered, subsequent attempts to set this variable fail with EFI_ACCESS_DENIED to prevent brute force attacks. In that case, system reboot is the only way to unlock the variables (see Figure 6).

The operating system detects the presence of MemoryOverwriteRequestControlLock and its state by calling **GetVariable**. The operating system can then lock the current value of MemoryOverwriteRequestControl by setting the MemoryOverwriteRequestControlLock value to 0x1. Alternatively, the operating system may specify a key to enable unlocking in the future after secret data has been securely purged from memory (see Figure 7).

See Table 2 for the detailed return values of **GetVariable** for MemoryOverwriteRequestControlLock. See Table 3 for detailed return status and action of **SetVariable** for MemoryOverwriteRequestControlLock.

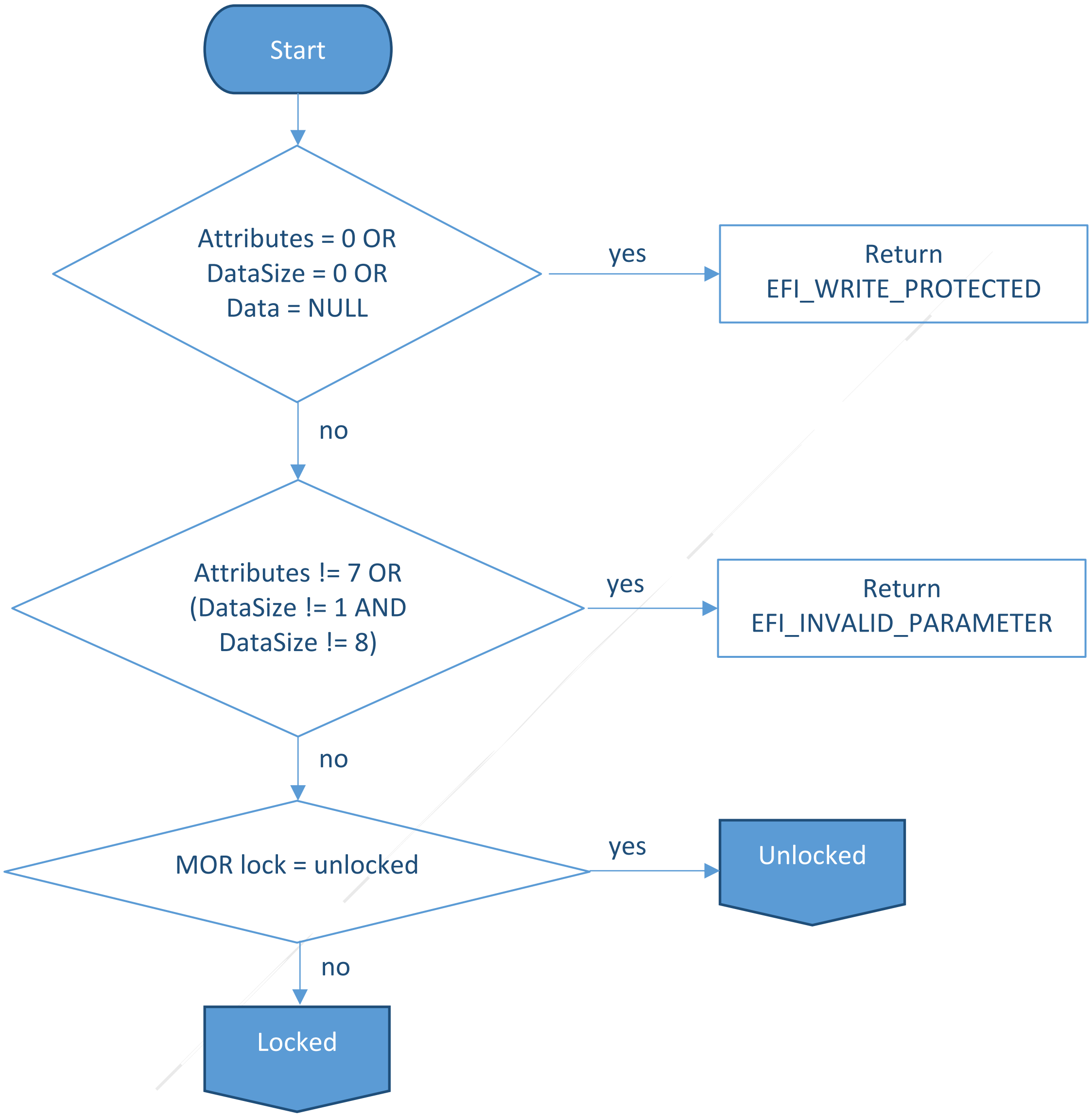

Figure 4 SetVariable (MemoryOverwriteRequestControlLock)

335 The following figures portray the normative requirements from the perspective of the workflow. If there is any conflict between the workflow described in the figures and the normative text, the normative text takes precedence.

