TRUSTED® COMPUTING GROUP

SPECIFICATION

# TCG PC Client Platform Reset Attack Mitigation Specification

Version 1.2

November 18, 2025

Contact: admin@trustedcomputinggroup.org

PUBLISHED


© TCG 2003-2025

# DISCLAIMERS, NOTICES, AND LICENSE TERMS

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

# CONTENTS

DISCLAIMERS, NOTICES, AND LICENSE TERMS ..... 1
Table of Figures ..... 3
Table of Tables ..... 4
1 SCOPE ..... 5
1.1 Key Words ..... 5
1.2 Statement Type ..... 5
2 Introduction and Concepts ..... 6
3 Requirements ..... 8
3.1 General Requirements ..... 8
3.2 Memory Overwrite Request Optimizations ..... 8
3.3 Auto Detection of Clean Static RTM Shutdown ..... 8
4 UEFI Interface ..... 10
4.1 MemoryOverwriteRequestControl Variable ..... 10
4.1.1 GUID ..... 10
4.1.2 Description ..... 10
4.1.3 Usage ..... 11
4.2 MemoryOverwriteRequestControlLock Variable ..... 11
4.2.1 GUID ..... 11
4.2.2 Description ..... 11
4.2.3 Usage ..... 15
5 References ..... 20

# Table of Figures

Figure 1 UEFI Platform Boot Cycles: With Complete OS Shutdown ..... 7
Figure 2 UEFI Platform Boot Cycles: Without Complete OS Shutdown ..... 7
Figure 3 Initialization of MemoryOverwriteRequestControlLock Variable ..... 15
Figure 4 **SetVariable** (MemoryOverwriteRequestControlLock) ..... 16
Figure 5 **SetVariable** (MemoryOverwriteRequestControlLock) if state is Unlocked ..... 17
Figure 6 **SetVariable** (MemoryOverwriteRequestControlLock) if state is Locked ..... 18
Figure 7 **GetVariable** (MemoryOverwriteRequestControlLock) ..... 19

 © TCG 2025

# Table of Tables

Table 1 Variable Layout ..... 10
Table 2 MemoryOverwriteRequestControlLock GetVariable Definition ..... 12
Table 3 MemoryOverwriteRequestControlLock SetVariable Definition ..... 13

# 1 SCOPE

This specification for PC client and server platform firmware defines a mitigation against platform reset attacks, where an attacker forcibly resets a system in order to extract secrets that may be resident in memory.

## 1.1 Key Words

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

## 1.2 Statement Type

Please note a very 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 statement.

**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**

 © TCG 2025

# 2 Introduction and Concepts

**Start of informative comment**

**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 Static Root of Trust for Measurement (S-RTM) 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 only required to initiate and complete a memory overwrite operation when it is signaled to do so by firmware or the OS. This specification defines a mechanism to manage this signaling based on UEFI variables. The MemoryOverwriteControl variable defines a bit called ClearMemory, also referred to as the Memory Overwrite Request (MOR) bit, that persists across all types of Host Platform Resets. Figure 1 and Figure 2, which are part of this informative comment, show how platform firmware, a UEFI Bootloader, and the OS use the MOR bit to communicate with each other across all types of Host Platform Reset events.

The platform firmware must overwrite system memory following a Host Platform Reset event if the MOR bit has previously been set by a component such as the Bootloader or the OS.

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.

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 bit 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 OS within the S-RTM. However, it’s possible (and even likely) that memory under control of a Dynamic Root of Trust for Measurement (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 an external storage device such as a USB stick or 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 the hardware privilege level at which the OS runs. Therefore, the protections to invoke and control these methods are also exposed to any component at that privilege level. For this reason, this specification makes a fundamental assumption that the OS implements protections to defend itself, and any violation of those protections renders the methods discussed in this specification useless.

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

This specification assumes the Host Platform manufacturer tightly controls platform firmware update. The requirements for protecting the platform firmware update process are described in the TCG PC Client Platform Firmware Profile (PFP) Specification [1].

During a platform firmware update initiated during runtime it is expected that any platform reset is controlled by the OS and the MOR bit is cleared by the OS prior to reset. Following a reset, if platform firmware detects the MOR bit is set, the firmware will unconditionally clear memory. This means that any firmware update capsules resident in memory would also be cleared, resulting in any pending firmware updates being aborted.

**Functionality**

On a UEFI system the MOR bit is implemented using the ClearMemory bit in the UEFI variable MemoryOverwriteRequestControl. The EFI OS loader sets the MOR bit prior to the loader putting any secrets in the clear in system memory. This variable is defined in Section 4.

