

[AMD Public Use]



Secure Encrypted Virtualization API Version 0.24

Technical Preview

Publication #	55766	Revision:	3.24
Issue Date:	April 2020		

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.

© 2016-2020 Advanced Micro Devices, Inc. All rights reserved.

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, 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.

Reverse engineering or disassembly is prohibited.

Dolby Laboratories, Inc.

Manufactured under license from Dolby Laboratories.

Rovi Corporation

This device is protected by U.S. patents and other intellectual property rights. The use of Rovi Corporation's copy protection technology in the device must be authorized by Rovi Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by Rovi Corporation.

USE OF THIS PRODUCT IN ANY MANNER THAT COMPLIES WITH THE MPEG ACTUAL OR DE FACTO VIDEO AND/OR AUDIO STANDARDS IS EXPRESSLY PROHIBITED WITHOUT ALL NECESSARY LICENSES UNDER APPLICABLE PATENTS. SUCH LICENSES MAY BE ACQUIRED FROM VARIOUS THIRD PARTIES INCLUDING, BUT NOT LIMITED TO, IN THE MPEG PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, L.L.C., 6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111.

Contents

Chapter 1	Overview	19
1.1	Operational Model.....	19
1.2	Platform Management	19
1.2.1	Platform Lifecycle.....	19
1.2.2	Identity	20
1.2.3	Device Authenticity.....	20
1.2.4	Ownership	20
1.2.5	Encrypted State	21
1.3	Guest Lifecycle	21
1.3.1	Launch.....	21
1.3.2	Activate and Deactivate	22
1.3.3	Snapshot and Migrate.....	23
1.3.4	Decommission.....	23
1.3.5	Debugging	24
Chapter 2	Key Management.....	25
2.1	Keys.....	25
2.1.1	Platform Diffie-Hellman Key.....	25
2.1.2	Platform Endorsement Key	26
2.1.3	Chip Endorsement Key	26
2.1.4	AMD Signing Key	26
2.1.5	Owner Certificate Authority Signing Key	26
2.1.6	AMD Root Key	27
2.1.7	Transport Integrity Key	27
2.1.8	Transport Encryption Key	27
2.1.9	Key Encryption Key	27
2.1.10	Key Integrity Key	28
2.1.11	VM Encryption Key	28
2.2	Usage.....	28
2.2.1	Key Derivation Function.....	28

2.2.2	Master Secret Derivation	28
2.2.3	Transport Key Establishment.....	29
2.2.4	Data Protection.....	29
Chapter 3	Guest Policy	31
Chapter 4	Mailbox Register Protocol.....	32
4.1	Mailbox Mode.....	32
4.1.1	Mailboxes.....	32
4.2	Ring Buffer Mode.....	33
4.3	Command Buffer	37
4.4	Command Identifiers	37
4.5	Status Codes.....	39
4.6	Endianness	40
4.7	Synchrony	40
4.8	Address Validation	40
Chapter 5	Platform Management API.....	42
5.1	Overview.....	42
5.1.1	Platform Context.....	42
5.1.2	Platform State Machine.....	43
5.1.3	Authenticity.....	43
5.1.4	Ownership.....	43
5.1.5	Non-volatile Storage	44
5.1.6	Power State Transitions	44
5.1.7	SEV-ES Trusted Memory Region	44
5.2	INIT	45
5.2.1	Actions	45
5.2.2	Parameters.....	45
5.2.3	Status Codes.....	46
5.3	INIT_EX.....	46
5.3.1	Actions	46
5.3.2	Parameters.....	47
5.3.3	Status Codes.....	48

5.4	SHUTDOWN	48
5.4.1	Actions	48
5.4.2	Parameters	49
5.4.3	Status Codes	49
5.5	PLATFORM_RESET	49
5.5.1	Actions	49
5.5.2	Parameters	49
5.5.3	Status Codes	49
5.6	PLATFORM_STATUS	50
5.6.1	Actions	50
5.6.2	Parameters	50
5.6.3	Status Codes	51
5.7	PEK_GEN	51
5.7.1	Actions	51
5.7.2	Parameters	52
5.7.3	Status Codes	52
5.8	PEK_CSR.....	52
5.8.1	Actions	53
5.8.2	Parameters	53
5.8.3	Status Codes	53
5.9	PEK_CERT_IMPORT	54
5.9.1	Actions	54
5.9.2	Parameters	54
5.9.3	Status Codes	55
5.10	PDH_GEN.....	55
5.10.1	Actions	55
5.10.2	Parameters	56
5.10.3	Status Codes	56
5.11	PDH_CERT_EXPORT	56
5.11.1	Actions	56
5.11.2	Parameters	57

5.11.3	Status Codes.....	57
5.12	DOWNLOAD_FIRMWARE	58
5.12.1	Actions	58
5.12.2	Parameters.....	58
5.12.3	Status Codes.....	59
5.13	GET_ID	59
5.13.1	Actions	59
5.13.2	Parameters.....	59
5.13.3	Status Codes.....	60
5.14	NOP	60
5.14.1	Actions	60
5.14.2	Parameters.....	60
5.14.3	Status Codes.....	60
5.15	RING_BUFFER.....	61
5.15.1	Actions	61
5.15.2	Parameters.....	61
5.15.3	Status Codes.....	62
Chapter 6	Guest Management API	63
6.1	Overview.....	63
6.1.1	Guest Context.....	63
6.1.2	Activation and Deactivation	64
6.2	LAUNCH_START	64
6.2.1	Actions	65
6.2.2	Parameters.....	66
6.2.3	Status Codes.....	66
6.3	LAUNCH_UPDATE_DATA.....	67
6.3.1	Actions	67
6.3.2	Parameters.....	67
6.3.3	Status Codes.....	68
6.4	LAUNCH_UPDATE_VMSA	68
6.4.1	Actions	68

6.4.2	Parameters	69
6.4.3	Status Codes	69
6.5	LAUNCH_MEASURE	70
6.5.1	Actions	70
6.5.2	Parameters	70
6.5.3	Status Codes	71
6.6	LAUNCH_SECRET	71
6.6.1	Actions	71
6.6.2	Parameters	72
6.6.3	Status Codes	73
6.7	LAUNCH_FINISH	73
6.7.1	Actions	73
6.7.2	Parameters	74
6.7.3	Status Codes	74
6.8	ATTESTATION	74
6.8.1	Actions	74
6.8.2	Parameters	75
6.8.3	Status Codes	75
6.9	SEND_START	76
6.9.1	Actions	76
6.9.2	Parameters	77
6.9.3	Status Codes	79
6.10	SEND_UPDATE_DATA	79
6.10.1	Actions	79
6.10.2	Parameters	80
6.10.3	Status Codes	81
6.11	SEND_UPDATE_VMSA	82
6.11.1	Actions	82
6.11.2	Parameters	83
6.11.3	Status Codes	84
6.12	SEND_FINISH	84

6.12.1	Actions	84
6.12.2	Parameters	85
6.12.3	Status Codes	85
6.13	SEND_CANCEL	85
6.13.1	Actions	85
6.13.2	Parameters	86
6.13.3	Status Codes	86
6.14	RECEIVE_START	86
6.14.1	Actions	86
6.14.2	Parameters	87
6.14.3	Status Codes	88
6.15	RECEIVE_UPDATE_DATA	89
6.15.1	Actions	89
6.15.2	Parameters	90
6.15.3	Status Codes	90
6.16	RECEIVE_UPDATE_VMSA	91
6.16.1	Actions	91
6.16.2	Parameters	92
6.16.3	Status Codes	92
6.17	RECEIVE_FINISH	93
6.17.1	Actions	93
6.17.2	Parameters	94
6.17.3	Status Codes	94
6.18	GUEST_STATUS	94
6.18.1	Actions	94
6.18.2	Parameters	95
6.18.3	Status Codes	95
6.19	ACTIVATE	95
6.19.1	Actions	95
6.19.2	Parameters	96
6.19.3	Status Codes	96

6.20	ACTIVATE_EX.....	97
6.20.1	Actions	97
6.20.2	Parameters	98
6.20.3	Status Codes	98
6.21	DEACTIVATE.....	99
6.21.1	Actions	99
6.21.2	Parameters	99
6.21.3	Status Codes	99
6.22	DF_FLUSH.....	100
6.22.1	Actions	100
6.22.2	Parameters	100
6.22.3	Status Codes	100
6.23	DECOMMISSION.....	101
6.23.1	Actions	101
6.23.2	Parameters	101
6.23.3	Status Codes	101
6.24	COPY.....	102
6.24.1	Actions	102
6.24.2	Parameters	102
6.24.3	Status Codes	102
6.25	SWAP_OUT.....	103
6.25.1	Actions	103
6.25.2	Parameters	104
6.25.3	Status Codes	105
6.26	SWAP_IN.....	105
6.26.1	Actions	106
6.26.2	Parameters	107
6.26.3	Status Codes	107
Chapter 7	Debugging API.....	109
7.1	DBG_DECRYPT	109
7.1.1	Actions	109

7.1.2	Parameters.....	109
7.1.3	Status Codes.....	110
7.2	DBG_ENCRYPT.....	110
7.2.1	Actions.....	110
7.2.2	Parameters.....	111
7.2.3	Status Codes.....	111
Appendix A Usage Flows		112
Appendix B AMD Certificate Authority Certificates		116
B.1	Certificate Format.....	116
B.2	Certificate Signature.....	117
B.3	Certificate Validation.....	117
Appendix C SEV Certificates		118
C.1	Certificate Format.....	118
C.2	Elliptic Curve Enumeration.....	119
C.3	Public Key Formats.....	120
C.3.1	RSA Public Key.....	120
C.3.2	ECDSA Public Key.....	120
C.3.3	ECDH Public Key.....	120
C.4	Signature Formats.....	121
C.4.1	RSA Signature.....	121
C.4.2	ECDSA Signature.....	121
C.5	Certificate Validation.....	121

List of Tables

Table 1. Summary of Keys.....	25
Table 2. Guest Policy Structure.....	31
Table 3. CmdResp Register Layout	32
Table 4. CmdBufAddr_Hi Register Layout	32
Table 5. CmdBufAddr_Lo Register Layout.....	32
Table 6. CmdPtr Entry Layout	34
Table 7. Command Flags Values	34
Table 8. StatVal Entry Layout.....	34
Table 9. RBCtl Register Layout.....	35
Table 10. RBTail Register Layout	35
Table 11. RBHead Register Layout	35
Table 12. Ring Buffer Mode Error Recovery Handshake Sequence	37
Table 13. Command Identifiers.....	37
Table 14. Status Codes	39
Table 15. Platform Context (PCTX) Fields	42
Table 16. PSTATE Finite State Machine.....	43
Table 17. Power Management Transitions.....	44
Table 18. INIT Command Buffer.....	45
Table 19. INIT Command Status Codes	46
Table 20. INIT_EX Command Buffer	47
Table 21. INIT_EX Command Status Codes.....	48
Table 22. SHUTDOWN Command Status Codes	49
Table 23. PLATFORM_RESET Command Status Codes.....	50
Table 24. PLATFORM_STATUS Command Buffer	50
Table 25. PLATFORM_STATUS Command Status Codes.....	51
Table 26. PEK_GEN Command Status Codes.....	52
Table 27. PEK_CSR Command Buffer.....	53
Table 28. PEK_CSR Command Status Codes	53
Table 29. PEK_CERT_IMPORT Command Buffer.....	54