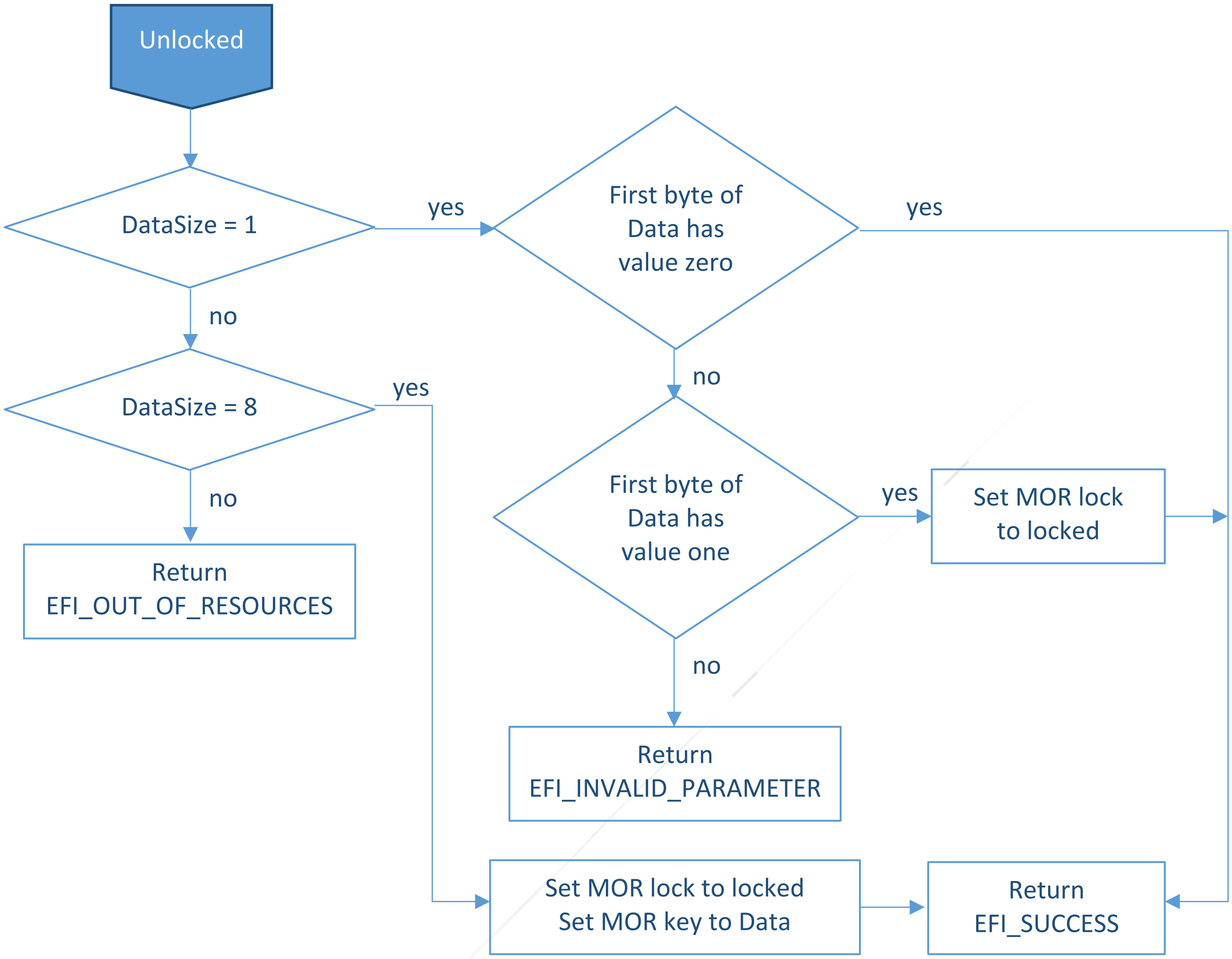

Figure 5 SetVariable (MemoryOverwriteRequestControlLock) if unlocked

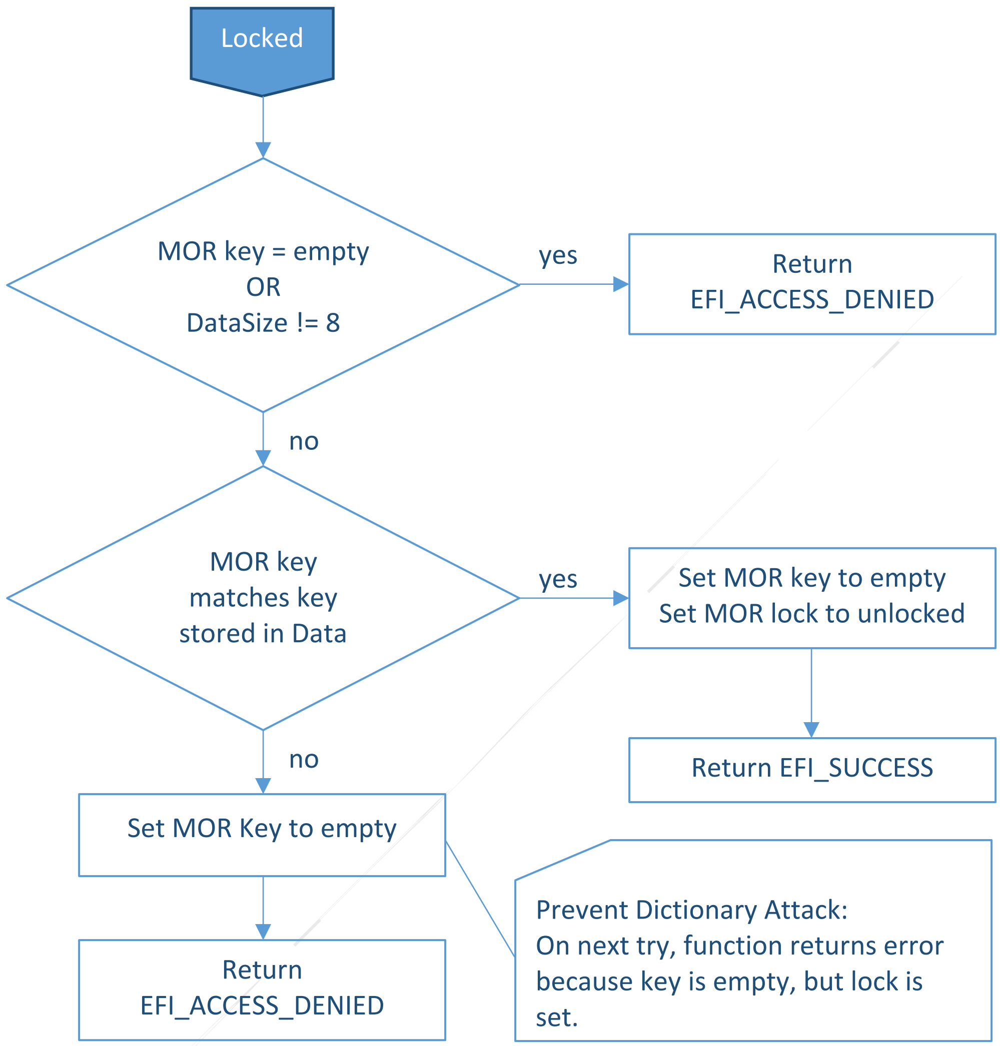

Figure 6 SetVariable (MemoryOverwriteRequestControlLock) if locked with key

340