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.

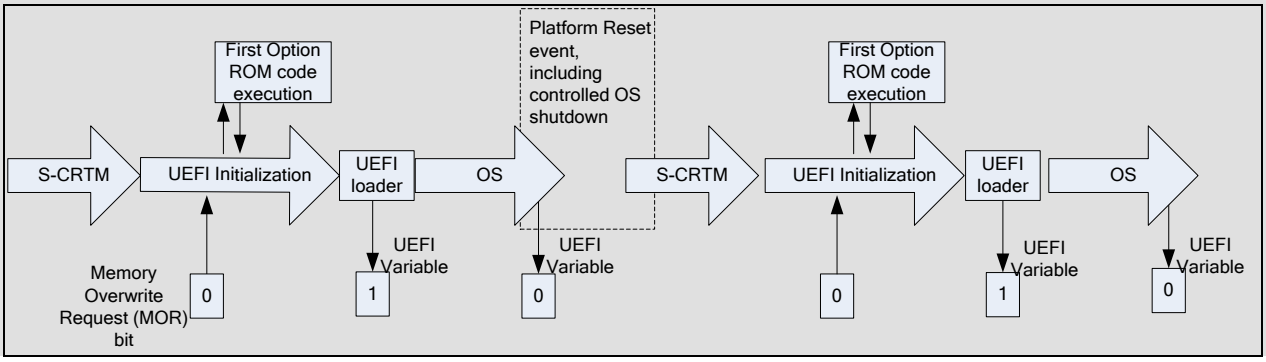



*Figure 1 UEFI Platform Boot Cycles: With Complete OS Shutdown*

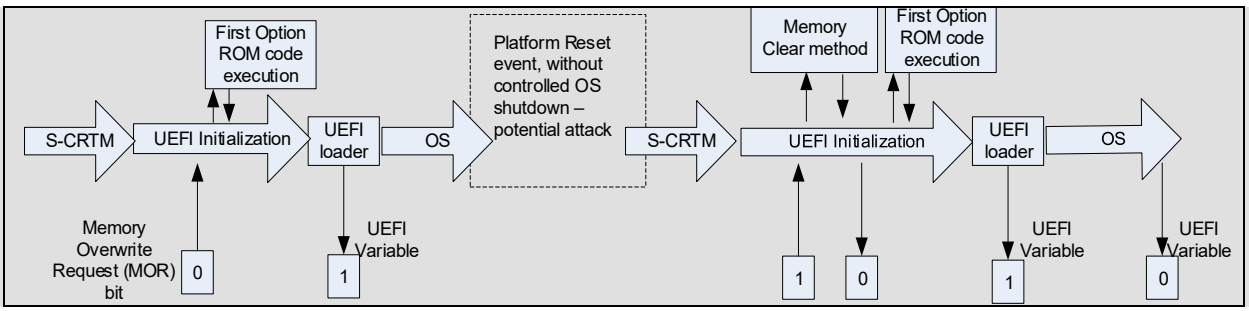



*Figure 2 UEFI Platform Boot Cycles: Without Complete OS Shutdown*

**End of informative comment**

 © TCG 2025

# 3 Requirements

**Start of informative comment**

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

**End of informative comment**

## 3.1 General Requirements

**Start of informative comment**

During a transition from the S1 to S3 power states defined by the ACPI specification [2], the OS 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 OS 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.

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.

**End of informative comment**

1. Platform firmware MUST support reading and writing the Memory Overwrite Request (MOR) bit to and from non-volatile memory (NVM) 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 any 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, or
    c. The platform firmware is unable to detect the MemoryOverwriteRequestControl variable.
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.

## 3.2 Memory Overwrite Request Optimizations

**Start of informative comment**

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.

**End of informative comment**

## 3.3 Auto Detection of Clean Static RTM Shutdown

**Start of informative comment**

Some OSes 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. OSes 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 OSes may also indicate this potential behavior by clearing the DisableAutoDetect bit in the MemoryOverwriteRequestControl variable. When this bit is zero (clear), the platform firmware can detect a clean shutdown of the OS and clear the flag itself.

There are various target shutdown operational states and certain conventional steps an OS takes when transitioning to those states. If allowed by the DisableAutoDetect bit, the platform firmware may detect an orderly OS 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 a System Management Interrupt (SMI) but no particular method is mandated by this specification.

OSes 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 OS's shutdown.

OSes 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.