Table 30. PEK_CERT_IMPORT Command Status Codes	55
Table 31. PDH_GEN Command Status Codes.....	56
Table 32. PDH_CERT_EXPORT Command Buffer.....	57
Table 33. PDH_CERT_EXPORT Certificates Buffer.....	57
Table 34. PDH_CERT_EXPORT Command Status Codes	57
Table 35. DOWNLOAD_FIRMWARE Command Buffer	58
Table 36. DOWNLOAD_FIRMWARE Command Status Codes.....	59
Table 37. GET_ID Command Buffer	59
Table 38. GET_ID Command Status Codes	60
Table 39: NOP Status Codes.....	60
Table 40. RING_BUFFER Command Buffer	61
Table 41. RING_BUFFER Status Codes.....	62
Table 42. Guest Context (GCTX) Fields	63
Table 43. GSTATE Finite State Machine.....	63
Table 44. LAUNCH_START Command Buffer	66
Table 45. LAUNCH_START Session Data Buffer.....	66
Table 46. LAUNCH_START Status Codes	66
Table 47. LAUNCH_UPDATE_DATA Command Buffer.....	67
Table 48. LAUNCH_UPDATE_DATA Status Codes	68
Table 49. LAUNCH_UPDATE_VMSA Command Buffer	69
Table 50. LAUNCH_UPDATE_VMSA Status Codes.....	69
Table 51. LAUNCH_MEASURE Command Buffer.....	70
Table 52. LAUNCH_MEASURE Measurement Buffer	71
Table 53. LAUNCH_MEASURE Status Codes.....	71
Table 54. LAUNCH_SECRET Command Buffer.....	72
Table 55. LAUNCH_SECRET Packet Header Buffer	72
Table 56. LAUNCH_SECRET Status Codes	73
Table 57. LAUNCH_FINISH Command Buffer.....	74
Table 58. LAUNCH_FINISH Status Codes	74
Table 59. ATTESTATION Command Buffer	75
Table 60. Attestation Report Buffer.....	75

Table 61. ATTESATION Status Codes	76
Table 62. SEND_START Command Buffer.....	77
Table 63. SEND_START Platform Certificates Buffer	78
Table 64. SEND_START AMD Certificates Buffer	78
Table 65. SEND_START Session Data Buffer	79
Table 66. SEND_START Status Codes	79
Table 67. SEND_UPDATE_DATA Command Buffer	80
Table 68. SEND_UPDATE_DATA Packet Header Buffer	81
Table 69. SEND_UPDATE_DATA Status Codes.....	81
Table 70. SEND_UPDATE_VMSA Command Buffer	83
Table 71. SEND_UPDATE_VMSA Packet Header Buffer	83
Table 72. SEND_UPDATE_VMSA Status Codes	84
Table 73. SEND_FINISH Command Buffer	85
Table 74. SEND_FINISH Status Codes.....	85
Table 75. SEND_CANCEL Command Buffer	86
Table 76. SEND_CANCEL Status Codes.....	86
Table 77. RECEIVE_START Command Buffer	87
Table 78. RECEIVE_START Session Data Buffer	88
Table 79. RECEIVE_START Status Codes	88
Table 80. RECEIVE_UPDATE_DATA Command Buffer	90
Table 81. RECEIVE_UPDATE_DATA Packet Header Buffer	90
Table 82. RECEIVE_UPDATE_DATA Status Codes	91
Table 83. RECEIVE_UPDATE_VMSA Command Buffer	92
Table 84. RECEIVE_UPDATE_VMSA Packet Header Buffer	92
Table 85. RECEIVE_UPDATE_VMSA Status Codes.....	93
Table 86. RECEIVE_FINISH Command Buffer	94
Table 87. RECEIVE_FINISH Status Codes	94
Table 88. GUEST_STATUS Command Buffer.....	95
Table 89. GUEST_STATUS Status Codes	95
Table 90. ACTIVATE Command Buffer.....	96
Table 91. ACTIVATE Status Codes	96

Table 92. ACTIVATE_EX Command Buffer	98
Table 93. ACTIVATE_EX Status Codes	98
Table 94. DEACTIVATE Command Buffer	99
Table 95. DEACTIVATE Status Codes	99
Table 96. DF_FLUSH Command Status Codes	100
Table 97. DECOMMISSION Command Buffer	101
Table 98. DECOMMISSION Status Codes.....	101
Table 99. COPY Command Buffer.....	102
Table 100. COPY Status Codes	103
Table 101. SWAP_OUT Command Buffer	104
Table 102. SWAP_OUT & SWAP_IN Metadata Entry	105
Table 103. SWAP_OUT Status Codes	105
Table 104. SWAP_IN Command Buffer	107
Table 105. SWAP_IN Status Codes	107
Table 106. DBG_DECRYPT Command Buffer.....	109
Table 107. DBG_DECRYPT Status Codes	110
Table 108. DBG_ENCRYPT Command Buffer.....	111
Table 109. DBG_ENCRYPT Status Codes	111
Table 110. AMD Signing Key Certificate Format.....	116
Table 111. Key Usage Encoding	116
Table 112. SEV Certificate Format	118
Table 113. USAGE Enumeration (All other encodings are reserved).....	118
Table 114. ALGO Enumeration (All other encodings are reserved)	119
Table 115. CURVE Enumeration (All other encodings are reserved).....	119
Table 116. RSA Public Key.....	120
Table 117. ECDSA Public Key	120
Table 118. ECDH Public Key.....	120
Table 119. RSA Signature	121
Table 120. ECDSA Signature	121

Revision History

Date	Revision	Description
April 2020	3.24	<ul style="list-style-type: none"> Change API version to 0.24 Add RING_BUFFER command
April 2020	3.23	<ul style="list-style-type: none"> Change API version to 0.23 Add SWAP_OUT/SWAP_IN commands Add SEND_CANCEL command Add NOP command Add ATTESTATION command DF_FLUSH is valid in any state
July 2019	3.22	<ul style="list-style-type: none"> Change API version to 0.22 Describe differences between versions of the API Add ACTIVATE_EX command Add INIT_EX command Add Sent Guest state
October 2018	3.07	<ul style="list-style-type: none"> Change API version to 0.17 Add COPY command Improve LAUNCH_MEASURE, LAUNCH_SECRET
February 2018	3.06	<ul style="list-style-type: none"> Change API version to 0.16 Added DOWNLOAD_FIRMWARE command. Added GET_ID command. API version doesn't matter in certificate validation. MNONCE generation and usage in LAUNCH_MEASURE and LAUNCH_SECRET. WBINVD required after SEV-ES SHUTDOWN command. Document mailbox register numbers.
October 2017	3.04	<ul style="list-style-type: none"> RECEIVE_START works in PSTATE.INIT as well as PSTATE.WORKING. Updated flowchart to emphasize dependency on PDH_CERT_EXPORT before LAUNCH_START. Clarification on snapshot constraint. Other minor edits

Date	Revision	Description
August 2017	3.03	<ul style="list-style-type: none"> • Move Policy to command buffer in all commands • SEND_UPDATE_DATA TRANS_LENGTH parameter is In and Out. • Document MNonce generation • Insert documentation for RECEIVE_UPDATE_DATA • Removed “Key Management” from title • Many other minor edits
January 2017	3.02	<ul style="list-style-type: none"> • Split all the *_UPDATE commands into *_UPDATE_DATA and *_UPDATE_VMSA • Added LAUNCH_SECRET command to allow for secret data to be injected into a launched guest. Updated state machine to reflect new command. • Replaced X.509 certificate usage with API-specific formatted certificates. • Added clarity around cryptographic algorithms; Switched from ECC curve P-256 to ECC P-384. • Addressed industry partner inputs to command buffer structures. • General clarifications.
August 2016	3.01	Bug fixes and minor edits.
May 2016	3.00	Initial Release

References

FIPS 186-4	Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-4. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS 198-1	The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication 198-1. http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
SP 800-38F	Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping. NIST Special Publication 800-38F http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP 800-56A	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A Revision 2. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf
SP 800-108	Recommendation for Key Derivation Using Pseudorandom Functions. NIST Special Publication 800-108. http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf
RFC 3279	Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF RFC 3279. https://www.ietf.org/rfc/rfc3279.txt
RFC 5480	Elliptic Curve Cryptography Subject Public Key Information. IETF RFC 5480. https://www.ietf.org/rfc/rfc5480.txt
RFC 5758	Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA. IETF RFC 5758. https://www.ietf.org/rfc/rfc5758.txt

Chapter 1 Overview

The Secure Encrypted Virtualization (SEV) feature allows the memory contents of a virtual machine (VM) to be transparently encrypted with a key unique to the guest virtual machine (VM). The memory controller contains a high-performance encryption engine which can be programmed with multiple keys for use by different VMs in the system. The programming and management of these keys and secure data transfer between host hypervisor and guest VM memory is handled by the SEV firmware running AMD Secure Processor. The API available to the host hypervisor for these operations is the focus of this document.

1.1 Operational Model

The capabilities of the SEV feature can support many operational environments. It is capable of supporting both lightweight virtualized containers as well as conventional virtual machines within an enterprise cloud environment. In either case, there are two parties concerned in the deployment of SEV guests: the guest owner and the platform owner. For example, in a cloud environment, the platform owner would be the cloud vendor and the guest owner would be the user that wishes to run their workload in the cloud.

Guest owners wish to protect the confidentiality of the data running within their guests. While they trust the platform owner to provide the infrastructure to run their guest, they benefit from reducing their risk exposure to vulnerabilities within that infrastructure. The SEV feature encrypts the contents of the memory of the guest and provides assurances that it was encrypted properly. The encryption of memory places an additional burden on attackers within the operational environment who may have already obtained some illicit access to the guest's memory through a vulnerability in the hypervisor or other supporting enterprise software.

Platform owners wish to provide such protection to guest owners, but also must manage the computing resources efficiently and effectively. Resource allocation needs to be fluid and their customers data must be safely backed up in case of failure. The SEV feature has been developed to be sensitive to this requirement; it integrates into the conventional guest lifecycle surrounding launching, migration, and snapshotting of guests.

1.2 Platform Management

This section describes the management of the platform by the platform owner.

1.2.1 Platform Lifecycle

The firmware maintains context for the platform. To initialize the platform, the INIT command is executed to initialize and configure the platform context. The INIT command accepts configuration parameters that affect the behavior of the platform until the system is reset or the

SHUTDOWN command is invoked. The SHUTDOWN command clears all platform and guest state that exists in non-persistent storage.

The status of the platform can be queried using the PLATFORM_STATUS command. This command provides information about the state of the platform.

If the platform needs to be reset to factory settings, the PLATFORM_RESET command can be issued by the hypervisor. This command clears all persistently stored platform state including the platform certificates and signing keys (see below).