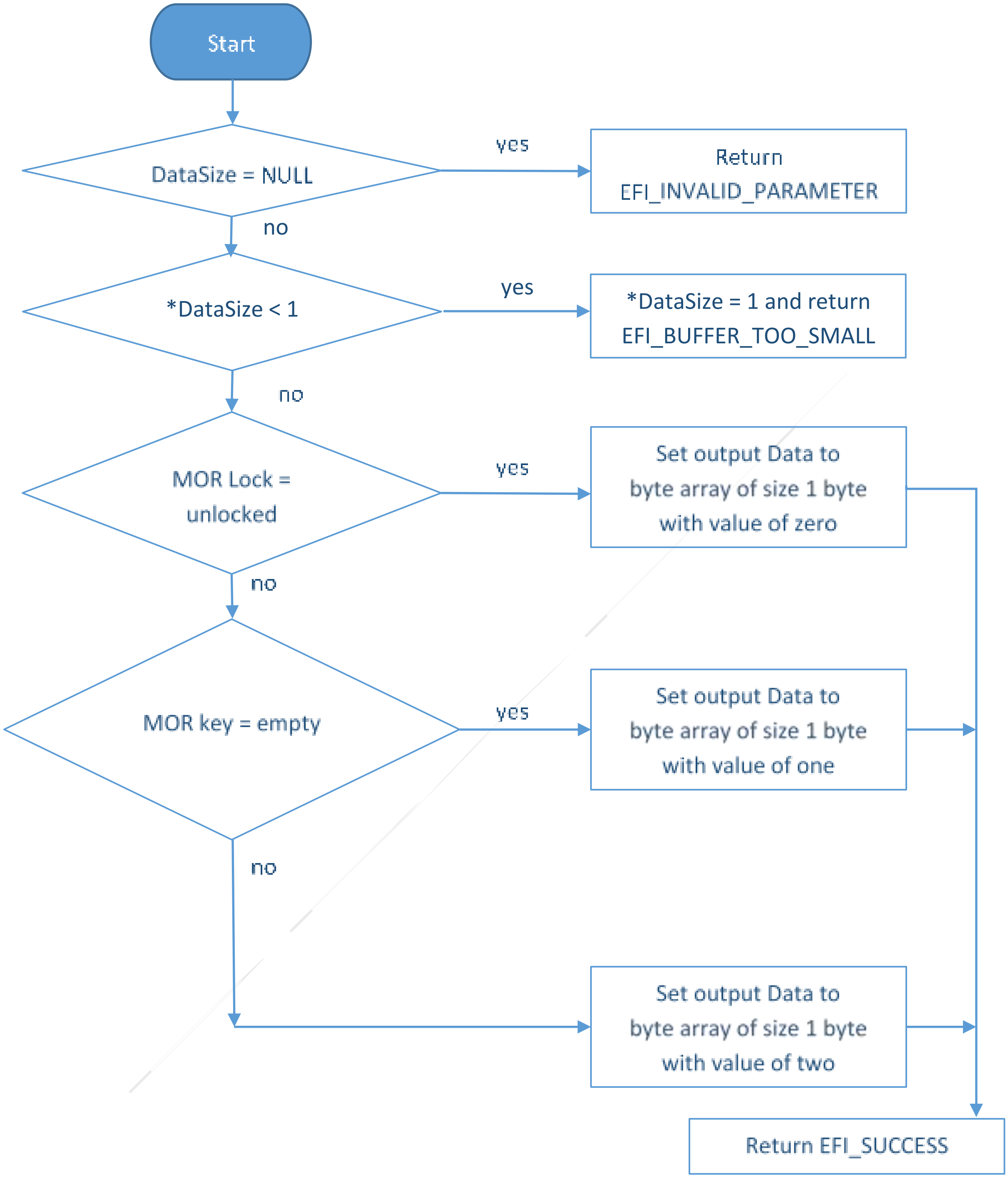

Figure 7 GetVariable (memoryOverwriteRequestControlLock)

5 Interface for Conventional Platform Firmware

This Section is superseded and kept here for informative purposes only. It can be used to set the MOR bit in systems with conventional Platform Firmware. For conventional Platform Firmware, the diagrams analogous to Figure 1 and Figure 2 are below, with Figure 8 corresponding to Figure 1 and Figure 9 corresponding to Figure 2.

