[Public]

Secure VM Service Module for SEV-SNP Guests

Guest Communication Interface

Publication # **58019**	Revision: **1.01**
Issue Date:	**March 2026**

Advanced Micro Devices 

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, AMD EPYC, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Specification Agreement

This Specification Agreement (this "Agreement") is a legal agreement between Advanced Micro Devices, Inc. ("AMD") and "You" as the recipient of the attached AMD Specification (the "Specification"). If you are accessing the Specification as part of your performance of work for another party, you acknowledge that you have authority to bind such party to the terms and conditions of this Agreement. If you accessed the Specification by any means or otherwise use or provide Feedback (defined below) on the Specification, You agree to the terms and conditions set forth in this Agreement. If You do not agree to the terms and conditions set forth in this Agreement, you are not licensed to use the Specification; do not use, access or provide Feedback about the Specification.

In consideration of Your use or access of the Specification (in whole or in part), the receipt and sufficiency of which are acknowledged, You agree as follows:

1. You may review the Specification only (a) as a reference to assist You in planning and designing Your product, service or technology ("Product") to interface with an AMD product in compliance with the requirements as set forth in the Specification and (b) to provide Feedback about the information disclosed in the Specification to AMD.
2. Except as expressly set forth in Paragraph 1, all rights in and to the Specification are retained by AMD. This Agreement does not give You any rights under any AMD patents, copyrights, trademarks or other intellectual property rights. You may not (i) duplicate any part of the Specification; (ii) remove this Agreement or any notices from the Specification, or (iii) give any part of the Specification, or assign or otherwise provide Your rights under this Agreement, to anyone else.
3. The Specification may contain preliminary information, errors, or inaccuracies, or may not include certain necessary information. Additionally, AMD reserves the right to discontinue or make changes to the Specification and its products at any time without notice. The Specification is provided entirely "AS IS." AMD MAKES NO WARRANTY OF ANY KIND AND DISCLAIMS ALL EXPRESS, IMPLIED AND STATUTORY WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, TITLE OR THOSE WARRANTIES ARISING AS A COURSE OF DEALING OR CUSTOM OF TRADE. AMD SHALL NOT BE LIABLE FOR DIRECT, INDIRECT, CONSEQUENTIAL, SPECIAL, INCIDENTAL, PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, LOST PROFITS, LOSS OF CAPITAL, LOSS OF GOODWILL) REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE) AND STRICT PRODUCT LIABILITY OR OTHERWISE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
4. Furthermore, AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur.

5. You have no obligation to give AMD any suggestions, comments or feedback ("Feedback") relating to the Specification. However, any Feedback You voluntarily provide may be used by AMD without restriction, fee or obligation of confidentiality. Accordingly, if You do give AMD Feedback on any version of the Specification, You agree AMD may freely use, reproduce, license, distribute, and otherwise commercialize Your Feedback in any product, as well as has the right to sublicense third parties to do the same. Further, You will not give AMD any Feedback that You may have reason to believe is (i) subject to any patent, copyright or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any product or intellectual property incorporating or derived from Feedback or any Product or other AMD intellectual property to be licensed to or otherwise provided to any third party.
6. You shall adhere to all applicable U.S., European, and other export laws, including but not limited to the U.S. Export Administration Regulations ("EAR"), (15 C.F.R. Sections 730 through 774), and E.U. Council Regulation (EC) No 428/2009 of 5 May 2009. Further, pursuant to Section 740.6 of the EAR, You hereby certifies that, except pursuant to a license granted by the United States Department of Commerce Bureau of Industry and Security or as otherwise permitted pursuant to a License Exception under the U.S. Export Administration Regulations ("EAR"), You will not (1) export, re-export or release to a national of a country in Country Groups D:1, E:1 or E:2 any restricted technology, software, or source code You receive hereunder, or (2) export to Country Groups D:1, E:1 or E:2 the direct product of such technology or software, if such foreign produced direct product is subject to national security controls as identified on the Commerce Control List (currently found in Supplement 1 to Part 774 of EAR). For the most current Country Group listings, or for additional information about the EAR or Your obligations under those regulations, please refer to the U.S. Bureau of Industry and Security's website at *http://www.bis.doc.gov/*.
7. If You are a part of the U.S. Government, then the Specification is provided with "RESTRICTED RIGHTS" as set forth in subparagraphs (c) (1) and (2) of the Commercial Computer Software-Restricted Rights clause at FAR 52.227-14 or subparagraph (c) (1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7013, as applicable.
8. This Agreement is governed by the laws of the State of California without regard to its choice of law principles. Any dispute involving it must be brought in a court having jurisdiction of such dispute in Santa Clara County, California, and You waive any defenses and rights allowing the dispute to be litigated elsewhere. If any part of this agreement is unenforceable, it will be considered modified to the extent necessary to make it enforceable, and the remainder shall continue in effect. The failure of AMD to enforce any rights granted hereunder or to take action against You in the event of any breach hereunder shall not be deemed a waiver by AMD as to subsequent enforcement of rights or subsequent actions in the event of future breaches. This Agreement is the entire agreement between You and AMD concerning the Specification; it may be changed only by a written document signed by both You and an authorized representative of AMD.

Acknowledgements

AMD would like to acknowledge Jon Lange for his work in creating the initial draft of the SVSM specification. Thank you, Jon.

Contents

List of Tables

Revision History

Date	Revision	Description

March 2026	1.01	• New protocols
		  ○ APIC Emulation protocol
		  ○ UEFI Management Mode protocol
		• Core protocol changes
		  ○ SVSM_CORE_PVALIDATE: 2M to 4K page validation fallback
		• vTPM protocol changes
		  ○ Introduce attestation manifest version 1 format

May 2023	1.00	• Incorporated feedback from 0.62 review ○ Added requirement to zero memory of pages that are being made valid via the PVALIDATE call

March 2023	0.62	• Incorporated feedback from 0.61 review
		  ○ Clarifications around vCPU create and delete
		  ○ Added check for attestation buffer size
		  ○ Document clarifications

February 2023	0.61	• Incorporated feedback from 0.60 review
		  ○ Updated vTPM attestation information
		  ○ Added single service attestation call
		  ○ Support for returning attestation certificates
		  ○ Document clarifications

January 2023	0.60	• Attestation protocol added • vTPM protocol added • Core protocol updates
August 2022	0.50	• Initial public draft

1 Abstract

AMD's Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) offers powerful and flexible support for isolation of a guest Virtual Machine (VM) state from an untrusted host operating system. The VM Permission Level (VMPL) feature permits the inclusion of components in the guest that can run with a higher privilege than the guest operating system, offering an environment for secure, privileged code modules to run without interference from the bulk of the guest. Such code modules are not part of the guest operating system and may be designed to be modular, offering compatibility with a wide variety of guest configurations. For such a modular design to be successful, a standard calling convention must exist to permit the guest operating system and secure modules to communicate without interference from the untrusted host environment. This document defines the standard by which these modules can exist together and communicate within a single SNP guest.

2 Scope of Document

This document defines a Secure VM Service Module (SVSM) as an environment that can host privileged modules within the guest, and it also defines mechanisms by which a guest operating system can determine the presence of the SVSM, the set of services offered by the SVSM, and the mechanism by which the guest can communicate with those services.

This document does not describe the internals of the SVSM. Multiple SVSM implementations are encouraged; this document defines standards by which an SVSM communicates with a guest operating system so that multiple SVSM implementations can be compatible with multiple guest operating systems.

This document does not describe the mechanism by which the Virtual Machine Manager (VMM) loads the SVSM. That mechanism is assumed to be specific to the host architecture. The SVSM and the firmware are expected to be loaded and measured as part of the initial guest image so that the specific identity of the SVSM and firmware associated with a guest can be verified through measurement and attestation. It is expected that the following items will be measured:

- Items measured at VMPL0
    - SVSM binary
    - SVSM BSP VM Save Area (VMSA) as a VMSA page
    - Firmware BSP VMSA as a Normal page
    - Secrets page
    - CPUID page
- Items measured at VMPL1+
    - Firmware binary

The hypervisor can measure additional pages as needed.

This document does not describe mechanisms by which the VMM communicates with the SVSM nor chooses to invoke the SVSM. Those details are assumed to be specific to the host environment, though they could be standardized under a different specification.