1.2.2 Identity

The firmware identifies itself by an asymmetric signing key generated during the one-time configuration steps taken before the platform may be put into production. This key is called the Platform Endorsement Key (PEK) and is used to sign the Platform Diffie-Hellman key (PDH). By signing this key, the firmware authenticates the cryptographic contexts built with the PDH.

The firmware provides interfaces to manage the PDH: PDH_GEN and PDH_CERT_EXPORT. The PDH_GEN command generates a new PDH, which is then signed by the PEK. This command is not necessary for normal operation; it is provided to support the ability to generate a fresh PDH in case the platform owner's security policies require it.

The PDH_CERT_EXPORT command exports the PDH and its PEK signature. The PEK is signed by two keys to connect it to two different roots of trust. These roots of trust provide authenticity and ownership proof to third parties, described in the following sections. These signatures are also exported via the PDH_CERT_EXPORT command.

1.2.3 Device Authenticity

The firmware provides a mechanism to verify that it is executing on AMD hardware that is capable of supporting SEV. Each platform contains a chip-unique signing key called the Chip Endorsement Key (CEK). The CEK is signed by the AMD SEV Signing Key (ASK), which is signed by the AMD root signing key (ARK). A guest owner or remote platform interacting with the firmware can obtain the public component of the AMD signing keys to prove the hardware authenticity. The mechanism to obtain the public key is outside the scope of this document.

The CEK signs the PEK. This signature connects the PEK to the ARK, which is a root of trust.

1.2.4 Ownership

The firmware also provides interfaces to verify the identity of the owner of the platform on which it is executing: PEK_CSR and PEK_CERT_IMPORT.

PEK_CSR generates a certificate signing request (CSR) for the PEK. This signing request contains the PEK public key and other identifying information about the platform. When the platform owner's certificate authority is provided this CSR, it generates a certificate using the

public key of the PEK and signs it with its root signing key. This signed certificate connects the PEK to the platform owner's root of trust. It can be used by external parties to verify ownership of the platform.

The platform itself keeps a copy of the PEK certificate in persistent storage. In order to install the certificate in the platform, the PEK_CERT_IMPORT command is issued. Once the certificate is imported, the platform owner has taken ownership of the platform.

1.2.5 Encrypted State

When a #VMEXIT occurs, the register state of the guest is saved to memory in the VMCB save area. Access to this save area exposes the guest register state to manipulation and observation by attackers. To mitigate this, some SEV capable platforms support the AMD SEV Encrypted State (SEV-ES) feature. When enabled, SEV-ES encrypts and integrity protects the VMCB save area with the memory encryption key.

1.3 Guest Lifecycle

The SEV feature supports the common guest lifecycle events so that SEV can integrate well within already existing cloud infrastructures. These events include launching, running, snapshotting, migrating, and decommissioning guests.

The status of a guest is available via the GUEST_STATUS command. This command reports the current SEV state the guest is in and other information.

1.3.1 Launch

When a guest is launched, its memory must first be encrypted before SEV can be enabled in hardware for this guest. The firmware provides interfaces to bootstrap the memory encryption for this purpose: the LAUNCH commands. These six commands together generate a fresh memory encryption key for the guest, encrypt guest memory, and provide an attestation of the successful launch.

LAUNCH_START is called first to create a guest context within the firmware. The hypervisor provides the firmware with the guest's security policy and the public key of the guest owner. The guest policy constrains the use and features activated for the lifetime of the launched guest, such as disallowing debugging, enabling key sharing, or turning on other SEV-related features. The provided public key allows the firmware to establish a cryptographic session with the guest owner to negotiate keys used for attestation (see Chapter 2).

The hypervisor loads the guest into memory and begins calling subsequent LAUNCH_UPDATE_DATA commands.

LAUNCH_UPDATE_DATA encrypts the memory provided to it by the hypervisor. It also calculates a measurement of the memory. This measurement is a signature of the memory contents

that can be sent to the guest owner as an attestation that the memory was encrypted correctly by the firmware.

The hypervisor also initializes the VM save area within the VMCB by issuing `LAUNCH_UPDATE_VMSA`. This command encrypts the VM save area. The `LAUNCH_UPDATE_VMSA` command is only used when the SEV-ES feature is enabled.

Once the guest data and VM save areas are encrypted, the hypervisor issues the `LAUNCH_MEASURE` command to produce a measurement of the data encrypted by the launch flow. The guest owner may wait to provide the guest with confidential information until it can verify the attestation measurement. Since the guest owner knows the initial contents of the guest at boot, the attestation measurement can be verified by comparing it to what the guest owner expects.

To provide the guest with secret data after the measurement is validated, the guest owner wraps a secret and sends it to the guest through the `LAUNCH_SECRET` command.

After completion of the launch workflow, the hypervisor issues `LAUNCH_FINISH` to ready the guest for execution.

1.3.2 Activate and Deactivate

AMD-V requires the guest to be associated with an ASID within its VMCB prior to executing a `VMRUN` instruction to run the guest. Additionally, `LAUNCH_UPDATE_DATA` and `LAUNCH_UPDATE_VMSA` require an active ASID. An ASID is an identifier associated with all memory accesses of the guest. SEV uses ASIDs to associate memory encryption keys with the guest. The number of guests that the firmware supports is conceptually unlimited (though resource constraints impose a more practical limit.) However, there are only a limited number of ASIDs that can be simultaneously associated with keys. The firmware provides to the hypervisor interfaces to overcommit ASIDs to guests using the `ACTIVATE`, `DEACTIVATE`, and `DF_FLUSH` commands, and the `WBINVD` instruction.

The `ACTIVATE` command allows the hypervisor to inform the firmware that a given guest is associated with an ASID. When this command is issued, the firmware programs the memory controller with the guest's memory encryption key. This key is then associated with the guest's ASID.

The hypervisor uses the `DEACTIVATE` command to de-allocate an ASID from a guest. This removes the association in the memory controller between the guest and the ASID. A hypervisor may wish to re-allocate this ASID due to operating system specific ASID scheduling protocols. It may also need to re-allocate ASIDs if there are no SEV-capable ASIDs currently available.

Before re-activating an ASID, the firmware requires the hypervisor to flush the cache back to memory with a `WBINVD` instruction on each core. Once `WBINVD` completes, the hypervisor must call `DF_FLUSH` to ensure that the data fabric is completely flushed. The `DF_FLUSH` command checks that a `WBINVD` has been executed since the last `DEACTIVATE`. The firmware enforces the `WBINVD` and `DF_FLUSH` flow to ensure any unencrypted data in the cache and data

fabric write buffers associated with the previous key value is written back to memory before the key can be changed. On success, the DF_FLUSH command marks the previously deactivated ASIDs as usable. The DF_FLUSH command provides the hypervisor flexibility to batch multiple DEACTIVATE calls together and flushing the cache and data fabric only once afterwards.

1.3.3 Snapshot and Migrate

Snapshotting and migration both require the memory image of a guest to be removed from the system to be reconstituted at another platform later in time. (Note that in this version of the API, the platform reconstituting the image must be known at the time the snapshot is taken.) To protect the confidentiality of an SEV protected guest while in transit and storage, its memory image must be encrypted with a key that can be recovered by the next platform to execute it. The firmware provides the same set of interfaces to support this protection: SEND_START, SEND_UPDATE_DATA, SEND_UPDATE_VMSA, and SEND_FINISH for the sending or snapshotting platform and RECEIVE_START, RECEIVE_UPDATE_DATA, RECEIVE_UPDATE_VMSA, and RECEIVE_FINISH for the receiving or reconstituting platform.

The SEND_START, SEND_UPDATE_DATA, SEND_UPDATE_VMSA, and SEND_FINISH commands are used to transform the guest's memory image for transmission. SEND_START establishes a cryptographic context with the remote platform. Subsequent invocations of SEND_UPDATE_DATA may be used to encrypt additional memory regions. Each UPDATE operation is independently integrity protected.

The SEND_START command enforces the guest's policy restriction on which platforms it may be transferred to. The policy can require the destination to be an SEV-capable platform or require the receiving platform to have the same owner as the sending one—or both. This check is performed by verifying the receiving platform's platform certificates and certificate chains.

The receiving platform uses the related RECEIVE_START, RECEIVE_UPDATE_DATA, RECEIVE_UPDATE_VMSA, and RECEIVE_FINISH in the same manner as the sending platform. The RECEIVE_START derives the cryptographic context necessary to re-encrypt the guest. The RECEIVE_UPDATE_DATA and RECEIVE_UPDATE_VMSA commands re-encrypts memory regions provided by the hypervisor. The RECEIVE_FINISH command finalizes the operation. However, the RECEIVE_START command does not verify the identity of the sending platform.

The RECEIVE_START command is the only command other than the LAUNCH_START command that generates a new guest context and guest handle.

1.3.4 Decommission

When the hypervisor needs to shut down a guest, it issues the DECOMMISSION command to the firmware. The firmware then deletes the memory encryption key and all other internal state the firmware has kept for the guest.

1.3.5 Debugging

The firmware provides two interfaces to assist in the debugging of an SEV enabled guest: `DBG_ENCRYPT` and `DBG_DECRYPT`. These commands request the firmware to encrypt or decrypt the data at the given memory regions provided by the hypervisor. Since decrypting protected memory allows the hypervisor to gain access to guest memory, the guest policy must explicitly allow debugging to enable these two commands.

Chapter 2 Key Management

The primary purpose of the firmware is to manage the memory encryption keys and the keys to support the guest lifecycle within an enterprise environment. This section describes each of these keys. Table 1 summarizes all the keys, their abbreviations, their algorithm, and their usage.

Table 1. Summary of Keys

Key	Abbr.	Algorithm	Usage
Platform Diffie-Hellman Key	PDH	ECDH curve P-384	Key agreement
Platform Endorsement Key	PEK	ECDSA curve P-384	Signing the PDH
Chip Endorsement Key	CEK	ECDSA curve P-384	Signing the PEK
AMD SEV Signing Key	ASK	RSA 2048	Signing the CEK
AMD Root Key	ARK	RSA 2048	Signing the ASK; AMD root of trust
Owner Certificate Authority	OCA	ECDSA curve P-384	Signing the PEK; platform owner root of trust
Transport Integrity Key	TIK	HMAC SHA-256	Trusted channel Integrity
Transport Encryption Key	TEK	AES 128	Trusted channel confidentiality
Key Encryption Key	KEK	AES 128	Key wrapping
Key Integrity Key	KIK	HMAC SHA-256	Key wrapping
VM Encryption Key	VEK	AES 128	Guest memory encryption

2.1 Keys

This section describes the management of the platform by the platform owner.

2.1.1 Platform Diffie-Hellman Key

The Platform Diffie-Hellman key (PDH) is an Elliptic Curve Diffie Hellman (ECDH) key using curve P-384 as defined in section D.1.2.3 of [FIPS 186-4]. The PDH is used to negotiate a master secret between the SEV firmware and external entities. This master secret is then used with a key derivation function (KDF) to establish a trusted channel.