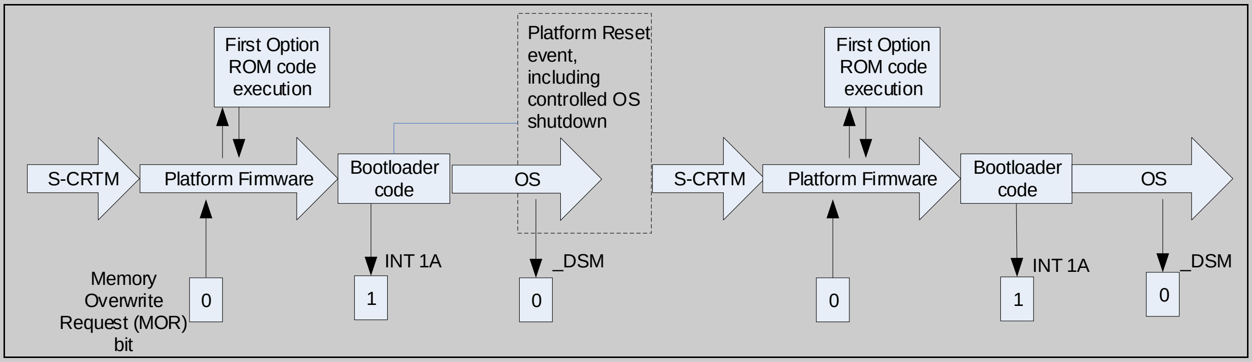

Figure 8 Platform Boot Cycle: with Complete OS Shutdown

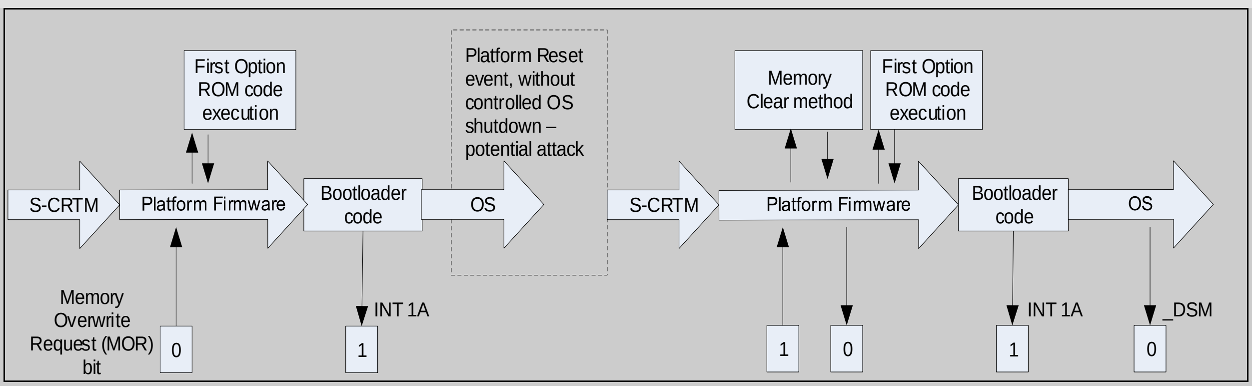

Figure 9 Platform Boot Cycle: without Complete OS Shutdown

5.1 TCG_SetMemoryOverwriteRequestBit Function

INT 1Ah, (AH)=BBh, (AL)=08h

This function sets or clears the Memory Overwrite Request (MOR) bit.

On entry:

(AH) = BBh
(AL) = 08h
(ES) = Segment portion of pointer to input parameter block
(DI) = Offset portion of pointer to input parameter block
(EBX) = 41504354h
(ECX) = 0
(EDX) = 0

On return:

(EAX) = Return Code as defined in the section titled “Application Level Interface – INT 1A TCG Functions” in the latest TCG PC Client Specific Implementation Specification for Conventional Platform Firmware.

All other registers are preserved.

5.2 TCG_SetMemoryOverwriteRequestBit Input Parameter Block

Table 4 TCG_SetMemoryOverwriteRequestBit Input Parameter Block

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes, of the input parameter block, set to 0005h.
02h	WORD	Reserved	Caller MUST set all to 0s
04h	BYTE	MemoryOverwriteAction_BitValue	This parameter sets the value POST Platform Firmware is to set the non-volatile Memory Overwrite Request (MOR) bit to. Values are described in Table 1 Variable Layout.

6 ACPI _DSM Function

This section is superseded and kept here for informative purposes only. It can be used to clear the MOR bit in systems with conventional Platform Firmware.

OS code MUST use a _DSM ACPI control method to clear the MOR bit as part of a controlled OS shutdown.

- Table 5 defines the function that MUST be exposed by the Platform Firmware, and the behavior the OS can expect upon invoking this function.
- The function MUST reside in the _DSM control method in the ACPI device object for the TPM 1.2. The UUID function identifier to be used exclusively for the Memory Clear Interface MUST be "376054ED-CC13-4675-901C-4756D7F2D45D"
- Function indices MUST start at index 1, since function 0 is the standard _DSM query function

Table 5 Memory Clear Interface Functions

Function Definition	Function Behavior
a) Set MOR Bit State Arguments: Arg0 (Buffer): UUID = 376054ED-CC13-4675-901C-4756D7F2D45D Arg1 (Integer): Revision ID = 1 Arg2 (Integer): Function Index = 1 Arg3 (Package): Arguments = Package -- Type: Integer Purpose: Operation Value of the Request Description: Byte 0 of Arg3 is defined in Table 1 Variable Layout The caller MUST set all other bytes of Arg3 to 0. Returns: Type: Integer Purpose: Function return code Description: 0: Success 1: General Failure Example: A submitted operation value of 0 and a return value of 0 indicates that the MOR bit has been cleared.	This function allows the OS to force the Platform Firmware to initiate a memory overwrite operation on the next boot cycle. If the OS calls this function with Arg3.ClearMemory = 1 and the function successfully sets the MOR bit, then the function MUST return Success. If the OS calls this function with Arg3.ClearMemory = 1 and the function is unable to set the MOR bit, then the function MUST return General Failure If the OS calls this function with Arg3.ClearMemory = 0 and the function successfully clears the MOR bit, then the function MUST return Success. If the OS calls this function with Arg3.ClearMemory = 0 and the function is unable to clear the MOR bit, then the function MUST return General Failure.