This document does not describe the threading model of the SVSM. Some SVSM implementations may choose a separate execution context (a unique VMSA) per guest vCPU, while other SVSM implementations may choose a single execution context that services all guest vCPUs. The negotiation of threading models between the SVSM and the host as well as the mechanism by which the host indicates which guest vCPU makes a request are assumed to be specific to the host environment, though they could be standardized under a different specification.

3 Environment

The SVSM is expected to execute in VMPL0 of the guest. To ensure privilege separation for security-sensitive services, the bulk of the guest is expected to run at a VMPL other than 0. The SVSM must offer services to proxy requests that would normally be made by a guest running at VMPL0 but which are architecturally impossible when the guest is running at a VMPL other than 0 (e.g., use of the PVALIDATE instruction and creating VMSA pages using RMPADJUST). Those services form part of the "core protocol." (See the "Core Protocol" section on page 17.)

The SVSM image and its initial data are expected to occupy a contiguous portion of the guest physical address (gPA) space. That memory must be configured with VMPL permissions that grant access to VMPL0 but not to a less privileged VMPL (lower privilege VMPL levels have a numerically higher number). The initial SVSM memory configuration must be sufficient to allow the SVSM to initialize and offer all core protocol SVSM services. A mechanism is defined by which the SVSM can obtain additional memory if required to support additional services that may be requested by the guest OS.

4 Discovery

4.1 Boot

When the guest first starts, it is expected to execute in the context of the SVSM before any lower VMPL is started. This permits the SVSM to initialize itself and make itself discoverable.

The SVSM advertises its presence by writing information into the secrets page, as described in Table 1. Note that the portions of the secrets page at byte offsets described here are always zeroed during initial construction of the secrets page and reserved for use by the SVSM. They will always be 0 for every guest unless they are initialized by the SVSM.

Table 1: Secrets Page Fields

Byte Offset	Size	Field Name	Description
0x140	8 bytes	SVSM_BASE	Base gPA of the SVSM area. This must be a multiple of 4 KB.
0x148	8 bytes	SVSM_SIZE	Number of bytes in the SVSM area. This must be a multiple of 4 KB.
0x150	8 bytes	SVSM_CAA	gPA of an 4KB area used for guest/SVSM communication.
0x158	4 bytes	SVSM_MAX_VERSION	Maximum version of the core protocol supported by the SVSM.
0x15C	1 byte	SVSM_GUEST_VMPL	Indicates the VMPL at which the guest is executing.
0x15D	3 bytes		Reserved.

Note that the SVSM is expected to zero the portion of the secrets page that contains the VM Platform Communication Key (VMPCK) associated with VMPL0. This prevents the guest OS from intruding on any conversation between the SVSM and the PSP.

When the guest OS starts, it must first read the SVSM_SIZE field. If this field is 0, then no SVSM is present. If this field is non-0, indicating that an SVSM is present, then the guest must note the memory range spanned by the SVSM so that it does not attempt to use any memory in that range. (It may, for example, choose to identify that memory as EfiPlatformReserved.) It must use the specified SVSM_CAA value for use in communication with the SVSM.

This specification assumes that the initial guest image contains only a single VMSA, used for the startup vCPU, whose contents have been measured as part of the guest launch and validated by the SVSM before executing the initial guest image. If the guest requires additional VMSAs, they should be created dynamically. (See “SVSM_CORE_CREATE_VCPU Call” section on page 24.)

It is expected that the SVSM will examine the SEV_FEATURES specified for the guest and terminate if unsupported features are enabled. For example, if SEV_FEATURES for the guest OS has the VmsaRegProt feature enabled, but the SVSM does not have support for accessing such a VMSA, then the SVSM can terminate.

4.2 Post-Boot

After boot, it may still be necessary for guest components that do not have access to the secrets page (e.g., UEFI runtime services invoked after the guest OS has started and taken over the SVSM interface) to perform calls to the SVSM. For those components, an alternate discovery mechanism is required.

To support separate components, a discovery mechanism is defined that requires specific handling of #VC exceptions. It is assumed that if the SVSM was discovered at boot, then the guest component that implements the #VC handler is also aware of the SVSM and the location of its Calling Area.

To discover whether an SVSM is present, the separate component must execute CPUID(EAX=8000_001F). This will result in a #VC exception, and the #VC handler is expected to set EAX[28]=1 in the response. When the component that executed the CPUID observes EAX[28]=1 in the response, it will know that an SVSM is reachable. This CPUID response bit is reserved for the purpose of enumerating the presence of an SVSM and will never be set by hardware or in any CPUID data generated by the PSP.

To discover the location of the SVSM Calling Area, the separate component must read MSR C001_F000. This will result in a #VC exception, and the #VC handler is expected to supply the gPA of the SVSM Calling Area as the MSR value. Writes to the MSR are not expected, and the #VC handler is not required to handle writes to the MSR. This MSR is reserved for the purpose of exposing the gPA of the SVSM Calling Area and will never be implemented in hardware.

Once the SVSM Calling Area has been located, the separate component can issue calls to the SVSM normally. The separate component should not alter the state of the SVSM in a way that is not expected by the remainder of the guest (e.g., no component should issue the SVSM_CORE_REMAP_CA command).

5 Calling Convention

Each call to the SVSM conveys data through a combination of the SVSM Calling Area (whose address was first configured through the SVSM_CAA field of the secrets page) and registers. Use of the Calling Area is necessary for the SVSM to detect the difference between a call that was issued by the guest or a spurious invocation by a poorly behaved host. Registers are used for all other parameters.

The initially configured SVSM Calling Area is a page of memory that lies outside the initial SVSM memory range and has not had its VMPL permissions restricted in any way. The address is guaranteed to be aligned to a 4 KB boundary, so the remainder of the page may be used by the guest for memory-based parameter passing if desired.

The contents of the Calling Area are described in Table 2:

Table 2: Calling Area

Byte Offset	Size	Name	Description
0x000	1 byte	SVSM_CALL_PENDING	Indicates whether a call has been requested by the guest (0=no call requested, 1=call requested).
0x001	1 byte	SVSM_MEM_AVAILABLE	Free memory is available to be withdrawn.
0x002	1 byte	SVSM_NO_EOI_REQ	Indicates to the guest that an EOI is not required after processing an interrupt under the APIC emulation protocol.
0x003	5 bytes		Reserved. The SVSM is not required to verify that these bytes are 0.

Each call is identified by a 32-bit protocol number and a 32-bit call identifier specific to the protocol as defined in Table 3.

Table 3: Protocols

Protocol Number	Protocol Description
0	Core
1	Attestation
2	vTPM
3	APIC Emulation
4	UEFI Management Mode
0x8000_0000 – 0x8000_FFFF	Reserved for AMD reference implementation specific protocols

To make a call to the SVSM, the guest OS must load the RAX register with the identifier of the call, where bits [63:32] hold the protocol number and bits [31:0] hold the call identifier within the protocol. Additional registers and/or memory may need to be configured with values specific to the call being issued. Once all memory and registers have been prepared, the guest OS must write a value of 1 to the SVSM_CALL_PENDING field of the Calling Area to indicate its readiness to issue the call. Finally, the guest OS must execute VMGEXIT to request that the host execute the SVSM on behalf of the calling vCPU (see the *Guest-Hypervisor Communication Block Standardization* for details.)

When the SVSM receives the call, it is expected to set VMSA.EFER.SVME=0 for the calling vCPU of the guest OS; this ensures that the host cannot attempt to reenter the calling vCPU while SVSM call processing is underway. (An attempt to enter the guest would result in a failure due to invalid VMSA contents.) The SVSM is then expected to examine the SVSM_CALL_PENDING field to determine whether any call was actually requested by the guest OS; if the host illegally entered the SVSM, this field will be 0. In such a case, no action will be taken by the SVSM other than setting VMSA.EFER.SVME=1 for the calling vCPU and returning to the guest. In addition, the SVSM is expected to examine VMSA.EXITCODE after setting VMSA.EFER.SVME=0 to ensure that the guest is on the expected VMGEXIT instruction boundary before proceeding. If the exit code does not indicate exiting due to VMGEXIT, the SVSM should reset VMSA.EFER.SVME=1 and take no further action before returning to the guest.

Once the SVSM determines that a calling request is legitimate, it will read the value of RAX from the VMSA of the requesting vCPU and process the call accordingly. Upon completion of the call, the SVSM will set RAX in the VMSA of the requesting vCPU to indicate the result of the call. It will clear SVSM_CALL_PENDING in the Calling Area to indicate that the call was completed, set VMSA.EFER.SVME=1 for the calling vCPU (only after VMSA.RAX and SVSM_CALL_PENDING fields have been set), and return to the guest.