It is permissible for platform 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.

**End of informative comment**

 © TCG 2025

# 4 UEFI Interface

**Start of informative comment**

UEFI uses variables to manage the MOR bit. The generic UEFI interfaces to set and get these variables are used. Familiarity with these UEFI APIs is assumed.

**End of informative comment**

## 4.1 MemoryOverwriteRequestControl Variable

**Start of informative comment**

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 support the semantics described here.

To enable the MemoryOverwriteRequestControlLock variable to be accessed at runtime and ensure that the variable's value will be preserved across reboots, the EFI_VARIABLE_NON_VOLATILE and EFI_VARIABLE_RUNTIME_ACCESS attributes are set in accordance with the UEFI specification.

**End of informative comment**

### 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 1-byte unsigned UEFI variable MUST be "MemoryOverwriteRequestControl". The variable's attributes MUST be:

```
EFI_VARIABLE_NON_VOLATILE |
EFI_VARIABLE_BOOTSERVICE_ACCESS |
EFI_VARIABLE_RUNTIME_ACCESS
```

The structure of the variable is defined in Table 1.

*Table 1 Variable Layout*

| Mnemonic | Bit Offset | Bit Length | Description |
|---|---|---|---|
| ClearMemory | 0 | 1 | 0 = Firmware MUST NOT clear memory<br>1 = Firmware MUST clear memory.<br><br>See detailed requirements in section 4.1.3. |
| Reserved | 1 | 3 | Reserved (currently unused). Firmware MUST ignore these bits on a read and set to 0 on a write. |
| DisableAutoDetect | 4 | 1 | 0 = Firmware MAY autodetect a clean shutdown of the Static RTM OS. See Section 3.3 for details.<br>1 = Firmware MUST NOT autodetect a clean shutdown of the Static RTM OS |
| Reserved | 5 | 3 | Reserved (currently unused). Firmware MUST ignore these bits on a read and set to 0 on a write. |

### 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 in section 4.1.2, the platform firmware MUST create the variable as defined in section 4.1.2. Upon creation, the platform firmware SHOULD set the initial value of the MemoryOverwriteRequestControl variable to 0x00.

Upon each reboot, the platform firmware MUST check the ClearMemory bit in the MemoryOverwriteRequestControl variable. If the ClearMemory bit is set, the platform firmware MUST overwrite all memory prior to continuing with the boot process. Once the memory is overwritten, the ClearMemory bit SHALL be cleared since any secrets have been removed.

The MemoryOverwriteRequestControl variable is protected by the MemoryOverwriteRequestControlLock variable.

The OS is expected to purge secrets from memory and set the MemoryOverwriteRequestControl variable to 0x00 in the event of a normal shutdown so that platform firmware will not normally be required to clear memory.

If MemoryOverwriteRequestControl is locked, an attempt to delete MemoryOverwriteRequestControl by calling **SetVariable** with the parameters *DataSize* or *Attributes* set to 0, platform firmware MUST return `EFI_ACCESS_DENIED` without changing the state of the variable.

**Start of informative comment**

If MemoryOverwriteRequestControl is not locked, platform firmware treats the deletion request as it would any normal EFI variable. This means platform firmware may process the deletion request without error or enforce firmware-specific policies around deletion of MemoryOverwriteRequestControl. EFI reference code implemented to an earlier version of this specification may return `EFI_INVALID_PARAMETER`.

**End of informative comment**

After receiving a call to **SetVariable** to modify MemoryOverwriteRequestControl with an unexpected *DataSize*, or unexpected *Attributes*, platform firmware MUST return `EFI_INVALID_PARAMETER` without changing the state of the variable.

## 4.2 MemoryOverwriteRequestControlLock Variable

**Start of informative comment**

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

To enable the MemoryOverwriteRequestControlLock variable to be accessed at runtime the EFI_VARIABLE_NON_VOLATILE and EFI_VARIABLE_RUNTIME_ACCESS attributes are set in accordance with the UEFI specification.

**End of informative comment**

### 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 MUST be **"MemoryOverwriteRequestControlLock**" and the unsigned value MUST be either 1 byte or 8 bytes. The variable's attributes MUST be:

```
EFI_VARIABLE_NON_VOLATILE |
```

 © TCG 2025

```
EFI_VARIABLE_BOOTSERVICE_ACCESS |
EFI_VARIABLE_RUNTIME_ACCESS
```

A call to **GetVariable** to read MemoryOverwriteRequestControlLock returns a value reflecting the current lock state. The definition of these values is in Table 2. Any other output value is treated as undefined and MUST NOT be returned by platform firmware. Figure 7 illustrates the behavior when a call to **GetVariable** is made for MemoryOverwriteRequestControlLock with different lock states and input parameters.