The SEV firmware generates the PDH using output from a secure entropy source. The lifetime of this key is the lifetime of the platform's membership in its cloud or domain. When the Platform Encryption Key (PEK) changes, the PDH is regenerated.

2.1.2 Platform Endorsement Key

The Platform Endorsement Key (PEK) is an ECDSA signing key using curve P-384 as defined in section D.1.2.3 of [FIPS 186-4]. It signs the PDH to anchor the PDH to the AMD root of trust and the platform owner's root of trust.

The SEV firmware generates the PEK using a secure entropy source. The lifetime of this key is the lifetime of the platform's membership in its cloud or domain.

2.1.3 Chip Endorsement Key

The Chip Endorsement Key (CEK) is an ECDSA signing key using curve P-384 as defined in section D.1.2.3 of [FIPS 186-4]. It signs the PEK to anchor the PEK to the AMD root of trust. Each chip has a unique CEK which is derived from secrets stored in chip-unique OTP fuses. The lifetime of this key is the lifetime of the individual chip.

2.1.4 AMD Signing Key

The AMD SEV Signing Key (ASK) is an RSA 2048-bit key pair. It is an intermediate signing key, signed by the ARK. The ASK private key signs the CEK public key to demonstrate that the CEK is an authentic AMD key.

The AMD Key Distribution Server (KDS) managed by AMD contains the ASK private key. The lifetime of this key is the lifetime of the product line.

2.1.5 Owner Certificate Authority Signing Key

The platform owner's Certificate Authority key (OCA) is an ECDSA signing key using curve P-384 as defined in section D.1.2.3 of [FIPS 186-4]. It is the root of trust of the platform owner and its signatures signify platform ownership. The OCA private key signs the PEK public key to demonstrate that the PEK is owned by the platform owner.

When the platform is self-owned, the firmware stores both the OCA private key and public key. In this case, no entity outside of the SEV firmware has access to the OCA private key; the SEV firmware generates the OCA key pair using a secure entropy source.

When the platform is owned by an external entity, the firmware stores only the OCA public key. In this case, only the platform owner has access to the private key; the platform owner must generate the OCA key pair according to its own security policies.

When the SEV firmware stores anything, including the OCA certificate, in non-volatile storage, it is encrypted and integrity protected.

2.1.6 AMD Root Key

The AMD Root key (ARK) is an RSA 2048 key pair. It is the root of trust of AMD and its signatures signify AMD authenticity. The ARK private key signs the ASK public key to demonstrate that the ASK is an authentic AMD key.

The AMD KDS that is managed by AMD contains the ARK private key. The lifetime of this key is the lifetime of the product line.

2.1.7 Transport Integrity Key

The Transport Integrity Key (TIK) is an HMAC SHA-256 integrity key. The SEV firmware uses the TIK to integrity protect all trusted information transferred between the firmware and external entities such as the guest owner or another SEV firmware instance.

The SEV firmware generates the TIK during the sending operational flows. In this case, the firmware generates the TIK from a secure entropy source.

The SEV firmware imports a wrapped TIK during the launching and receiving operational flows.

The lifetime of this key is the lifetime of a single send, receive, or launch session. Once SEND_FINISH, RECEIVE_FINISH, or LAUNCH_FINISH are invoked respectively, the TIK is destroyed.

2.1.8 Transport Encryption Key

The Transport Encryption Key (TEK) is an AES-128 encryption key. The SEV firmware uses the TEK to encrypt all confidential information transferred between the firmware and external entities such as the guest owner or another SEV firmware instance.

The SEV firmware generates the TEK during the sending operational flows. In this case, the firmware generates the TEK from a secure entropy source.

The SEV firmware imports a wrapped TEK during the launching and receiving operational flows.

The lifetime of this key is the lifetime of a single send, receive, or launch session. Once SEND_FINISH, RECEIVE_FINISH, or LAUNCH_FINISH are invoked respectively, the TEK is destroyed.

2.1.9 Key Encryption Key

The Key Encryption Key (KEK) is an AES-128 encryption key. It is used to wrap the TEK and TIK during session establishment.

The SEV firmware derives the KEK from the master secret negotiated during the key agreement protocol. The lifetime of this key is the lifetime of a single send, receive, or launch session. Once

SEND_FINISH, RECEIVE_FINISH, or LAUNCH_FINISH are invoked respectively, the KEK is destroyed.

2.1.10 Key Integrity Key

The Key Integrity Key (KIK) is an HMAC-SHA-256 key. It is used to wrap the TEK and TIK during session establishment.

The SEV firmware derives the KIK from the master secret negotiated during the key agreement protocol. The lifetime of this key is the lifetime of a single send, receive, or launch session. Once SEND_FINISH, RECEIVE_FINISH, or LAUNCH_FINISH are invoked respectively, the KIK is destroyed.

2.1.11 VM Encryption Key

The VM Encryption Key (VEK) is an AES-128 encryption key. The UMC uses the memory encryption key to encrypt guest data while the guest is operating.

The SEV firmware generates the memory encryption key from a secure entropy source. The lifetime of this key is the lifetime of the guest while it executes on this platform. The guest will use a different VEK when migrated to a remote platform. The remote platform is responsible for generation of the new VEK.

2.2 Usage

The SEV firmware must be capable of communicating securely with other parties. This section describes how the firmware establishes a trusted channel between itself and the remote party.

Each trusted channel has a client and server. In the SEV API, the guest owner and the SEV firmware instance issuing send commands are designated the client parties. The SEV firmware issuing launch and receive commands are designated the servers. These designations do not imply any further semantics beyond this trusted channel protocol.

2.2.1 Key Derivation Function

The SEV firmware uses a key derivation function (KDF) to securely derive new secrets from old secrets. Specifically, it uses the KDF to derive the CEK from fused secrets, and to derive transport keys from a negotiated master secret. The KDF used is the counter mode KDF specified in NIST [SP 800-108], Section 5.1. It uses the HMAC-SHA-256 as the underlying pseudorandom function (PRF). The parameter r is set to 32 and the length in bits of $[L]-2$ is 32.

2.2.2 Master Secret Derivation

Key agreement is accomplished using the Elliptic Curve Diffie-Hellman (ECDH) protocol. The protocol follows the Static Unified Model described in Section 6.3.2 of NIST [SP 800-56A].

In summary, the following occurs:

- The client retrieves the server's ECDH public key, $Q_{\rightarrow S}$
- The client transmits its ECDH public key, Q_C , to the server
- The client transmits a fresh nonce, N , to the server
- The client calculates a shared secret, Z , from $Q_{\rightarrow S}$ and its private key, d_C .
- The server calculates a shared secret, Z , from $Q_{\rightarrow C}$ and its private key, d_S .
- The client and server calculate the master secret, M , from Z and N using the KDF.

The derivation of M uses the KDF. The parameters to the KDF are:

- Secret input: Z
- Label: "sev-master-secret", ASCII encoded (17 bytes).
- Context: N

The intermediate secret Z must be securely deleted after the master secret is derived.

2.2.3 Transport Key Establishment

The client generates the transport encryption key (TEK) and the transport integrity key (TIK). The TEK and TIK are then transmitted to the server using key wrapping keys derived from the master secret, M .

The Key Encryption Key (KEK) is derived using the KDF with the following parameters:

- Secret input: M
- Label: "sev-kek", ASCII encoded (7 bytes).
- Context: None

The Key Integrity Key (KIK) is derived using the KDF with the following parameters:

- Secret input: M
- Label: "sev-kik", ASCII encoded (7 bytes).
- Context: None

The TEK and TIK are concatenated together in that order, encrypted, and integrity protected using the scheme specified in Section 2.2 on page 28, using the KEK as the encryption key, the KIK as the integrity key, and a freshly generated IV.

When transmitting guest data from client to server, the TEK and TIK are used with the data protection scheme specified in Section 2.2 on page 28, according to the command specification.

2.2.4 Data Protection

The SEV API uses a common data protection scheme for multiple uses. The data protection scheme requires the following input parameters:

- A 128-bit initialization vector, IV
- A 128-bit encryption key, K
- A 128-bit integrity protection key, I
- A message to protect, M

Two different messages should never be protected with the same IV and K. The IV may be repeated only when used with a different K.

The data protection scheme used by this API utilizes AES-128 CTR mode for confidentiality protection and HMAC-SHA-256 for integrity protection. The output of this data protection is:

- $C = \text{AES-128-CTR}(M; K, IV)$
- $\text{MAC} = \text{HMAC-SHA-256}(IV \parallel C; I)$

The client must send C, MAC, and IV to the server to allow the server to recover the plaintext.

Chapter 3 Guest Policy

The firmware maintains a guest policy provided by the guest owner. This policy is enforced by the firmware and restricts what configuration and operational commands can be performed on this guest by the hypervisor. The policy also requires a minimum firmware level.

The guest policy is provided to firmware during guest launch. The policy is then bound to the guest and cannot be changed throughout the lifetime of the guest. The policy is also transmitted during snapshot and migration flows and enforced on the destination platform.

The guest policy is a 4-byte structure with the fields shown in Table 2:

Table 2. Guest Policy Structure

Offset	Bit(s)	Name	Description
000h	0	NODBG	Debugging of the guest is disallowed when set
	1	NOKS	Sharing keys with other guests is disallowed when set
	2	ES	SEV-ES is required when set
	3	NOSEND	Sending the guest to another platform is disallowed when set
	4	DOMAIN	The guest must not be transmitted to another platform that is not in the domain when set.
	5	SEV	The guest must not be transmitted to another platform that is not SEV capable when set.
	15:6	Reserved. Should be zero.	
002h	7:0	API_MAJOR	The guest must not be transmitted to another platform with a lower firmware version.
003h	7:0	API_MINOR	

The policy bits for a given guest are referenced with the format `POLICY.<FLAG_NAME>`. For instance, the key sharing flag is referred to as `POLICY.NOKS`.

Chapter 4 Mailbox Register Protocol

Software on the x86 CPUs communicate with the AMD Secure Processor through a set of MMIO registers, referred to as mailbox registers. This section describes the protocol used by the x86 SEV driver for communicating through these mailbox registers to invoke the SEV key management functions described in this API.

There are two modes these mailboxes can be used in:

- Mailbox mode — the original mode the SEV FW supports
- Ring Buffer mode — a new mode with SEV FW 0.24

Entry to Ring Buffer mode requires a RING_BUFFER command, sent in Mailbox mode. After that, commands are submitted and status returned via queues in the ring buffers.

4.1 Mailbox Mode

In a nutshell, the driver writes the command ID and 64-bit physical address of a command buffer (if parameters are required) into the mailbox registers. The SEV firmware takes action and returns a 16-bit return code and potentially writes output to the same physical address parameter region.

4.1.1 Mailboxes

The SEV API exposes an external interface to the x86 system software through three memory mapped registers: CmdResp, CmdBufAddr_Hi, and CmdBufAddr_Lo. Their formats are described in Table 3, Table 4, and Table 5, respectively.

Table 3. CmdResp Register Layout

Bit(s)	Description/Purpose
31	0: Command; 1: Response
[30:26]	Reserved. Must be zero.
[25:16]	Command ID
[15:0]	When bit 31 is 1: Status code from SEV Firmware When bit 31 is 0: Bit 0: x86 requests SEV FW to interrupt on completion. Other bits are reserved, must be zero.