Upon its return, the guest must atomically clear the SVSM_CALL_PENDING field and examine the previous value. The guest cannot trust that the host has executed the SVSM call as desired, nor can it assume that the host will not attempt to execute the SVSM call at an inopportune time, so the guest must clear the pending request at the same time that it extracts the previous value for examination. If the previous value of SVSM_CALL_PENDING was non-0, the guest knows that the SVSM never executed the call and must either retry the call or accept the fact that the host did not honor the request to execute the SVSM. If the previous value of SVSM_CALL_PENDING was 0, then the guest knows that the call completed and can examine the value of RAX to determine whether the call completed successfully.

Result values returned in RAX are 32-bit values (a 64-bit sign extension is ignored) divided into three categories:

- successful completion with distinct completion information

- unsuccessful completion for a specified reason
- requests for additional memory

Most result codes are specific to individual protocols, but a portion of the result space is reserved for common values as defined in Table 4.

Table 4: Result Codes

Result code	Name	Meaning
0x0000_0000	SVSM_SUCCESS	The call completed successfully.
0x0000_0000 - 0x0000_0FFF		Reserved for future use.
0x0000_1000 - 0x3FFF_FFFF		Defined by the protocol that was invoked.
0x4000_0000 - 0x7FFF_FFFF		Additional memory is required to complete the requested operation. Bits 29:0 indicate the number of 4 KB pages that are required.
0x8000_0000	SVSM_ERR_INCOMPLETE	The requested call was partially performed. The guest must request additional processing by setting SVSM_CALL_PENDING and invoking the SVSM again.
0x8000_0001	SVSM_ERR_UNSUPPORTED_PROTOCOL	The requested protocol is not supported.
0x8000_0002	SVSM_ERR_UNSUPPORTED_CALL	The requested call ID is not supported by the requested protocol.
0x8000_0003	SVSM_ERR_INVALID_ADDRESS	A gPA specified as part of a call is invalid.
0x8000_0004	SVSM_ERR_INVALID_FORMAT	A reserved value was specified in SVSM_CALL_PENDING.
0x8000_0005	SVSM_ERR_INVALID_PARAMETER	One or more invalid parameters were specified to a call.
0x8000_0006	SVSM_ERR_INVALID_REQUEST	The request cannot be supported by the protocol handler that was invoked.
0x8000_0007	SVSM_ERR_BUSY	The request cannot be handled at this time; the guest should issue the request again.
0x8000_0008 - 0x8000_0FFF		Reserved for future use.
0x8000_1000 - 0xFFFF_FFFF		Defined by the protocol that was invoked.

Any input parameters that do not meet the alignment requirement will return the SVSM_ERR_INVALID_PARAMETER result value.

Any input address that results in a fault when accessing the memory at that address will return the SVSM_ERR_INVALID_ADDRESS result value.

Returning the SVSM_ERR_BUSY result code is implementation dependent and not specifically identified in any protocol call description. Any call returning this result code:

- must not leave the system in an intermediate state
- describe the progress made or else be idempotent.

Similarly, it is expected that any call that returns a result code in the 0x4000_0000 – 0x7FFF_FFFF range:

- must not leave the system in an intermediate state
- describe the progress made or else be idempotent.

6 Core Protocol

All SVSM implementations must support the core protocol, which has the protocol ID value 0. The core protocol is versioned, permitting its extension over time; the initial version of the protocol is version 1. Versioning of the core protocol is strictly additive, i.e., all calls present in version 1 must be supported by all future implementations. Table 5 enumerates the set of calls supported by the core protocol.

Table 5: Core Protocol Services

Call ID	First version supported	Name	Function
0	1	SVSM_CORE_REMAP_CA	Remap the SVSM Calling Area to a new gPA.
1	1	SVSM_CORE_PVALIDATE	Execute PVALIDATE.
2	1	SVSM_CORE_CREATE_VCPU	Create a new vCPU.
3	1	SVSM_CORE_DELETE_VCPU	Delete a vCPU.
4	1	SVSM_CORE_DEPOSIT_MEM	Deposit additional memory for use by the SVSM.
5	1	SVSM_CORE_WITHDRAW_MEM	Withdraw unused memory no longer required by the SVSM.
6	1	SVSM_CORE_QUERY_PROTOCOL	Query the availability of a certain protocol.
7	1	SVSM_CORE_CONFIGURE_VTOM	Reconfigures the use of vTOM.

6.1 Protocol Version Updates

6.1.1 Version 2

Version 2 of the Core Protocol adds support for the following:

- SVSM_CORE_PVALIDATE
  - Provides a new flag to allow the SVSM to fall back to 4K page validation when page validation for a 2M page fails because of a size mismatch with the RMP.

6.2 SVSM_CORE_REMAP_CA Call

This call is used to request that a new gPA be used for all future communication with the SVSM. It affects the Calling Area for the calling vCPU only. Table 6 describes the register usage.

Table 6: SVSM_CORE_REMAP_CA

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	4 KB	In	gPA of the new Calling Area

Upon completion of the call, the SVSM_CALL_PENDING field of the previously configured Calling Area is cleared to indicate that the call has completed. In addition, if the call is successful, the SVSM will set the SVSM_CALL_PENDING field of the new Calling Area to 0 so that a spurious invocation by an uncooperative host cannot trick the SVSM into thinking that another call was requested by the guest. The guest can examine RAX to determine whether the call was successful. If so, the previously configured Calling Area will no longer be examined by the SVSM and can be reused by the guest for any purpose.

6.3 SVSM_CORE_PVALIDATE Call

This call is used to request that the SVSM execute PVALIDATE on behalf of the guest. Table 7 describes the register usage.

Table 7: SVSM_CORE_PVALIDATE

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	8	In	gPA of the list of requested operations

Version 2 of the SVSM_CORE_PVALIDATE call allows for performing 4K page validations of a 2M page if it is detected that there is a page size mismatch with the RMP. The list of requested operations is specified according to the format in Table 8 or Table 9.

Table 8: PVALIDATE Operation (Protocol Version 1)

Byte Offset	Size (Bytes)	Meaning
0x000	2	Number of entries in the list.
0x002	2	Index of the next entry in the list to be processed.
0x004	4	Reserved - MBZ.
0x008	8	First entry in the list. Each entry specifies bits as follows: Bits 1:0 Value of RCX for the PVALIDATE operation (0=4 KB page, 1=2 MB page). Bit 2 Value of RDX for the PVALIDATE operation (0=make invalid, 1=make valid).

Byte Offset	Size (Bytes)	Meaning
		Bit 3 Ignore EFLAGS.CF warnings when set to 1. Bits 11:4 Reserved - MBZ. Bits 63:12 gPA page number. Note that bits [20:12] must be 0 if the entry describes 2 MB.
0x010	8	Second entry in the list, if any. Additional list entries follow.

Table 9: PVALIDATE Operation (Protocol Version 2)

Byte Offset	Size (Bytes)	Meaning
0x000	2	Number of entries in the list.
0x002	2	Index of the next entry in the list to be processed.
0x004	4	Reserved - MBZ.
0x008	8	First entry in the list. Each entry specifies bits as follows: Bits 1:0 Value of RCX for the PVALIDATE operation (0=4 KB page, 1=2 MB page). Bit 2 Value of RDX for the PVALIDATE operation (0=make invalid, 1=make valid). Bit 3 Ignore EFLAGS.CF warnings when set to 1. Bit 4 Fallback to validation at 4 KB page size if validation at 2 MB page size fails because of a size mismatch with the RMP. On return, this flag will be 0 if 2 MB page validation was performed and succeeded. This bit is ignored for a 4 KB page operation. Bits 11:5 Reserved - MBZ. Bits 63:12 gPA page number. Note that bits [20:12] must be 0 if the entry describes 2 MB.
0x010	8	Second entry in the list, if any. Additional list entries follow.

The number of entries in the list must not be so large that the parameter list crosses a 4 KB boundary. The number of entries must be at least 1. If the number of entries is not within a valid range, the call will return SVSM_ERR_INVALID_PARAMETER.

The index of the next entry to be processed must be strictly less than the number of entries in the list; otherwise, the call will return SVSM_ERR_INVALID_PARAMETER.