If there is a call to **GetVariable** to read MemoryOverwriteRequestControlLock where the value of DataSize parameter is less than 1, platform firmware MUST return `EFI_BUFFER_TOO_SMALL` without changing the state of the variable.

If there is a call to **GetVariable** to read MemoryOverwriteRequestControlLock where the DataSize parameter is NULL, platform firmware MUST return `EFI_INVALID_PARAMETER` without changing the state of the variable.

*Table 2 MemoryOverwriteRequestControlLock GetVariable Definition*

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

The definition of the value set by **SetVariable** is in Table 3. Any other input value is treated as an invalid input and MUST be rejected. Figure 4, Figure 5, and Figure 6 illustrate the behavior when a call to set **SetVariable** is made for MemoryOverwriteRequestControlLock with different lock states and input parameters.

After receiving an attempt to delete MemoryOverwriteRequestControlLock by calling **SetVariable** with the parameters *DataSize* or *Attributes* set to 0, platform firmware MUST return `EFI_WRITE_PROTECTED` without changing the state of the variable (see Figure 4).

After receiving a call to **SetVariable** to modify MemoryOverwriteRequestControlLock with an unexpected *DataSize*, or unexpected *Attributes*, platform firmware MUST return `EFI_INVALID_PARAMETER` without changing the state of the variable (see Figure 4).

*Table 3 MemoryOverwriteRequestControlLock SetVariable Definition*

| Lock action | Size in Bytes | Input Value | Description |
|---|---|---|---|
| Unlock | 1 | 0 | Try to unlock MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl.<br><br>**SetVariable** MUST return one of the following:<br>If current lock state is *Unlocked* then the lock state is unchanged and SetVariable returns **`EFI_SUCCESS`**.<br>If current lock state is *Locked without key* or *Locked with key* then the lock state is unchanged and SetVariable returns **`EFI_ACCESS_DENIED`**.<br><br>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.<br><br>**SetVariable** MUST return one of the following:<br>If current lock state is *Unlocked* then the lock state is updated to *Locked without key* and SetVariable returns **`EFI_SUCCESS`**.<br>If current lock state is *Locked without key* or *Locked with key* then the lock state is unchanged and SetVariable returns **`EFI_ACCESS_DENIED`**. |

 © TCG 2025

| | | | |
|---|---|---|---|
| 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*.<br>Try to unlock MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl with key, if the current lock state is *Locked with key*.<br><br>**SetVariable** MUST return one of the following:<br>If current lock state is *Unlocked* then the lock state is updated to *Locked with key* and SetVariable returns **`EFI_SUCCESS`**.<br>If current lock state is *Locked with key* and the input 8-byte shared secret key matches the 8-byte shared secret key in the previous SetVariable() lock with key action then the lock state is updated to *Unlocked* and SetVariable returns **`EFI_SUCCESS`**.<br>If current lock state is *Locked without key* then the lock state is unchanged and SetVariable returns **`EFI_ACCESS_DENIED`**.<br>If current lock state is *Locked with key*<br>• If the input key value is empty or the data size is not 8, SetVariable returns **`EFI_ACCESS_DENIED`**.<br>• If the input 8-byte shared secret key does not match the 8-byte shared secret key in the 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

**Start of informative comment**

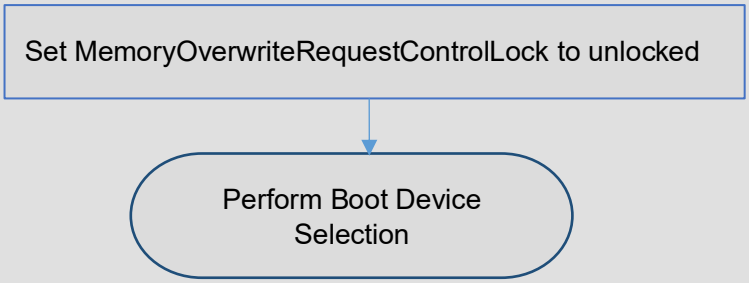



**Figure 3 Initialization of MemoryOverwriteRequestControlLock Variable**

On every boot, platform firmware initializes MemoryOverwriteRequestControlLock to a single-byte value of 0x00 (indicating a status of *Unlocked*) before the Boot Device Selection (BDS) phase (see Figure 3). Platform firmware is responsible for preventing deletion of the MemoryOverwriteRequestControlLock and MemoryOverwriteRequestControl variables and the modification of their attributes.