Table 4. CmdBufAddr_Hi Register Layout

Bit(s)	Description/Purpose
31:0	Most significant 32 bits of the physical address of the command buffer

Table 5. CmdBufAddr_Lo Register Layout

Bit(s)	Description/Purpose
31:0	Least significant 32 bits of the physical address of the command buffer

The x86 system software issues commands to the firmware by writing to these 32-bit registers. These registers appear in the memory-mapped I/O space of the PSP device in PCI space. The precise physical address of these registers is normally determined by BIOS when it performs PCI device enumeration. Relative to the base address assigned by BIOS to the PSP device's memory-mapped I/O space, `CmdResp` is register 32 (offset 128), `CmdBufAddr_Lo` is register 56 (offset 224), and `CmdBufAddr_Hi` is register 57 (offset 228).

The x86 system software first constructs a command buffer (formatted specifically for the given command) within system DRAM. The x86 system software then writes the most significant 32 bits of the command buffer's physical address into the `CmdBufAddr_Hi` register and the least significant 32 bits into the `CmdBufAddr_Lo` address.

The x86 software then writes to the `CmdResp` register to issue the command with bit 31 set to 0 to indicate that this is an issued command and bits [25:16] containing the command identifier. The values written into bits [15:1] are reserved and must be written as 0 by the x86 software. The x86 software may set bit 0 to inform the SEV firmware to interrupt the x86 core upon command completion.

Upon command completion, the PSP firmware writes to the `CmdResp` register. The PSP firmware sets bit 31 to 1 to indicate that this is a command response and bits [15:0] to the 16-bit status code.

If the command is not recognized, an `INVALID_COMMAND` status code is returned.

4.2 Ring Buffer Mode

Upon successful completion of the `RING_BUFFER` command from Mailbox mode, communications between the x86 SEV driver and the PSP firmware changes the semantics of the three registers.

- The `CmdResp` register becomes the `RBCtl` register. It is only ever written by the x86.
- The `CmdBufAddr_Hi` register becomes the `RBTail` register. It is only ever written by the x86.
- The `CmdBufAddr_Lo` register becomes the `RBHead` register. It should never be written by the x86; the PSP will update it.

There are up to two queues created by the `RING_BUFFER` command, each with parallel arrays. The parallel arrays for `CmdPtr` entries (written only by the x86) and `StatVal` entries (written only by the PSP FW). The two queues are a low priority queue (required) and a high priority queue (optional). Each time the PSP is ready to process a new command, the high priority queue is checked for work before the low priority queue. As long as the high priority queue has work, the low priority queue will not be serviced.

Work to be entered on a queue is written by the x86 in an entry at the tail of the queue in a 16-byte entry called a `CmdPtr`. The `CmdPtr` replicates and extends the information that Mailbox mode passes in the three registers. The `CmdPtr` entry is structured as shown in Table 6.

Table 6. CmdPtr Entry Layout

Byte Offset	Bits	Description
00h	15:0	Command Identifier (from Table 13)
02h	15:0	Command Flags
04h	31:0	Software Data
08h	63:0	Command Buffer Pointer

Software Data is an arbitrary 32-bit value that the x86 software can use for anything. The Command Identifier contains the Command ID value from Table 13. Command Flags contains the flags defined in Table 7. The Command Buffer Pointer points at the command buffer for the Command Identifier, if any.

Table 7. Command Flags Values

Bit(s)	Name	Description
[15:2]	-	Reserved. Must be 0.
1	ContinueOnError	1 – Continue even if command returns an error. 0 – Pause the queue if command returns an error.
0	InterruptOnCompletion	1 – Interrupt when command completes. 0 – No interrupt when command completes.

There are two bits defined for the Command Flags in a CmdPtr entry. Bit 0, as in Mailbox mode, is the request for the completion of this command to interrupt the x86. Bit 1 is the ContinueOnError flag. If it is 0, commands that return errors will block the queue, preventing any additional commands from being processed from the queue until the queue error recovery sequence is completed for the queue.

After the PSP processes a command from the CmdPtrs array, it writes the result in to the StatVal array, which parallels (by index) the CmdPtrs array. That is, after processing a command from the CmdPtrs array at index N, the result of that command will be written to the StatVal array at index N. The layout of a StatVal entry is shown in Table 8.

Table 8. StatVal Entry Layout

Byte Offset	Bits	Description
00h	15:0	Status.
02h	15:0	Reserved.
04h	31:0	Reserved.
08h	63:0	Reserved.

As mentioned earlier, in Ring Buffer mode, the three mailbox registers have different semantics. The RBCtl register is written only by the x86 and has the following layout.

Table 9. RBCtl Register Layout

Bit(s)	Name	Description/Purpose
31	X86Writes	Must be 1. 0 will exit Ring Buffer mode (indicates x86 controls the register).
30	RBModeAct	Must be 1. 0 will exit Ring Buffer mode (indicates Ring Buffer mode active.)
29	ClrIntStat	Clear Interrupt Status bits in RBHead register.
[28:10]	-	Reserved. Must be 0.
9	QHiResume	Resume Hi priority queue. (Part of queue error recovery.)
8	QLoResume	Resume Lo priority queue. (Part of queue error recovery.)
[7:2]	-	Reserved. Must be 0.
1	QHiFreeIntEn	Enable Hi priority queue near empty interrupt.
0	QLoFreeIntEn	Enable Lo priority queue near empty interrupt.

In Ring Buffer mode, the RBTail register is written only by the x86 and has the following layout:

Table 10. RBTail Register Layout

Bit(s)	Name	Description/Purpose
[31:24]	-	Reserved. Must be 0.
[23:16]	QHiTail	Hi priority Queue CmdPtr Tail index.
[15:8]	-	Reserved. Must be 0.
[7:0]	QLoTail	Lo priority Queue CmdPtr Tail index.

In Ring Buffer mode, the RBHead register is written only by the PSP and has the following layout:

Table 11. RBHead Register Layout

Bit(s)	Name	Description/Purpose
31	QFreeIntStat	A queue is near empty.
30	QPausedIntStat	A command without ContinueOnError set did not return SUCCESS.
29	CmdDoneIntStat	A command with InterruptOnCompletion set completed.
28	AllQEmptyIntStat	All queues are empty.
27	QHiRunning	Hi priority Queue is running.

Bit(s)	Name	Description/Purpose
[26:24]	-	Reserved.
[23:16]	QHiHead	Hi priority Queue CmdPtr Head index.
[15:12]	-	Reserved.
11	QLoRunning	Lo priority Queue is running.
[10:8]	-	Reserved.
[7:0]	QLoHead	Lo priority Queue CmdPtr Head index.

The x86 submits a command to a queue in ring buffer mode by creating any necessary buffers, writing the entry in the CmdPtr array at the desire Q*Tail index, incrementing the Q*Tail index (modulo the queue size provided in the RING_BUFFER command) in the RBTail register and finally writing an appropriate value to RBCtl. Multiple commands may be submitted at one time. Additional commands can be submitted any time the queue has room. The depth of the queue is (Q*Tail – Q*Head) modulo QSize.

The PSP, at the beginning of each command, clears the four IntStat bits in the RBHead register if ClrIntStat in the RBCtl register is set. Note that ClrIntStat is sampled only at the beginning of each command execution.

If an error occurs executing the command and the command didn't have ContinueOnError set, the PSP pauses the queue from which the command was fetched and clears that queue's Running bit.

At the end of each command, each IntStat bit may be set (and will stay set until ClrIntStat clears them) if the corresponding condition is met. If one or more of the conditions is met, one interrupt will be sent to the x86.

If the command caused the queue to be paused, the QPausedIntStat bit will be set. If the command had its InterruptOnCompletion flag set, CmdDoneIntStat bit will be set. If the RBCtl register has a Q*FreeIntEn bit set and the corresponding queue is below the threshold established with the RING_BUFFER command, the QFreeIntStat bit will be set. If both queues are empty and the RING_BUFFER command had the InterruptOnEmpty flag set, the AllQEmptyIntStat bit will be set. Once set, each IntStat bit will remain set until ClrIntStat is set when processing a later command.

If a queue is paused, it will stay paused until the x86 cleans up the queue. The x86 is free to manipulate any command in the queue and the queue tail index. The x86 may not adjust the queue head index. Setting the queue tail index equal to the queue head index will empty the queue. When the x86 is done cleaning up the queue, it shall set the queue's Resume bit in the RBCtl register. This bit must remain set until the PSP acknowledges seeing the Resume bit by setting the queue's Running bit in the RBHead register. When the x86 clears the Resume bit, the PSP will again start processing commands from the queue.

Table 12. Ring Buffer Mode Error Recovery Handshake Sequence

QRunning bit PSP Controlled	QResume Bit x86 Controlled	Description
1	0	Normal operation. Commands are accepted from the queue.
0	0	A command without ContinueOnError set has failed. The queue is paused.
0	1	The x86 has cleaned up the queue.
1	1	The PSP acknowledges clean up complete.
1	0	Normal operation resumes.

Ring Buffer mode exits if a SHUTDOWN command is executed or an invalid value is read from the RBCtl register (that is, one with either the x86Writes or RBModeAct bits clear, or one of the “must be zero” bits set). Mailbox mode is active after Ring Buffer mode is exited. If Ring Buffer mode exits because of an invalid value written to RBCtl, then the value in that register is ignored. If bit 31 of the mailbox mode CmdResp register is 0, indicating a command has been issued, the response shall be the status value RB_MODE_EXITED and the command will be ignored.

4.3 Command Buffer

In both Mailbox mode and Ring Buffer mode, the x86 system software is responsible for allocating the command buffer. The size of the command buffer is fixed for a given command. Some commands require the command buffer to be populated with pointers to secondary buffers. The x86 system software is also responsible for allocating these secondary buffers.

The command buffer and all secondary buffers must reside within memory accessible to the x86 system software. The SEV API enforces this by checking the validity of all pointers provided by the x86 system software before use.

All pointers provided by the x86 software are 64-bit system physical addresses.

4.4 Command Identifiers

Table 13 summarizes the platform management, guest management, and debugging commands. See the command definitions for further details. The API column indicates which version of the API the command first appeared in. The first public SEV API version was 0.16.