Upon completion of a call, the index of the next entry to be processed will indicate the number of entries in the list that have been successfully processed. If the call returns SVSM_ERR_INCOMPLETE, then the SVSM was unable to process the entire list in a single operation, and the guest should reload RAX with the correct calling code (RCX will remain unmodified during the call) and issue the call again; the SVSM will continue processing based on the index of the next entry to be processed. If the call returns any other error, the index of the

next entry will indicate the index of the entry that failed processing. If the call succeeds, the index of the next entry will be equal to the number of entries in the list.

The SVSM is expected to check that the guest is not attempting to execute PVALIDATE on a gPA that is assigned to the SVSM itself. If the SVSM detects that the guest is attempting to execute PVALIDATE on an address that is assigned to the SVSM, the call will return SVSM_ERR_INVALID_ADDRESS.

If an entry sets bit 2=0 (requesting invalidation of the page), the SVSM will first execute RMPADJUST to revoke permission for all VMPLs other than VMPL0. This is necessary to ensure that subsequent attempts to validate a page will observe a consistent VMPL permission state regardless of whether the host executes RMPUPDATE at any point in time.

If invocation of a PVALIDATE instruction results in the instruction completing with EFLAGS.CF=1, and if the entry that provoked the EFLAGS.CF=1 warning did not set the appropriate bit, the call will fail with the error code 0x8000_1010.

Version 2+ specific:

> If validation of a 2 MB page succeeds, the SVSM will clear bit 4 of the entry.
>
> If validation of a 2 MB page fails because of a size mismatch with the RMP and bit 4 is set, the validation will proceed at the 4K page size for all 512 4K pages within the 2M page, starting with the first page in the 2M page range. If validation of a 4K page within the 2M page fails, the GPA page number associated with the failed validation will be set in the entry GPA page number field.

If invocation of a PVALIDATE instruction (or RMPADJUST instruction) results in the instruction completing with EAX != 0, the call will fail with an error code in the range 0x8000_1000 through 0x8000_100F, where the value is equal to (0x8000_1000 + EAX). PVALIDATE is architecturally specified to return EAX error codes in the range 0x0000-0x000F; if PVALIDATE unexpectedly returns a value outside of that range (e.g., due to architectural expansion of the error code space in a future revision), the call will fail with the error code 0x8000_1011.

If invocation of PVALIDATE completes successfully, the SVSM will execute RMPADJUST to grant full permission to the VMPL of the vCPU making the request and all more privileged VMPLs (numerically lower VMPL level). Additionally, if the entry sets bit 2=1 (requesting validation of the page), the SVSM will zero out the memory for the page before performing the RMPADJUST to ensure no leakage of VMPL0 memory to the lower VMPL.

6.4 SVSM_CORE_CREATE_VCPU Call

This call is used to request creation of a new vCPU context. Table 10 describes the register usage.

Table 10: SVSM_CORE_CREATE_VCPU

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	4 KB	In	gPA of the VMSA to be created for the vCPU
RDX	8	4 KB	In	gPA of the Calling Area associated with the vCPU
R8	4		In	APIC ID of the vCPU

The SVSM will check that both gPA values specified do not conflict with any existing usage (SVSM private pages, any active VMSA address or any active Calling Area address). If a conflict is detected, the call will return SVSM_ERR_INVALID_ADDRESS.

The SVSM will revoke access to every VMPL other than itself to ensure that the VMSA page is not tampered with during the process of VMSA validation and assignment.

The SVSM will validate the new VMSA:

- If VMSA.VMPL == 0, the call will return SVSM_ERR_INVALID_PARAMETER.
- If VMSA.VMPL < vCPU.VMPL, the call will return SVSM_ERR_INVALID_PARAMETER.
- If VMSA.EFER.SVME != 1, the call will return SVSM_ERR_INVALID_PARAMETER.
- If VMSA.SEV_FEATURES != startup vCPU VMSA.SEV_FEATURES, the call will return SVSM_ERR_INVALID_PARAMETER.

If the VMSA checks pass, the SVSM will execute RMPADJUST to turn the page into a VMSA page so it can be used immediately. The SVSM will cache the gPA of the VMSA and the gPA of the Calling Area associated with the target vCPU for use in subsequent calls to the SVSM (register input, protocol request, etc.).

Once a page is established as a VMSA page, it is treated as privately owned by the SVSM for the purpose of detecting memory usage conflicts. Any call which specifies the gPA of a VMSA page as an input gPA will fail with SVSM_ERR_INVALID_ADDRESS. This is also true of the VMSA of the startup vCPU. (This VMSA is not required to be within the initial contiguous range of pages assigned to the SVSM since the guest is expected to know where its own VMSA is located.)

6.5 SVSM_CORE_DELETE_VCPU Call

This call is used to delete a vCPU that was previously configured. Table 11 describes the register usage.

Table 11: SVSM_CORE_DELETE_VCPU

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	4 KB	In	gPA of the VMSA to be deleted

The SVSM will verify that the specified gPA belongs to a known VMSA address. If the gPA is not a known VMSA address, the call will return SVSM_ERR_INVALID_PARAMETER. The startup vCPU can never be deleted. If the VMSA is associated with the startup vCPU, the call will return SVSM_ERR_INVALID_PARAMETER. If the VMPL of the VMSA is less than the VMPL of the requester, the call will return SVSM_ERR_INVALID_PARAMETER.

The SVSM will set VMSA.EFER.SVME = 0. If the SVSM fails to set VMSA.EFER.SVME to 0 (because the VMSA is currently executing), the call will return a result value of 0x80001003 (equivalent to the FAIL_INUSE error code).

The SVSM will execute RMPADJUST to convert the page from a VMSA page to a normal page and ensure that full access is available to the VMPL of the requesting vCPU and all more privileged VMPLs (numerically lower than or equal to the requesting VMPL).

Once deletion of the vCPU is complete, the previously configured VMSA and Calling Area will no longer be examined by the SVSM and can be reused by the guest for any purpose. If the VMSA was the VMSA of the requester, the SVSM will not return to the caller.

6.6 SVSM_CORE_DEPOSIT_MEM Call

This call can be used to grant additional memory for exclusive use by the SVSM in case it requires additional memory to perform its work. The guest can know when additional memory is required because the SVSM will return a status code in the range 0x4nnn_nnnn, indicating the number of additional 4 KB pages required. Table 12 describes the register usage.

The caller is responsible for coordinating calls to deposit or withdraw memory across all vCPUs.

Table 12: SVSM_CORE_DEPOSIT_MEM

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	8	In	gPA of the list of memory pages

The list of memory pages is specified according to the format in Table 13.

Table 13: Deposit Memory Operation

Byte Offset	Size	Meaning
0x000	2 bytes	Number of entries in the list.
0x002	2 bytes	Index of the next entry in the list to be processed.
0x004	4 bytes	Reserved - MBZ.
0x008	8 bytes	First entry in the list. Each entry specifies bits as follows: Bits 1:0 Size of the memory range described by this entry (0=4 KB page, 1=2 MB page). Bits 11:2 Reserved - MBZ. Bits 63:12 gPA page number. Note that bits [20:12] must be 0 if the entry describes 2 MB.
0x010	8 bytes	Second entry in the list, if any. Additional list entries follow.

The number of entries in the list must not be so large that the parameter list crosses a 4 KB boundary. The number of entries must be at least 1. If number of entries is not within a valid range, the call will return SVSM_ERR_INVALID_PARAMETER.

The index of the next entry to be processed must be strictly less than the number of entries in the list; otherwise, the call will return SVSM_ERR_INVALID_PARAMETER.

Upon completion of a call, the index of the next entry to be processed will indicate the number of entries in the list that have been successfully processed. If the call returns SVSM_ERR_INCOMPLETE, then the SVSM was unable to process the entire list in a single operation, and the guest should reload RAX with the correct calling code (RCX will remain unmodified during the call) and issue the call again; the SVSM will continue processing based on the index of the next entry to be processed. If the call returns any other error, the index of the next entry will indicate the index of the entry that failed processing. If the call succeeds, the index of the next entry will be equal to the number of entries in the list.

For each entry in the list, the SVSM will verify that the memory described is not already private to the SVSM and does not overlap any page that has been configured as a Calling Area. If any overlap is detected, the call will return SVSM_ERR_INVALID_ADDRESS.