When **SetVariable** for MemoryOverwriteRequestControlLock is first called with 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.

If **SetVariable** (MemoryOverwriteRequestControlLock) is passed a single byte value of 0x01 the lock state is changed to *Locked without key*.

**SetVariable** (MemoryOverwriteRequestControlLock) also accepts an 8-byte value that represents a shared secret key and results in a lock state of *Locked with 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.

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

**SetVariable** (MemoryOverwriteRequestControlLock) does not commit the Data parameter passed in to non-volatile memory (just changes the 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) should be checked against the registered key using a constant-time algorithm, ensuring that the check takes the same time independent of optimizations in the implementation. This prevents time-based side-channel attacks. If there is a registered key, the input is a key, and both keys match, the variables transition back to an unlocked state. After an unsuccessful 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 OS detects the presence of MemoryOverwriteRequestControlLock and its state by calling **GetVariable**. The OS can then lock the current value of MemoryOverwriteRequestControl by setting the MemoryOverwriteRequestControlLock value to 0x1. Alternatively, the OS may specify a key to enable unlocking in the future after secret data has been securely purged from memory.

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.

**End of informative comment**

 © TCG 2025

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.

In Figure 5, Figure 6, and Figure 7 the “MOR key” is the shared secret key described in Table 3.

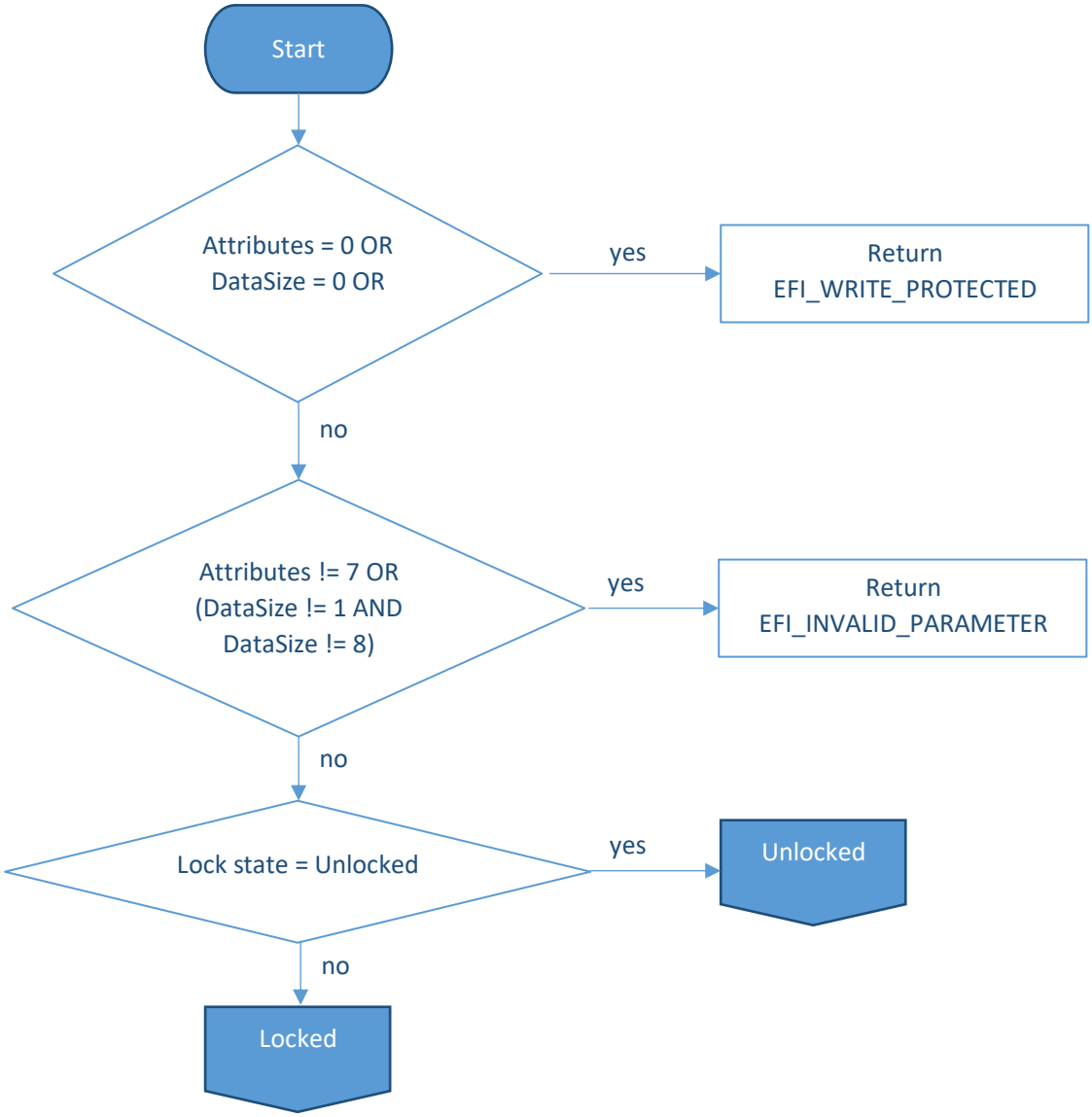



*Figure 4 **SetVariable** (MemoryOverwriteRequestControlLock)*

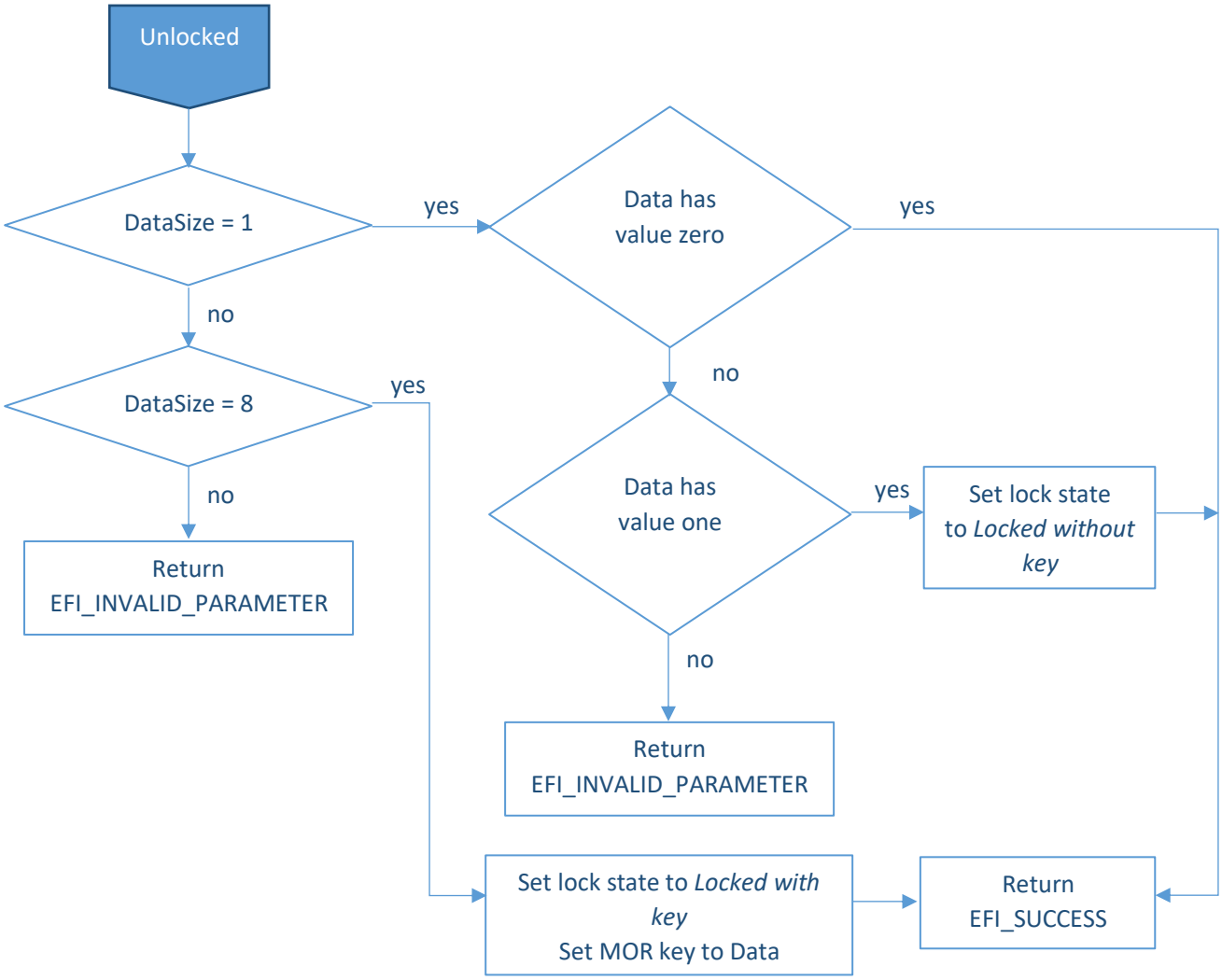



*Figure 5* ***SetVariable*** *(MemoryOverwriteRequestControlLock) if state is Unlocked*