Table 13. Command Identifiers

Command	ID	API	Description
INIT	001h	16	Initialize the platform
INIT_EX	00Dh	18	Initialize the platform with extended parameters
SHUTDOWN	002h	16	Shut down the platform
PLATFORM_RESET	003h	16	Delete the persistent platform state

Command	ID	API	Description
PLATFORM_STATUS	004h	16	Return status of the platform
PEK_GEN	005h	16	Generate a new PEK
PEK_CSR	006h	16	Generate a PEK certificate signing request
PEK_CERT_IMPORT	007h	16	Import the signed PEK certificate
PDH_GEN	009h	16	Generate a new PDH and PEK signature
PDH_CERT_EXPORT	008h	16	Export the PDH and its certificate chains
DOWNLOAD_FIRMWARE	00Bh	16	Download new SEV firmware
GET_ID	00Ch	16	Get System Unique ID
NOP	00Eh	23	Nop command
RING_BUFFER	00Fh	24	Enter Ring Buffer command mode
LAUNCH_START	030h	16	Begin to launch a new SEV enabled guest
LAUNCH_UPDATE_DATA	031h	16	Encrypt guest data for launch
LAUNCH_UPDATE_VMSA	032h	16	Encrypt guest VMCB save area for launch
LAUNCH_MEASURE	033h	16	Output the launch measurement
LAUNCH_UPDATE_SECRET	034h	16	Import a guest secret sent from the guest owner
LAUNCH_FINISH	035h	16	Complete launch of guest
ATTESTATION	036h	23	Provide signed measurement to hypervisor
SEND_START	040h	16	Begin to send guest to new remote platform
SEND_UPDATE_DATA	041h	16	Re-encrypt guest data for transmission
SEND_UPDATE_VMSA	042h	16	Re-encrypt guest VMCB save area for transmission
SEND_FINISH	043h	16	Complete sending guest to remote platform
SEND_CANCEL	044h	23	Cancel a send operation in progress
RECEIVE_START	050h	16	Begin to receive guest from remote platform
RECEIVE_UPDATE_DATA	051h	16	Re-encrypt guest data from transmission
RECEIVE_UPDATE_VMSA	052h	16	Re-encrypt guest VMCB save area from transmission
RECEIVE_FINISH	053h	16	Complete receiving guest from remote platform
GUEST_STATUS	023h	16	Query the status and metadata of a guest
ACTIVATE	021h	16	Load a guest's key into the memory controller
DEACTIVATE	022h	16	Unload a guest's key into the memory controller
ACTIVATE_EX	025h	18	Load a guest's key into the memory controller
DF_FLUSH	00Ah	16	Flush the data fabric
DECOMMISSION	020h	16	Delete the guest's SEV context in firmware
COPY	024h	17	Copy encrypted guest memory to a new guest location

Command	ID	API	Description
SWAP_OUT	070h	23	Decrypt guest memory page and re-encrypt for temporary storage
SWAP_IN	071h	23	Decrypt temporarily-stored page and re-encrypt for guest use
DBG_DECRYPT	060h	16	Decrypt guest memory region for debugging
DBG_ENCRYPT	061h	16	Encrypt guest memory region for debugging

4.5 Status Codes

Table 14 summarizes the status codes that can be returned by the firmware commands. See the command definitions for further details.

Table 14. Status Codes

Status	Code	Description
SUCCESS	0x0000	Successful completion
INVALID_PLATFORM_STATE	0x0001	The platform state is invalid for this command
INVALID_GUEST_STATE	0x0002	The guest state is invalid for this command
INVALID_CONFIG	0x0003	The platform configuration is invalid
INVALID_LENGTH	0x0004	A memory buffer is too small.
ALREADY_OWNED	0x0005	The platform is already owned
INVALID_CERTIFICATE	0x0006	The certificate is invalid
POLICY_FAILURE	0x0007	Request is not allowed by guest policy
INACTIVE	0x0008	The guest is inactive
INVALID_ADDRESS	0x0009	The address provided is invalid
BAD_SIGNATURE	0x000A	The provided signature is invalid
BAD_MEASUREMENT	0x000B	The provided measurement is invalid
ASID_OWNED	0x000C	The ASID is already owned
INVALID_ASID	0x000D	The ASID is invalid
WBINVD_REQUIRED	0x000E	WBINVD instruction required
DF_FLUSH_REQUIRED	0x000F	DF_FLUSH invocation required
INVALID_GUEST	0x0010	The guest handle is invalid
INVALID_COMMAND	0x0011	The command issued is invalid.
ACTIVE	0x0012	The guest is active.

Status	Code	Description
HWERROR_PLATFORM	0x0013	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	0x0014	A hardware condition has occurred affecting the platform. Re-allocating parameter buffers is not safe.
UNSUPPORTED	0x0015	Feature is unsupported.
INVALID_PARAM	0x0016	A parameter is invalid.
RESOURCE_LIMIT	0x0017	The SEV FW has run out of a resource necessary to complete the command.
SECURE_DATA_INVALID	0x0018	The part-specific SEV data failed integrity checks.
RB_MODE_EXITED	0x001F	A Mailbox mode command was sent while the SEV FW was in Ring Buffer mode. Ring Buffer mode has been exited; the Mailbox mode command has been ignored. Retry is recommended.

4.6 Endianness

All integral values passed between the firmware and the CPU driver are little-endian formatted unless otherwise specified. That is, the first byte of the integer representation is the least significant byte. Note that this applies to all bit integer values in public keys and signatures unless explicitly stated otherwise.

4.7 Synchrony

The SEV API processes commands serially. The x86 system software receives a response when the SEV firmware has completed processing the command. The x86 system software must not send subsequent commands before receiving a response from the SEV firmware for the first sent command.

Any subsequent commands sent by the x86 system software before receiving a response must be ignored by the SEV firmware and must not result in any change to the platform or any guest context.

4.8 Address Validation

All addresses provided by x86 system software are checked by SEV firmware to ensure that the regions do not overlap invalid memory regions. Each address is checked that it does not have any of the following properties:

- Bits 46:43 != 0
- Overlaps TSseg
- Overlaps ASeg
- Is above the max physical address (7FD_0000_0000)
- Lies within any Trusted Memory Region (TMR) that disallows x86 software access

This check is performed before any action is taken on the contents of addresses provided by the x86 system software as parameters. If the check fails, an `INVALID_ADDRESS` status code is returned.

Chapter 5 Platform Management API

This chapter describes the platform management commands. Platform management commands are used by the platform owner to provision the platform and query platform-wide data.

5.1 Overview

This section overviews the platform management and provides the background to understand the semantics of each command.

5.1.1 Platform Context

The SEV firmware maintains a platform context throughout the lifetime of the platform. This context contains data and metadata necessary to implement the SEV API.

Table 15. Platform Context (PCTX) Fields

Field	Field Description
STATE	The current state of the platform in the PSTATE finite state machine.
CONFIG	The configuration of the platform.
PDH	Platform Diffie Hellman key.
PEK	Platform Endorsement Key.
CEK	Chip Endorsement Key.
OCA	Owner Certificate Authority key.
GUEST_COUNT	Number of guest contexts currently managed by the firmware.
GUESTS	The guest contexts currently managed by the firmware.

The platform context is denoted PCTX. The PCTX fields are referenced as PCTX.<FIELD>. For instance, the number of guests currently managed by the platform would be denoted PCTX.GUEST_COUNT. When unambiguous, the PCTX portion may be omitted.

5.1.2 Platform State Machine

The SEV firmware traverses a platform finite state machine during operation. The firmware may only execute certain commands in each platform state. Guest commands are only valid in INIT and WORKING platform states.

Table 16. PSTATE Finite State Machine

State	Value	Description	Allowed Platform Commands
UNINIT	0h	The platform is uninitialized.	INIT, PLATFORM_RESET, PLATFORM_STATUS, DOWNLOAD_FIRMWARE, GET_ID, DF_FLUSH, NOP
INIT	1h	The platform is initialized but not currently managing any guests.	SHUTDOWN, PLATFORM_STATUS, PEK_GEN, PEK_CSR, PEK_CERT_IMPORT, PDH_GEN, PDH_CERT_EXPORT, DF_FLUSH, GET_ID, NOP, RING_BUFFER
WORKING	2h	The platform is initialized and current managing guests.	SHUTDOWN, PLATFORM_STATUS, PDH_GEN, PDH_CERT_EXPORT, DF_FLUSH, GET_ID, NOP, RING_BUFFER

The platform finite state machine is denoted PSTATE. The PSTATE states are referenced as PSTATE.<STATE>. For instance, the uninitialized state is denoted PSTATE.UNINIT.

5.1.3 Authenticity

An authentic platform is a platform that is executing AMD developed firmware on AMD hardware. Authenticity allows the guest owner to trust that the platform it communicates with will implement SEV correctly and securely.

The authenticity of the platform is defined by a certificate chain rooted in the ARK, the AMD root of trust. The ARK signs the ASK, which in turn signs the CEK. The CEK signs the PEK which roots the PEK in the AMD root of trust, asserting that this platform is an authentic AMD platform.

5.1.4 Ownership

A platform may be owned by an external entity or it may be self-owned. Platforms can disallow migration of guests to platforms that do not share the same owner. This allows platform owners to contain guests within a known set of servers.

The platform ownership is defined by a certificate chain rooted in the OCA, the owner's root of trust. The OCA signs the PEK. When the platform is not owned by an external entity, the platform generates its own OCA key pair.

5.1.5 Non-volatile Storage

The platform stores identity information in non-volatile storage available to the firmware. The identity information includes the following:

- PDH key pair
- PDH certificate
- PEK key pair
- PEK certificate
- OCA public key
- OCA private key (only if self-owned)
- OCA certificate

The firmware writes each of these keys immediately after generation, derivation, or import. The persistent data is stored encrypted and integrity protected.

If a power failure occurs during a write operation to non-volatile storage, the persistent data may become corrupted. Upon executing the INIT command immediately after the power failure, the command will return an integrity check failure status code. In such a case, the platform must be reset with the PLATFORM_RESET command.

5.1.6 Power State Transitions

Table 17 summarizes the effects of power state transitions on the firmware.

Table 17. Power Management Transitions

Power State	Effect
S0, S1, S2, S3	The firmware volatile state is not affected and resumes execution after transitioning out of the low power states.
Hybrid Sleep	This is equivalent to preparing to transition to S4, but transitioning to S3. If a power failure occurs, the system resumes as if it transitioned to S4. If not, it the system resumes as if it transitioned to S3.
S4, S5, Mechanical Off	The firmware starts from the reset state and must be re-initialized. No non-persistent state is recovered from the point before the power state transition.

5.1.7 SEV-ES Trusted Memory Region

The SEV-ES product feature requires that the x86 system software donate a 1 MB region aligned on a 1 MB boundary to the firmware for exclusive use by the firmware during the lifetime of the platform (until a SHUTDOWN command invocation). The firmware uses the memory region to store integrity data necessary to support the SEV-ES functionality. Attempts by the x86 to access this memory while donated will result in exceptions that frequently cause system instability.

5.2 INIT

This command is used by the platform owner to initialize the platform. This command loads the SEV related persistent data from non-volatile storage and initializes the platform context. In typical workflows, this command should be the first command issued, aside from the possible invocation of PLATFORM_STATUS to determine the supported API version.

5.2.1 Actions

The platform must be in the PSTATE.UNINIT state.

The firmware first loads the persistent state into its private memory, and validates it. If validation fails, an error is returned and the persistent state is erased. If validation succeeds, then the following actions are performed:

- The CEK is derived from the chip unique values.
- If no OCA certificate exists, an OCA signing key is generated and a self-signed OCA certificate is created. The signing key and certificate are written to persistent storage.
- If no PEK exists or the OCA was just regenerated, a PEK signing key is generated and a PEK certificate is created and signed by the OCA and CEK. The PEK and its certificate are written to persistent storage.
- If no PDH exists or the PEK was just regenerated, a PDH key is generated. A certificate is created for the PDH and is signed by the PEK.
- All SEV-related ASIDs on all cores are marked invalid. Each core requires a WBINVD before activating any guest. See ACTIVATE and DEACTIVATE for further details.