For each valid entry in the list, the SVSM will execute RMPADJUST to restrict VMPL permissions so that the pages are only accessible to VMPL0, making the pages private to the SVSM.

The call may return failure with the value of the next entry index not being 0. This indicates that some memory was successfully deposited with the SVSM, and some was not.

6.7 SVSM_CORE_WITHDRAW_MEM Call

This call permits the guest to reclaim memory that was made private to the SVSM but is no longer needed by the SVSM. Any SVSM operation that results in free memory that can be reclaimed will set the SVSM_MEM_AVAILABLE flag in the Calling Area. Table 14 describes the register usage.

The caller is responsible for coordinating calls to deposit or withdraw memory across all vCPUs.

Table 14: SVSM_CORE_WITHDRAW_MEM

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	8	In	gPA of an area to hold a list of memory pages

The list of memory pages is populated by the SVSM and is specified according to the format in Table 15.

Table 15: Withdraw Memory Operation

Byte Offset	Size	Meaning
0x000	2 bytes	Number of entries in the list.
0x002	6 bytes	Unused.
0x008	8 bytes x number of entries	List of gPA values of 4 KB pages that are no longer in use.

The maximum number of entries returned is constrained so that the returned list will not cross a 4 KB boundary. If the parameter page is aligned such that there is no room for any entries (i.e., the parameter page gPA is aligned to a 4 KB boundary plus 0xFF8 bytes), the call will return SVSM_ERR_INVALID_PARAMETER.

If no memory is available to withdraw, the number of entries will be 0. Unused entries beyond the end of the list are not zeroed.

The SVSM must execute RMPADJUST for all memory being withdrawn to grant full access to the VMPL of the requesting vCPU and to all more privileged VMPLs (numerically lower than or equal to the requesting VMPL). Upon completion of the call, the SVSM will no longer access the pages, which can then be reused by the guest for any purpose.

The SVSM_MEM_AVAILABLE flag of the Calling Area of the startup vCPU may be updated to indicate whether additional memory remains available for withdrawal.

This call will never return SVSM_ERR_INCOMPLETE. If the SVSM is unable to withdraw all available memory, the call must complete with SVSM_SUCCESS. The SVSM_MEM_AVAILABLE flag of the startup vCPU's Calling Area will indicate that additional memory remains available for withdrawal.

6.8 SVSM_CORE_QUERY_PROTOCOL Call

This call is used to determine the availability of a given protocol. Bits [63:32] of register RCX contain the requested protocol number. Bits [31:0] of register RCX contain the desired version of the requested protocol. Upon completion of the call, register RCX is set to indicate availability of the requested protocol. If the protocol is unavailable at the requested version, register RCX will contain the value 0. If the protocol is available at the requested version, bits [63:32] of register RCX will support the maximum supported protocol version number, and bits [31:0] of register RCX will support the minimum supported protocol version number.

This call will always return SVSM_SUCCESS since the availability of the protocol is advertised through RCX. Querying for the presence of a protocol is not permitted to demand additional SVSM memory. (Calls to that protocol may request memory to be deposited.)

6.9 SVSM_CORE_CONFIGURE_VTOM Call

This call is used to query or reconfigure the use of vTOM on the calling processor. To support a transition between vTOM-based confidentiality and confidentiality determinations that rely exclusively on the Page Table Entry's C-bit, this call will also change the guest value of CR3, as well as RIP and RSP, to permit a clean transition from one environment to another.

SVSM_CORE_CONFIGURE_VTOM calls take two forms: query and configure, as indicated by bit 0 of RCX. (RCX[0]=1 indicates query, while RCX[0]=0 indicates configure.) Table 16 describes the register usage.

Table 16: vTOM Configuration Operation

Register	Contents		
RCX on entry	Query	Bit 0	Must be 1.
		Bit 63:1	Must be 0.

Register	Contents		
	Configure	Bit 0	Must be 0.
		Bit 1	Set to 0 to disable vTOM, or set to 1 to enable vTOM.
		Bit 2	If set to 1, will cause VMSA.CR3 to be set to the value in RDX upon successful completion of the call.
		Bit 3	If set to 1, will cause VMSA.RIP to be set to the value in R8 upon successful completion of the call.
		Bit 4	If set to 1, will cause VMSA.RSP to be set to the value in R9 upon successful completion of the call.
		Bits 11:5	Must be 0.
		Bits 63:12	Bits 63:12 of the desired vTOM value. Must be 0 if vTOM is being disabled.
RCX result	Bit 0	Will be 0.	
	Bit 1	Will be 1 if vTOM configuration is supported; otherwise 0.	
	Bits 11:2	Will be 0.	
	Bits 19:12	vTOM alignment requirement as a power of two (value of 20 would indicate that vTOM must be aligned to a 1 MB boundary).	
	Bits 63:20	0.	
RDX result	Minimum valid vTOM value if vTOM configuration is supported; otherwise undefined.		
R8 result	Maximum valid vTOM value if vTOM configuration is supported; otherwise undefined.		

If the call is successful, vTOM will be reported or reconfigured as requested. If vTOM is being reconfigured, then CR3 and RSP will be updated as requested, and execution will continue at the specified RIP with RAX containing the value SVSM_SUCCESS.

If the call is unsuccessful, no VMSA changes will occur, and execution will continue at the instruction following the VMGEXIT with RAX containing the appropriate error code.

The call may fail if any reserved bit RCX is set inappropriately; this will result in the call returning SVSM_ERR_INVALID_PARAMETER.

The SVSM may choose to deny the call if it cannot support the request. For example, the SVSM may be unable to reconfigure VTOM if more than a single vCPU has been configured or if the requested configuration differs from configurations present on other vCPUs. This will result in the call returning SVSM_ERR_INVALID_REQUEST. If the value of vTOM is one that cannot be supported by the hosting environment, then the call will result in SVSM_ERR_INVALID_ADDRESS.

7 Attestation Protocol

The attestation protocol has protocol ID value 1. The attestation protocol is versioned, permitting its extension over time; the initial version of the attestation protocol is version 1. Versioning of the attestation protocol is strictly additive, i.e., all calls present in version 1 must be supported by all future implementations. Table 17 enumerates the set of calls supported by the attestation protocol.

Table 17: Attestation Protocol Services

Call ID	First version supported	Name	Function
0	1	SVSM_ATTEST_SERVICES	Retrieve an attestation report for all SVSM services (e.g. vTPM, etc.).
1	1	SVSM_ATTEST_SINGLE_SERVICE	Retrieve an attestation report for a single SVSM service
2	2	SVSM_ATTEST_SINGLE_SERVICE_EXT	Retrieve an attestation report for a single SVSM service based on supplied input

7.1 SVSM_ATTEST_SERVICES Call

This call is used to request a VMPL0 attestation report that includes a services manifest of the services that are running in the SVSM as part of the report data. Optionally, a certificate data buffer can be supplied. When supplied, the hypervisor can provide certificate data back to the caller when processing the SNP Guest Request used to generate the attestation report. Table 18 describes the register usage.

Table 18: SVSM_ATTEST_SERVICES

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	8	In	gPA of the attestation services operation structure (see Table 19)
	8		Out	Services manifest size (in bytes)
RDX	8		Out	Certificate data size (in bytes)
R8	8		Out	Attestation report size (in bytes)

The inputs associated with the attest services call are specified according to the format in Table 19.

Table 19: Attest Services Operation

Byte Offset	Size (Bytes)	Alignment (Bytes)	Meaning
0x000	8	4 KB	Attestation report buffer gPA
0x008	4		Attestation report buffer size (in bytes)
0x00C	4		RESERVED – MBZ
0x010	8		Nonce gPA
0x018	2		Nonce size (in bytes)
0x01A	6		RESERVED – MBZ
0x020	8	4 KB	Services manifest buffer gPA
0x028	4		Services manifest buffer size (in bytes)
0x02C	4		RESERVED – MBZ
0x030	8	4 KB	Certificate data buffer gPA
0x038	4		Certificate data buffer size (in bytes)
0x03C	4		RESERVED – MBZ

The attest services operation structure must not cross a 4 KB boundary. If the gPA of the structure is such that the structure crosses a 4 KB boundary, the call will return SVSM_ERR_INVALID_PARAMETER.

The attestation report buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

The nonce must not cross a 4 KB boundary. If the nonce crosses a 4 KB boundary, the call will return SVSM_ERR_INVALID_PARAMETER.