 © TCG 2025

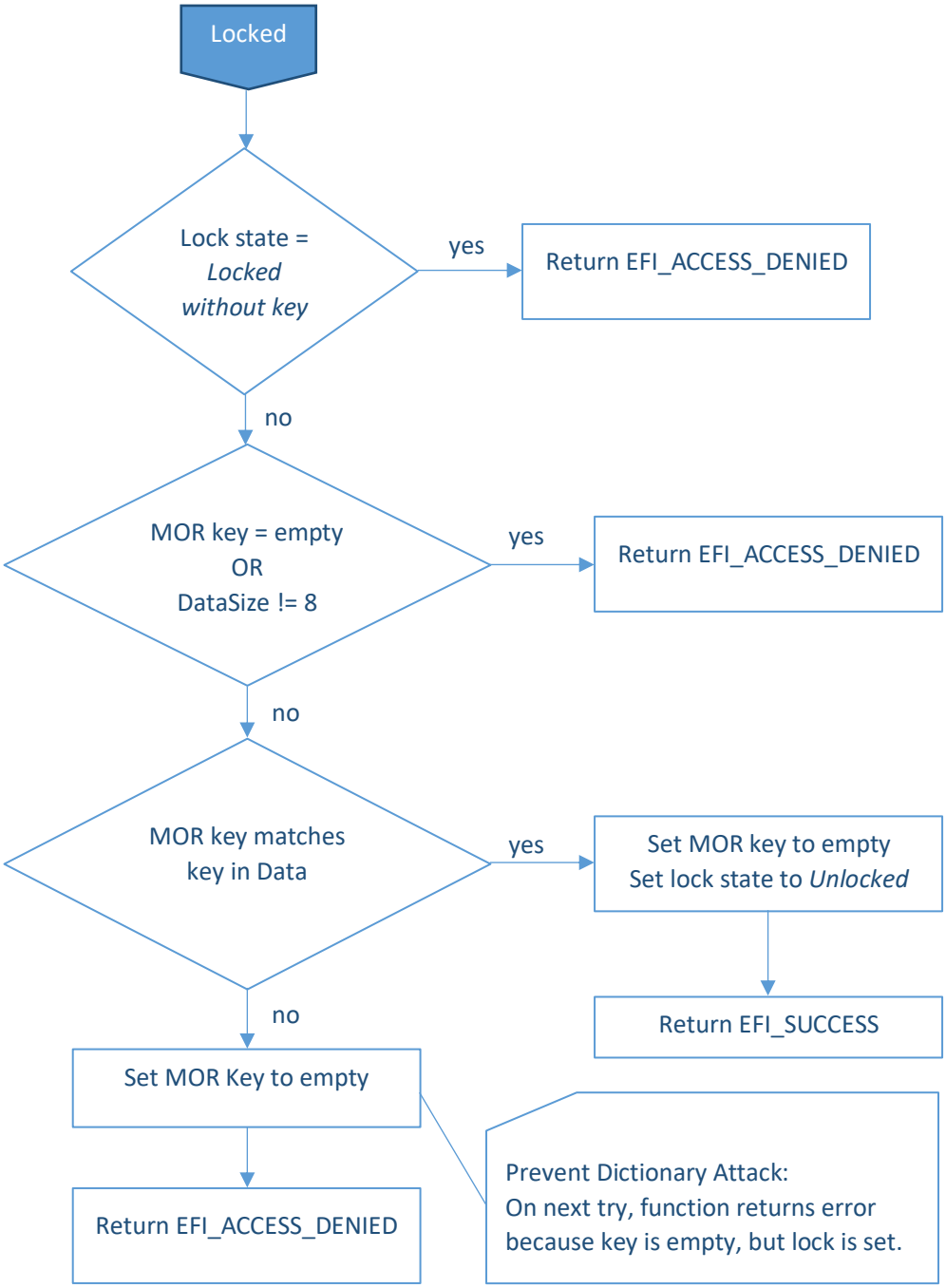



*Figure 6 **SetVariable** (MemoryOverwriteRequestControlLock) if state is Locked*

In Figure 7 “*DataSize” indicates a dereference of the pointer for the DataSize parameter passed to GetVariable.