Upon successful completion, the platform transitions to the PSTATE.INIT state.

5.2.2 Parameters

Table 18 specifies the parameters for the INIT command.

Table 18. INIT Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	0	In	–	SEV-ES is initialized for the platform when set. It is disabled for all guests when not set.
	31:1	–	–	Reserved. Must be zero.
04h	31:0	–	–	Reserved. Must be zero.
08h	63:0	In	TMR_PADDR	If CONFIG.ES=1, system physical address to memory region donated by hypervisor for SEV-ES operations. Must be 1MB aligned. Ignored if CONFIG.ES=0

Byte Offset	Bits	In / Out	Name	Description
10h	31:0	In	TMR_LENGTH	If CONFIG.ES=1, length of the memory region donated by hypervisor for SEV-ES operation. Must equal 1MB. Ignored if CONFIG.ES=0

5.2.3 Status Codes

Table 19 enumerates the possible status codes returned by this command.

Table 19. INIT Command Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PARAM	A parameter is invalid
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.UNINIT state
INVALID_LENGTH	A buffer length is not correct
INVALID_CONFIG	The configuration flags are invalid or unsupported
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
INVALID_SECURE_DATA	Data in non-empty SPI failed integrity checks.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.3 INIT_EX

The INIT_EX command is used by the platform owner to initialize the platform with extended parameters. This command loads the SEV related persistent data from user-supplied data and initializes the platform context. In typical workflows, this command should be the first command issued, aside from the possible invocation of PLATFORM_STATUS to determine the supported API version.

5.3.1 Actions

The platform must be in the PSTATE.UNINIT state.

The firmware first loads the persistent state into its private memory, and validates it. If validation fails, an error is returned and the persistent state is erased. If validation succeeds, then the following actions are performed:

- The CEK is derived from the chip unique values.
- If no OCA certificate exists, an OCA signing key is generated and a self-signed OCA certificate is created. The signing key and certificate are written to persistent storage.
- If no PEK exists or the OCA was just regenerated, a PEK signing key is generated and a PEK certificate is created and signed by the OCA and CEK. The PEK and its certificate are written to persistent storage.
- If no PDH exists or the PEK was just regenerated, a PDH key is generated. A certificate is created for the PDH and is signed by the PEK.
- All SEV-related ASIDs on all cores are marked invalid. Each core requires a WBINVD before activating any guest. See ACTIVATE and DEACTIVATE for further details.

Upon successful completion, the platform transitions to the PSTATE.INIT state.

The area of memory pointed to by NV_PADDR must be reserved for SEV FW use from the time of the INIT_EX call until SHUTDOWN. The uninitialized empty state of this memory is all bytes FFh. The caller is expected to preserve the 32KB area so that it may be passed in to a future INIT_EX call. The contents of the area are tied to the specific CPU in use and cannot be used with any other CPU, even of the same model. The area will only change for some platform commands (Chapter 5.) No other commands alter the area. If NV_PADDR is 0, the SPI will be used instead.

5.3.2 Parameters

Table 20 specifies the parameters for the INIT_EX command.

Table 20. INIT_EX Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	EX_LEN	Length of the command buffer. 24h for 0.22 of the API.
04h	0	In	FLAGS	SEV-ES is initialized for the platform when set. It is disabled for all guests when not set.
	31:1	In	-	Reserved. Must be zero.
08h	63:0	In	TMR_PADDR	If CONFIG.ES=1, system physical address to memory region donated by hypervisor for SEV-ES operations. Must be 1MB aligned. Ignored if CONFIG.ES=0
10h	31:0	In	TMR_LENGTH	If CONFIG.ES=1, length of the memory region donated by hypervisor for SEV-ES operation. Must equal 1MB. Ignored if CONFIG.ES=0
14h	31:0	In	-	Reserved. Must be zero.

Byte Offset	Bits	In / Out	Name	Description
18h	63:0	In	NV_PADDR	System Physical Address of an area of memory the SEV FW will use in lieu of SPI. Must be 4KB aligned.
20h	31:0	In	NV_LENGTH	Length of area in bytes to be used in lieu of SPI. Must be 32KB.

5.3.3 Status Codes

Table 21 enumerates the possible status codes returned by this command.

Table 21. INIT_EX Command Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PARAM	A parameter is invalid
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.UNINIT state
INVALID_LENGTH	A buffer length is not correct
INVALID_CONFIG	The configuration flags are invalid or unsupported
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
INVALID_SECURE_DATA	Data in non-empty NV memory area or SPI failed integrity checks.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.4 SHUTDOWN

The SHUTDOWN command is used by the platform owner to transition the platform to the uninitialized state.

5.4.1 Actions

The platform may be in any state.

All platform and guest state maintained by the firmware is securely deleted from volatile storage.

If SEV-ES is enabled during platform initialization, the firmware returns control of the SEV-ES TMR back to the x86. After this command completes, the TMR will be accessible by x86 software.

Upon successful completion, the platform transitions to the PSTATE.UNINIT state. The mailboxes revert to Mailbox mode.

If the SHUTDOWN is for an INIT for SEV-ES, then a WBINVD on all threads is required immediately following the completion of the SHUTDOWN command.

5.4.2 Parameters

None. CmdBuf_Lo and CmdBuf_Hi registers are ignored.

5.4.3 Status Codes

Table 22 enumerates the possible status codes returned by this command.

Table 22. SHUTDOWN Command Status Codes

Return Value	Reason
SUCCESS	Successful completion
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.5 PLATFORM_RESET

The PLATFORM_RESET command is used by the platform owner to reset the non-volatile SEV related data. Invoking this command is useful when the owner wishes to transfer the platform to a new owner or securely dispose of the system.

5.5.1 Actions

The platform must be in the PSTATE.UNINIT state. Otherwise, an error is returned.

The persistent state is securely deleted from non-volatile storage.

The platform remains in the PSTATE.UNINIT state after completion.

5.5.2 Parameters

None. CmdBuf_Lo and CmdBuf_Hi registers are ignored.

5.5.3 Status Codes

Table 23 enumerates the possible status codes returned by the PLATFORM_RESET command.

Table 23. PLATFORM_RESET Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.UNINIT state.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.6 PLATFORM_STATUS

The PLATFORM_STATUS command is used by the platform owner to collect the current status of the platform.

5.6.1 Actions

The platform may be in any state.

If the platform is in the PSTATE.UNINIT state, then the following fields are set to zero:

- OWNER
- CONFIG.ES
- GUEST_COUNT

The OWNER bit is set to 1 if the platform is owned by an external owner. The OWNER bit is set to 0 if the platform is self-owned.

The CONFIG flags are equal to the values passed to the INIT command.

The platform remains in the same state after completion.

5.6.2 Parameters

Table 24 specifies the parameters for the PLATFORM_STATUS command.

Table 24. PLATFORM_STATUS Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	7:0	Out	API_MAJOR	Major API version
01h	7:0	Out	API_MINOR	Minor API version
02h	7:0	Out	STATE	Current platform state. See Table 3.

Byte Offset	Bits	In / Out	Name	Description
03h	0	Out	OWNER	OWNER=0: Self-owned OWNER=1: Externally owned.
	7:1	–	–	Reserved. Set to zero.
04h	0	Out	CONFIG.ES	SEV-ES is initialized for the platform when set. It is disabled for all guests when not set.
	23:1	–	–	Reserved. Set to zero.
	31:24	Out	BUILD	Firmware Build ID for this API version.
08h	31:0	Out	GUEST_COUNT	Number of valid guests maintained by the firmware

5.6.3 Status Codes

Table 25 enumerates the possible status codes returned by the PLATFORM_STATUS command.

Table 25. PLATFORM_STATUS Command Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_ADDRESS	A memory region provided contains invalid physical addresses

5.7 PEK_GEN

This command is used to generate a new PEK. This may be used to regenerate the identity of the platform if required by the user's policy. This command is not necessary for normal workflows as the INIT command regenerates this after being invoked for the first time after a platform reset.

5.7.1 Actions

The platform must be in the PSTATE.INIT state. Otherwise, an error is returned.

The following objects are securely deleted from volatile and non-volatile storage:

- PEK key pair
- PEK certificate
- PDH key pair
- PDH certificate
- OCA key pair (if the platform is self-owned)
- OCA certificate

A new OCA signing key pair and self-signed certificate are generated and stored in non-volatile storage. The previous OCA certificate (and signing key if self-owned) is securely deleted.

A new PEK signing key pair and certificate signed by the new OCA are generated and stored in non-volatile storage. The previous PEK signing key and certificate are securely deleted.

A new PDH key pair is generated and signed by the PEK. Regeneration of the PDH is equivalent to the invocation of the PDH_GEN command.

Note that in this version of the API, this command is an alias for an invocation of SHUTDOWN, PLATFORM_RESET, and INIT, in that order.

The platform remains in the same state after completion.

5.7.2 Parameters

None. The CmdBuf_Lo and CmdBuf_Hi mailbox registers are ignored.

5.7.3 Status Codes

Table 26 enumerates the possible status codes returned by the PEK_GEN command.

Table 26. PEK_GEN Command Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.INIT state
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.8 PEK_CSR

During the provisioning process, the owner of the platform can take ownership using this command in conjunction with the PEK_CERT_IMPORT command. The CSR contains identifying information of this platform and also contains the PEK public key. A certificate authority processes the CSR by generating a certificate with the information from the CSR and signing it with its key.

The caller can issue this command with the PEK_CSR_LEN field set to zero to query the correct amount of memory to allocate for the CSR.

5.8.1 Actions

The platform must be in the PSTATE.INIT or PSTATE.WORKING states. Otherwise, an error is returned.

A CSR is generated which contains the PEK public key to be signed. The CSR format is identical to the SEV certificate with zero signatures. See Appendix C on page 118 for the certificate format.

If the PEK_CSR_LEN is too small, the required length is written out to that field and an error is returned. Otherwise, the number of bytes written to the buffer are written into PEK_CSR_LEN.

The platform remains in the same state after completion.

5.8.2 Parameters

Table 27 specifies the parameters for the PEK_CSR command.

Table 27. PEK_CSR Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	63:0	In	PEK_CSR_PADDR	System physical address of the region to output the PEK certificate signing request. See Appendix C for layout.
08h	31:0	In, Out	PEK_CSR_LEN	Length of the region to output the PEK certificate signing request in bytes.

5.8.3 Status Codes

Table 28 enumerates the possible status codes returned by the PEK_CSR command.

Table 28. PEK_CSR Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	A length field is invalid.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.INIT or PSTATE.WORKING states.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.9 PEK_CERT_IMPORT

During the provisioning process, the platform can be joined to a domain using this command in conjunction with the PEK_CSR command. The CSR contains the identifying information of this platform and the PEK public key. A certificate authority processes the CSR by generating a certificate with the information from the CSR and signing it with its key. This command imports the certificate of the PEK and OCA into the platform.