The services manifest buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

The certificate data buffer is optional. Its presence is indicated by setting the certificate data buffer size to a non-0 value. If the certificate data buffer length is non-0, the certificate data buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

All input gPA values must not be values that are assigned to the SVSM itself. If the SVSM detects that the guest is specifying an input gPA that is assigned to the SVSM, the call will return SVSM_ERR_INVALID_ADDRESS.

The SVSM will assemble a services manifest that will be used as input to the attestation request. Each service will produce a descriptive section for the manifest in a service-defined format. If the size of the assembled services manifest exceeds the size of the supplied services manifest buffer,

RCX will be set to the size of the services manifest (in bytes) and the call will return SVSM_ERR_INVALID_PARAMETER.

The services manifest is identified by a 16-byte GUID, 4-byte length and a 4-byte services count at the beginning of the manifest. The services in the manifest, if any, are identified in a table beginning at offset 24 (0x18) into the manifest. Each service entry consists of a 16-byte GUID, a 4-byte offset from the start of the manifest to the service data, and a 4-byte length of the service data. Table 20 describes the Services Manifest. A given SVSM binary must produce a services manifest with the services ordered and assembled in the same manner across multiple calls to SVSM_ATTEST_SERVICES and across different VM instances.

All attestation GUIDs defined in this document are formatted as described in RFC 4122, however, the time_low, time_mid, and time_hi_and_version fields are encoded in little-endian format.

Table 20: Services Manifest

Byte Offset	Size (Bytes)	Meaning
0x000	16	Services manifest GUID: 63849ebb-3d92-4670-a1ff-58f9c94b87bb
0x010	4	Services manifest length
0x014	4	Number of services contained in the manifest
First service table entry, if any.		
0x018	16	Service GUID
0x028	4	Service data offset
0x02C	4	Service data length
Next service table entry, if any. Additional table service table entries follow.		
0x030		

The minimum length of a services manifest (when there are no services present) is 24 (0x18) bytes.

Each service will document its GUID value and the format of its manifest content. It is suggested to use the same GUID/offset/length format as is used here.

The Input REPORT_DATA supplied on the SNP attestation request will be the SHA-512 digest of the input nonce and the services manifest, SHA-512(Nonce || Services Manifest). The input VMPL supplied on the SNP attestation request will be 0.

If a certificate data buffer was provided and if the size of the certificate data from the hypervisor exceeds the size of the supplied certificate data buffer, RCX will be set to the size of the services manifest and RDX will be set to the size of the certificate data (in bytes) and the call will return SVSM_ERR_INVALID_PARAMETER.

If the size of the SNP attestation report exceeds the size of the supplied attestation report buffer, RCX will be set to the size of the services manifest, RDX will be set to the size of the certificate data (in bytes, if a certificate buffer was supplied) and R8 will be set to the size of the attestation report (in bytes) and the call will return SVSM_ERR_INVALID_PARAMETER.

Upon successful completion of the SNP attestation request, the attestation report will be copied to the input attestation report buffer gPA, the services manifest will be copied to the input services manifest buffer gPA, RCX will be set to the size of the services manifest and, if a certificate data buffer was provided, the certificate data will be copied to the input certificate data buffer gPA and RDX will be set to the size of the certificate data. Should the SNP attestation request fail, RAX will be set to 0x8000_1000.

7.2 SVSM_ATTEST_SINGLE_SERVICE Call

This call is used to request a VMPL0 attestation report that includes a service manifest for the specified service that is running in the SVSM as part of the report data. Table 21 describes the register usage.

Table 21: SVSM_ATTEST_SINGLE_SERVICE

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	8	In	gPA of the attestation single service operation structure (see Table 22)
	8		Out	Service manifest size (in bytes)
RDX	8		Out	Certificate data size (in bytes)
R8	8		Out	Attestation report size (in bytes)

The inputs associated with the attest single service call are specified according to the format in Table 22.

Table 22: Attest Single Service Operation

Byte Offset	Size (Bytes)	Alignment (Bytes)	Meaning
0x000	8	4 KB	Attestation report buffer gPA
0x008	4		Attestation report buffer size (in bytes)
0x00C	4		RESERVED – MBZ
0x010	8		Nonce gPA
0x018	2		Nonce size (in bytes)
0x01A	6		Reserved
0x020	8	4 KB	Service manifest buffer gPA

Byte Offset	Size (Bytes)	Alignment (Bytes)	Meaning
0x028	4		Service manifest buffer size (in bytes)
0x02C	4		RESERVED – MBZ
0x030	8	4 KB	Certificate data buffer gPA
0x038	4		Certificate data buffer size (in bytes)
0x03C	4		RESERVED – MBZ
0x040	16		GUID of service to attest
0x050	4		Requested manifest version
0x054	4		RESERVED – MBZ

The attest single service operation structure must not cross a 4 KB boundary. If the gPA of the structure is such that the structure crosses a 4 KB boundary, the call will return SVSM_ERR_INVALID_PARAMETER.

The attestation report buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

The nonce must not cross a 4 KB boundary. If the nonce crosses a 4 KB boundary, the call will return SVSM_ERR_INVALID_PARAMETER.

The service manifest buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

The certificate data buffer is optional. Its presence is indicated by setting the certificate data buffer size to a non-0 value. If the certificate data buffer length is non-0, the certificate data buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

All input gPA values must not be values that are assigned to the SVSM itself. If the SVSM detects that the guest is specifying an input gPA that is assigned to the SVSM, the call will return SVSM_ERR_INVALID_ADDRESS.

The GUID of the service to be attested needs to be an available service of the SVSM. If the requested service is not available, the call will return SVSM_ERR_INVALID_PARAMETER.

The SVSM service must support the version of the manifest requested. If the requested manifest version is not supported, the call will return SVSM_ERR_INVALID_PARAMETER.

The SVSM will assemble a service manifest that will be used as input to the attestation request. The service will produce a descriptive manifest in a service-defined format. If the size of the assembled service manifest exceeds the size of the supplied service manifest buffer, RCX will be set to the size of the service manifest (in bytes) and the call will return SVSM_ERR_INVALID_PARAMETER.

The Input REPORT_DATA supplied on the SNP attestation request will be the SHA-512 digest of the input nonce and the service manifest, SHA-512(Nonce || Service Manifest). The input VMPL supplied on the SNP attestation request will be 0.

If a certificate data buffer was provided and if the size of the certificate data from the hypervisor exceeds the size of the supplied certificate data buffer, RCX will be set to the size of the services manifest and RDX will be set to the size of the certificate data (in bytes) and the call will return SVSM_ERR_INVALID_PARAMETER.

If the size of the SNP attestation report exceeds the size of the supplied attestation report buffer, RCX will be set to the size of the services manifest, RDX will be set to the size of the certificate data (in bytes, if a certificate buffer was supplied) and R8 will be set to the size of the attestation report (in bytes) and the call will return SVSM_ERR_INVALID_PARAMETER.

Upon successful completion of the SNP attestation request, the attestation report will be copied to the input attestation report buffer gPA, the service manifest will be copied to the input service manifest buffer gPA, RCX will be set to the size of the service manifest, if a certificate data buffer was provided, the certificate data will be copied to the input certificate data buffer gPA and RDX will be set to the size of the certificate data. Should the SNP attestation request fail, RAX will be set to 0x8000_1000.

8 vTPM Protocol

The vTPM protocol has protocol ID value 2. The vTPM protocol is versioned, permitting its extension over time; the initial version of the vTPM protocol is version 1. Versioning of the vTPM protocol is strictly additive, i.e., all calls present in version 1 must be supported by all future implementations.

The vTPM protocol follows the *Official TPM 2.0 Reference Implementation (by Microsoft)* simulator protocol.

Table 23 enumerates the set of calls supported by the vTPM protocol.

Table 23: vTPM Protocol Services

Call ID	First version supported	Name	Function
0	1	SVSM_VTPM_QUERY	Query vTPM command and feature support.
1	1	SVSM_VTPM_CMD	Execute a TPM command.

8.1 SVSM_VTPM_QUERY Call

This call is used to query the support provided by the vTPM. Table 24 describes the register usage.

Table 24: SVSM_VTPM_QUERY

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8		Out	Supported vTPM platform commands
RDX	8		Out	Supported vTPM features (see Table 25)

RCX is used to indicate the supported platform commands. For each platform command supported by the vTPM, the corresponding bit will be set in RCX. Bits for any unsupported and undefined platform commands must be cleared. The platform command values follow the values used by the *Official TPM 2.0 Reference Implementation (by Microsoft)* simulator protocol.