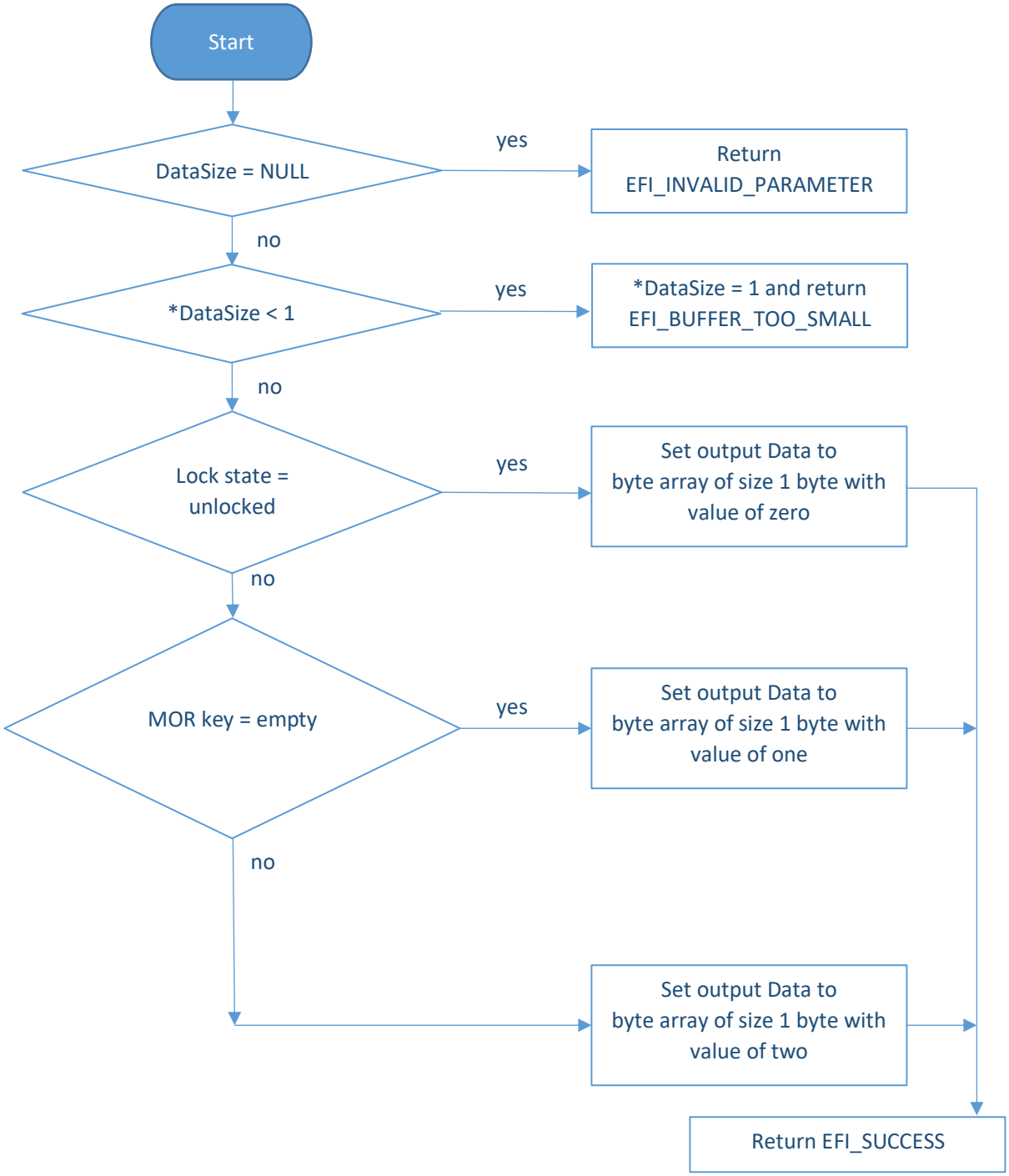



*Figure 7 **GetVariable** (MemoryOverwriteRequestControlLock)*

 © TCG 2025

# 5 References

Where a reference is undated, the latest available versions can be used.

[1] TCG PC Client Platform Firmware Profile (PFP)

https://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/

[2] Advanced Configuration and Power Interface (ACPI), Version 6.0 or later

https://uefi.org/specifications