Table 25: vTPM Features

Bit	Feature	Description
63:0		Must-be-0

8.2 SVSM_VTPM_CMD Call

This call is used to execute a vTPM operation. Table 26 describes the register usage.

Table 26: SVSM_VTPM_CMD

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	4 KB	In	gPA of the vTPM request/response structure

All command request buffers have a common structure as specified in Table 27.

Table 27: vTPM Common Request/Response Structure

Byte Offset	Size (Bytes)	In/Out	Description
0x000	4	In	Platform command
		Out	Platform command response size

Each command can build upon this common request/response structure to create a structure specific to the command. The commands that do this are:

- TPM_SIGNAL_HASH_DATA
- TPM_SEND_COMMAND
- TPM_REMOTE_HANDSHAKE
- TPM_SET_ALTERNATIVE_RESULT

Only the TPM_SEND_COMMAND will be documented in this specification.

If a vTPM request/response structure has not been supplied, the call will return SVSM_ERR_INVALID_PARAMETER. It is expected that the request/response structure is large enough to hold the expected output of the vTPM request. The vTPM request/response buffer will be treated as physically contiguous in the guest address space.

The vTPM request/response structure gPA must not be gPA values that is assigned to the SVSM itself. If the SVSM detects that the guest is specifying an address that is assigned to the SVSM, the call will return SVSM_ERR_INVALID_ADDRESS.

If the specified platform command is not supported, the call will return SVSM_ERR_INVALID_PARAMETER.

8.2.1 TPM_SEND_COMMAND

Execute a TPM command and return the results.

For TPM_SEND_COMMAND, platform command 8, the request buffer is specified according to the format Table 28.

Table 28: TPM_SEND_COMMAND Request Structure

Byte Offset	Size (Bytes)	Meaning
0x000	4	Platform command (8)
0x004	1	Locality (must-be-0)
0x005	4	TPM Command size (in bytes)
0x009	Variable	TPM Command

The response buffer is specified according to the format in Table 29.

Table 29: TPM_SEND_COMMAND Response Structure

Byte Offset	Size (Bytes)	Meaning
0x000	4	Response size (in bytes)
0x004	Variable	Response

Locality usage for the vTPM is not defined. If a Locality value other than 0 is specified, the call will return SVSM_ERR_INVALID_PARAMETER.

8.3 Service Attestation Data

8.3.1 Service Attestation GUID

The vTPM service attestation GUID is c476f1eb-0123-45a5-9641-b4e7dde5bfe3.

8.3.2 SVSM_ATTEST_SERVICES Manifest Data

The manifest data used in an SVSM_ATTEST_SERVICES call is the same format as the manifest version 0 format of the SVSM_ATTEST_SINGLE_SERVICE call, see section 8.3.3.1.

8.3.3 SVSM_ATTEST_SINGLE_SERVICE Manifest Data

8.3.3.1 Manifest Version 0

The manifest data used in an SVSM_ATTEST_SINGLE_SERVICE call for manifest version 0 is specified according to Table 30.

Table 30: vTPM Service Manifest Data Structure (Version 0)

Byte Offset	Size (Bytes)	Meaning
0x000	Variable	TPMT_PUBLIC structure of the endorsement key

8.3.3.2 Manifest Version 1

The manifest data used in an SVSM_ATTEST_SINGLE_SERVICE call for manifest version 1 is specified according to Table 31.

Table 31: vTPM Service Manifest Data Structure (Version 1)

Byte Offset	Size (Bytes)	Meaning
0x000	4	Version - 1
0x004	4	Number of TPM2B_PUBLIC structures present
0x008	Variable	TPM2B_PUBLIC structures representing storage root keys and attestation (signing) keys

9 APIC Emulation Protocol

The APIC Emulation protocol has protocol ID value 3. The APIC Emulation protocol is versioned, permitting its extension over time; the initial version of the APIC Emulation protocol is version 1. Versioning of the APIC Emulation protocol is strictly additive, i.e., all calls present in version 1 must be supported by all future implementations. The APIC Emulation protocol is used to support Alternate Injection with the guest. Table 32 enumerates the set of calls supported by the APIC emulation protocol.

Table 32: APIC Emulation Protocol Services

Call ID	First version supported	Name	Function
0	1	SVSM_APIC_QUERY_FEATURES	Query supported APIC features.
1	1	SVSM_APIC_CONFIGURE_EMULATION	Control usage of APIC emulation.
2	1	SVSM_APIC_READ_REGISTER	Read an APIC register.
3	1	SVSM_APIC_WRITE_REGISTER	Write an APIC register.
4	1	SVSM_APIC_CONFIGURE_VECTOR	Configure interrupt vector.

9.1 SVSM_APIC_QUERY_FEATURES Call

This call is used to determine the APIC features supported by the APIC emulation protocol. The APIC emulation protocol supports basic APIC functionality related to interrupt delivery. Table 33 describes the register usage.

Table 33: SVSM_APIC_QUERY_FEATURES

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8		Out	Additional Supported Features (see Table 35)

APIC emulation supports the APIC registers defined in Table 34.

Table 34: Supported APIC Registers

MSR Address	APIC Register Name	Access
0x802	x2APIC ID	RO
0x808	Task Priority Register (TPR)	R/W
0x80A	Processor Priority Register (PPR)	RO
0x80B	End of Interrupt Register (EOI)	WO
0x80D	Logical Destination Register (LDR)	RO
0x810 – 0x817	In-Service Register (ISR)	RO

MSR Address	APIC Register Name	Access
0x818 – 0x81F	Trigger Mode Register (TMR)	RO
0x820 – 0x827	Interrupt Request Register (IRR)	RO
0x830	Interrupt Command Register (ICR)	R/W
0x83F	Self IPI Register	WO

Only IPI types of Fixed and NMI are part of the basic APIC emulation protocol.

Table 35: APIC Additional Support Features

<table>
<tr><th>Bits</th><th>Meaning</th></tr>
<tr><td>63:2</td><td>Reserved – MBZ.</td></tr>
<tr><td>1</td><td>0 – INIT and SIPI message delivery is not supported.
1 – INIT and SIPI message delivery is supported.</td></tr>
<tr><td>0</td><td>0 – Timer functionality is not supported.
The guest must use the GHCB #HV Timer NAE event.

1 – Timer functionality is supported.
The following additional APIC registers are supported:
<table>
<tr><th>MSR Address</th><th>APIC Register Name</th><th>Access</th></tr>
<tr><td>0x832</td><td>Timer Local Vector Table (LVT) Entry</td><td>R/W</td></tr>
<tr><td>0x838</td><td>Timer Initial Count Register</td><td>R/W</td></tr>
<tr><td>0x839</td><td>Timer Current Count Register</td><td>RO</td></tr>
<tr><td>0x83E</td><td>Timer Divide Configuration Register</td><td>R/W</td></tr>
</table></td></tr>
</table>

The SVSM will validate that Alternate Injection is enabled for the guest. If Alternate Injection is not enabled, the call will return SVSM_ERR_UNSUPPORTED_PROTOCOL.

The SVSM will initialize RCX to 0. If the SVSM supports APIC emulation timer functionality, then RCX[0] is set to 1. If the SVSM support INIT and SIPI message delivery, then RCX[1] is set to 1.

9.2 SVSM_APIC_CONFIGURE_EMULATION Call

This call is used to control whether the guest can use the APIC emulation support. The APIC emulation protocol must support multiple guest runtimes (UEFI, OS, etc.) and therefore must be capable of disabling the protocol if needed. Table 36 describes the register usage.

Table 36: SVSM_APIC_CONFIGURE_EMULATION

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8		In	Emulation configuration data (see Table 37)

Table 37: Emulation Configuration Data

Bits	Meaning
63:2	Reserved – MBZ.
1:0	• 0b00 – QUERY_APIC_EMULATION Query APIC emulation registration count. • 0b01 – UNREGISTER_APIC_EMULATION_USE Decrement the APIC emulation registration count. • 0b10 – REGISTER_APIC_EMULATION_USE Increment the APIC emulation registration count. • 0b11 Invalid

The SVSM will validate that Alternate Injection is enabled for the guest. If Alternate Injection is not enabled, the call will return SVSM_ERR_UNSUPPORTED_PROTOCOL.

The SVSM will validate that the emulation configuration data is valid. If the emulation configuration data is not valid, the call will return SVSM_ERR_INVALID_PARAMETER.

The SVSM will configure APIC emulation based on the requested operation:

- QUERY_APIC_EMULATION
  If the APIC emulation registration count is 0, then Alternate Injection support is disabled on the calling vCPU and across all vCPUs of the guest.

- UNREGISTER_APIC_EMULATION_USE
  If the current APIC emulation registration count is 0, then the count is not decremented. If the current APIC emulation registration count is not 0, then the APIC emulation registration count is decremented by 1.

  If the new APIC emulation registration count is 0, then Alternate Injection support is disabled on the calling vCPU and across all vCPUs of the guest.

- REGISTER_APIC_EMULATION_USE
  If the current APIC emulation registration count is 0, the call will return the protocol error code 0x8000_1000 (SVSM_ERR_APIC_CANNOT_REGISTER).

  If the current APIC emulation registration count is not 0, then the APIC emulation registration count is incremented by 1.

9.3 SVSM_APIC_READ_REGISTER Call

This call is used to read an APIC register supported by the APIC emulation protocol. Table 38 describes the register usage.

Table 38: SVSM_APIC_READ_REGISTER

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	4		In	APIC MSR address
RDX	8		Out	APIC MSR address value

The SVSM will validate that Alternate Injection is enabled for the guest. If Alternate Injection is not enabled, the call will return SVSM_ERR_UNSUPPORTED_PROTOCOL.

The SVSM will validate that the requested APIC MSR address is supported and accessed correctly. If the APIC MSR address is not supported, the call will return SVSM_ERR_INVALID_ADDRESS. If the APIC MSR address is write-only (WO), the call will return SVSM_ERR_INVALID_PARAMETER.

RDX will be set to the emulated value of the specified APIC MSR address.

9.4 SVSM_APIC_WRITE_REGISTER Call

This call is used to write an APIC register supported by the APIC emulation protocol. Table 39 describes the register usage.

Table 39: SVSM_APIC_WRITE_REGISTER

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	4		In	APIC MSR address
RDX	8		In	APIC MSR address value

The SVSM will validate that Alternate Injection is enabled for the guest. If Alternate Injection is not enabled, the call will return SVSM_ERR_UNSUPPORTED_PROTOCOL.

The SVSM will validate that the requested APIC MSR address is supported and accessed correctly. If the APIC MSR address is not supported, the call will return SVSM_ERR_INVALID_ADDRESS. If the APIC MSR address is read-only (RO), the call will return SVSM_ERR_INVALID_PARAMETER. The SVSM will validate the APIC MSR address value. If any reserved bits in the APIC MSR value

that must-be-zero (MBZ) are not zero, the call will return SVSM_ERR_INVALID_PARAMETER. If the APIC MSR value is not valid for the APIC MSR address (for example, 0 is the only valid value that can be written to the APIC EOI MSR), the call will return SVSM_ERR_INVALID_PARAMETER.

The specified APIC MSR address will be set to the RDX value.

9.5 SVSM_APIC_CONFIGURE_VECTOR Call

This call allows the guest to configure the permissibility of an interrupt vector. Table 40 describes the register usage.

Table 40: SVSM_APIC_CONFIGURE_VECTOR

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8		In	Vector configuration data (see Table 41)

Table 41: Vector Configuration Data

Bits	Meaning
63:10	Reserved – MBZ.
9	0 – Action applies to the vector specified in bits 7:0 1 – Action applies to all vectors
8	0 – Target vector(s) are permitted to be delivered 1 – Target vector(s) are not permitted to be delivered
7:0	Vector • Allowed values are 2, 31 – 255 • Ignored if bit 9 is set

The SVSM will validate that Alternate Injection is enabled for the guest. If Alternate Injection is not enabled, the call will return SVSM_ERR_UNSUPPORTED_PROTOCOL.

The SVSM will validate that the emulation configuration data is valid. If the emulation configuration data is not valid, the call will return SVSM_ERR_INVALID_PARAMETER.

The SVSM will record the target vector(s) delivery permissibility.

9.6 Service Attestation Data

9.6.1 Service Attestation GUID

The APIC Emulation service attestation GUID is a1a3ede3-bae9-42c1-97b5-6b75133af830.

9.6.2 SVSM_ATTEST_SERVICES Manifest Data

The manifest data used in an SVSM_ATTEST_SERVICES call is the same format as the manifest version 0 format of the SVSM_ATTEST_SINGLE_SERVICE call, see section 9.6.3.1.

9.6.3 SVSM_ATTEST_SINGLE_SERVICE Manifest Data

9.6.3.1 Manifest Version 0

The manifest data used in an SVSM_ATTEST_SINGLE_SERVICE call for manifest version 0 is specified according to Table 42.

Table 42: APIC Emulation Service Manifest Data Structure

<table>
<tr><th>Byte Offset</th><th>Size (Bytes)</th><th>Meaning</th></tr>
<tr><td>0x000</td><td>4</td><td>Version - 0</td></tr>
<tr><td>0x004</td><td>4</td><td>Flags

<table><tr><th>Bits</th><th>Meaning</th></tr><tr><td>31:2</td><td>Reserved – MBZ.</td></tr><tr><td>1</td><td>INIT and SIPI message delivery supported</td></tr><tr><td>0</td><td>Timer functionality supported</td></tr></table></td></tr>
</table>

10 UEFI Management Mode Protocol

The UEFI Management Mode protocol has protocol ID value 4. The UEFI Management Mode protocol is versioned, permitting its extension over time; the initial version of the UEFI Management Mode protocol is version 1. Versioning of the UEFI Management Mode protocol is strictly additive, i.e., all calls present in version 1 must be supported by all future implementations.

Table 43 enumerates the set of calls supported by the UEFI Management Mode protocol.

Table 43: UEFI Management Mode Protocol Services

Call ID	First version supported	Name	Function
0	1	SVSM_UEFI_MM_REQUEST	Process the request in the communication buffer

10.1 SVSM_UEFI_MM_REQUEST Call

Request the UEFI Management Mode service to process a request contained in communication buffer. Table 44 describes the register usage.

Table 44: SVSM_UEFI_MM_REQUEST

Register	Size (Bytes)	Alignment (Bytes)	In/Out	Description
RAX	4		Out	Result value
RCX	8	4 KB	In	gPA of the request buffer
RDX	4		In	Size of the request buffer

The request buffer will be treated as physically contiguous in the guest address space if the buffer size is greater than 4 KB.

The request buffer gPA value must not be a gPA value that is assigned to the SVSM itself. If the SVSM detects that the guest is specifying an address that is assigned to the SVSM, the call will return SVSM_ERR_INVALID_ADDRESS.

The format of request buffer input and output is defined by EDK2. Output from the request is returned in the same buffer in which the request was made.

10.2 Service Attestation Data

10.2.1 Service Attestation GUID

The UEFI Management Mode service attestation GUID is a4453a59-9e1b-4787-a033-1986d6adbe55.

10.2.2 SVSM_ATTEST_SERVICES Manifest Data

The manifest data used in an SVSM_ATTEST_SERVICES call is the same format as the manifest version 0 format of the SVSM_ATTEST_SINGLE_SERVICE call, see section 10.2.3.1.

10.2.3 SVSM_ATTEST_SINGLE_SERVICE Manifest Data

10.2.3.1 Manifest Version 0

The manifest data used in an SVSM_ATTEST_SINGLE_SERVICE call for manifest version 0 is specified according to Table 45.

Table 45: UEFI Management Mode Service Manifest Data Structure

<table>
<tr><th>Byte Offset</th><th>Size (Bytes)</th><th>Meaning</th></tr>
<tr><td>0x000</td><td>4</td><td>Version - 0</td></tr>
<tr><td>0x004</td><td>4</td><td>Flags
<table>
<tr><th>Bits</th><th>Meaning</th></tr>
<tr><td>31:3</td><td>Reserved – MBZ.</td></tr>
<tr><td>2</td><td>Secure boot databases are updatable by the guest (db/dbx).</td></tr>
<tr><td>1</td><td>Secure boot is enabled.</td></tr>
<tr><td>0</td><td>Non-volatile UEFI variables are written to persistent storage.</td></tr>
</table></td></tr>
</table>