Note that this provisioning process should be performed by the platform owner in a trusted environment. In an untrusted environment—particularly where the x86 software is untrusted—this request could be intercepted and replaced with a certificate chain crafted by the attacker.

5.9.1 Actions

The platform must be in the PSTATE.INIT state. Otherwise, an error is returned.

The platform must be self-owned. This requirement ensures that the caller has regenerated the PEK via PEK_GEN and therefore PEKs will never be shared by owners.

The OCA and PEK certificates are validated. Validation involves the following steps:

- The algorithms of the PEK and OCA must be supported
- The version of the PEK and OCA certificates must be supported
- The PEK certificate must match the current PEK
- The OCA signature on the PEK certificate must be valid

The OCA certificate and PEK signature are then written into the platform context as well as to non-volatile storage.

The PDH is regenerated and signed with the new PEK. This action is equivalent to the PDH_GEN command.

The platform remains in the same state after completion.

5.9.2 Parameters

Table 29 specifies the parameters for the PEK_CERT_IMPORT command.

Table 29. PEK_CERT_IMPORT Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	63:0	In	PEK_CERT_PADDR	System physical address of the region containing the PEK certificate. See Appendix C for layout.
08h	31:0	In	PEK_CERT_LEN	Length of the region containing the PEK certificate, in bytes.

Byte Offset	Bits	In / Out	Name	Description
0Ch	31:0	—	—	Reserved. Must be zero.
10h	63:0	In	OCA_CERT_PADDR	System physical address of the region containing the OCA certificate. See Appendix C.
18h	31:0	In	OCA_CERT_LEN	First certificate in the chain. Signed by CERT2. Length of the region containing the OCA certificate, in bytes.

5.9.3 Status Codes

Table 30 enumerates the possible status codes returned by the PEK_CERT_IMPORT command.

Table 30. PEK_CERT_IMPORT Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.INIT states.
INVALID_LENGTH	A buffer length is not correct.
ALREADY_OWNED	The platform is already owned.
INVALID_CERTIFICATE	A provided certificate is invalid or the PEK does not match the current PEK.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.10 PDH_GEN

This command may be used to re-generate the PDH as often as desired. Note that if other entities are using the current PDH to establish keys to encrypt data or provide integrity checking, regenerating the PDH will invalidate any ongoing key establishment work. In this case, the other entities must retrieve the new PDH in order to perform key establishment.

5.10.1 Actions

The platform must be in the PSTATE.INIT or PSTATE.WORKING states. Otherwise, an error is returned.

The PDH key pair is regenerated. The PDH certificate is created and signed by the PEK. The current PDH key pair and certificate are replaced with the new PDH key pair and certificate.

The platform remains in the same state after completion.

5.10.2 Parameters

None. The CmdBuf_Lo and CmdBuf_Hi mailbox registers are ignored.

5.10.3 Status Codes

Table 31 enumerates the possible status codes returned by the PHD_GEN command.

Table 31. PDH_GEN Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform must be in the PSTATE.INIT or PSTATE.WORKING states.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.11 PDH_CERT_EXPORT

This command is used to retrieve the PDH and identity of the platform. This information may then be exported to remote entities which wish to establish a secure transport context with the platform in order to transmit data securely.

5.11.1 Actions

The platform must be in the PSTATE.INIT or PSTATE.WORKING states. Otherwise, an error is returned.

Exports the following data that can be used by an external party to authenticate the identity of the platform and establish keys:

- The PDH certificate
- The CEK certificate
- The PEK certificate
- The OCA certificate
- If the PDH_CERT_LEN or CERT_LEN fields are too small, the required length is written out to those fields and an error is returned. Otherwise, the number of bytes written to the buffers are written into PDH_CERT_LEN and CERT_LEN, respectively.

The platform remains be in the same state after completion.

5.11.2 Parameters

Table 32 and Table 33 specify the parameters for the PDH_CERT_EXPORT command.

Table 32. PDH_CERT_EXPORT Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	63:0	In	PDH_CERT_PADDR	System physical address of the region containing the PDH certificate. See Appendix C for layout.
08h	31:0	In, Out	PDH_CERT_LEN	Length of the region containing the PDH certificate, in bytes.
0Ch	31:0	—	—	Reserved. Must be zero.
10h	63:0	In	CERTS_PADDR	System physical address of the region containing the PDH certificate chain. See Table 33 for the layout.
18h	31:0	In, Out	CERTS_LEN	Length of the region containing the PDH certificate chain, in bytes.

Table 33. PDH_CERT_EXPORT Certificates Buffer

Byte Offset	Bits	In / Out	Name	Description
0000h – 0823h	—	Out	PEK_CERT	The PEK certificate. See Appendix C for layout.
0824h – 1047h	—	Out	OCA_CERT	The OCA certificate. See Appendix C for layout.
1048h – 186Bh	—	Out	CEK_CERT	The CEK certificate. See Appendix C for layout.

5.11.3 Status Codes

Table 34 enumerates the possible status codes returned by the PDH_CERT_EXPORT command.

Table 34. PDH_CERT_EXPORT Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	A buffer length is not correct.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.INIT or PSTATE.WORKING states.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.

Return Value	Reason
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

5.12 DOWNLOAD_FIRMWARE

The DOWNLOAD_FIRMWARE command is provided to allow operating systems to install SEV firmware newer than the currently active SEV firmware. This command was added in SEV API v0.15.

5.12.1 Actions

The platform must be in the PSTATE.UNINIT. Otherwise, an error is returned.

The OS provided firmware image will be validated by the current firmware. If validation fails, an error is returned and the current firmware remains active. If validation succeeds, the current firmware is replaced with the OS provided firmware. The provided firmware image must be the same build or newer than the current firmware. The new firmware image will remain active only until a system reset.

The platform remains in the same state after completion.

5.12.2 Parameters

Table 35 specifies the parameters for the DOWNLOAD_FIRMWARE command.

Table 35. DOWNLOAD_FIRMWARE Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	63:0	In	FW_PADDR	System physical address of the region that contains a proposed SEV FW image. This region must be 32-byte aligned.
08h	31:0	In	FW_LEN	Length of the proposed SEV FW image in bytes.

5.12.3 Status Codes

Table 36 enumerates the possible status codes returned by the `DOWNLOAD_FIRMWARE` command.

Table 36. DOWNLOAD_FIRMWARE Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	The length field is invalid.
INVALID_PLATFORM_STATE	The platform is not in the <code>PSTATE.UNINIT</code> state.
INVALID_ADDRESS	The memory region provided contains invalid physical addresses.
BAD_SIGNATURE	The proposed FW image has a bad signature.
INVALID_PARAM	The proposed FW image is not acceptable to the current FW.

5.13 GET_ID

The `GET_ID` command provides a unique ID for the CPU that can be used to obtain a certificate for the CEK public key that is signed by the ASK. This command was added in SEV API v0.16.

5.13.1 Actions

The platform may be in any state.

If the value of the `ID_LEN` field is too small, an `INVALID_LENGTH` error is returned and the minimum required length is written to the field. Otherwise, the ID of the master CPU (Processor 0) is returned in the area pointed to by `ID_PADDR` and the length of the ID value written is written to the `ID_LEN` field. If `ID_LEN` is precisely two times the minimum length required, and the system has 2 CPUs, the ID value for both CPUs is returned in the area pointed to by `ID_PADDR` and the `ID_LEN` field is updated to reflect the number of bytes written at `ID_PADDR`.

The platform remains in the same state after completion.

5.13.2 Parameters

Table 37 specifies the parameters for the `GET_ID` command.

Table 37. GET_ID Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	63:0	In	<code>ID_PADDR</code>	System physical address of the region that will be filled with the CPU ID.
08h	31:0	In, Out	<code>ID_LEN</code>	Length of the region.

5.13.3 Status Codes

Table 38 enumerates the possible status codes returned by the GET_ID command.

Table 38. GET_ID Command Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	The length field is invalid.
INVALID_ADDRESS	The memory region provided contains invalid physical addresses.

5.14 NOP

The NOP command does nothing. This command was added in SEV API v0.23 to support generation of interrupts from command completion in Ring Buffer mode without advance knowledge of the ‘last’ command in a batch.

5.14.1 Actions

The platform may be in any state.

There are no parameters. The command can return only SUCCESS.

The platform remains in the same state after completion.

5.14.2 Parameters

There are no parameters to the NOP command.

5.14.3 Status Codes

Table 39 enumerates the possible status codes returned by the NOP command.

Table 39: NOP Status Codes

Return Value	Reason
SUCCESS	Successful completion.

5.15 RING_BUFFER

The RING_BUFFER command causes commands to be fetched from a pair of queues, each implemented by a pair of ring buffers to allow parallelization of command creation on the x86 side with command execution by the SEV FW. This command was added in SEV API v0.24.

5.15.1 Actions

The platform may be in any state except UNINIT. The command is valid only as a Mailbox mode command. (That is, a RING_BUFFER command cannot be submitted from a ring buffer.) The RING_BUFFER command will have its status reported via Mailbox mode. Ring Buffer mode will be active only after the x86 writes the RBCtl register with a value meeting the requirements described in 4.2.

Details of the usage of ring buffers are detailed in Section 4.2, Ring Buffer Mode.

The platform remains in the same state after completion.

5.15.2 Parameters

Table 40 specifies the parameters for the RING_BUFFER command.

Table 40. RING_BUFFER Command Buffer

Byte Offset	Bit(s)	In / Out	Name	Description
00h	63:0	In	QLoCmdPtr	System physical address of the 4K aligned region to be used for the low priority queue's CmdPtr ring buffer.
08h	63:0	In	QLoStatVal	System physical address of the 4K aligned region to be used for the low priority queue's StatVal ring buffer.
10h	63:0	In	QHiCmdPtr	System physical address of the 4K aligned region to be used for the high priority queue's CmdPtr ring buffer.
18h	63:0	In	QHiStatVal	System physical address of the 4K aligned region to be used for the high priority queue's StatVal ring buffer.
20h	7:0	In	QLoSize	Size of the low priority queue in 4K pages. Must be 1.
21h	7:0	In	QHiSize	Size of the high priority queue in 4K pages. Must be 0 or 1. If 0, all other QHi values are ignored.
22h	15:0	In	QLoThreshold	Queue size, in entries, below which an interrupt may optionally be generated.

Byte Offset	Bit(s)	In / Out	Name	Description
24h	15:0	In	QHiThreshold	Queue size, in entries, below which an interrupt may optionally be generated.
26h	0	In	IntOnEmpty	Unconditionally interrupt when both queues are found empty.
	15:1	In	-	Reserved. Must be 0.

5.15.3 Status Codes

Table 41 enumerates the possible status codes returned by the RING_BUFFER command.

Table 41. RING_BUFFER Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	The length field is invalid.
INVALID_ADDRESS	The memory region provided contains invalid physical addresses.
INVALID_PLATFORM_STATE	The platform state is UNINIT.
INVALID_PARAM	A parameter is invalid.
INVALID_COMMAND	RING_BUFFER is only valid in Mailbox mode.

Chapter 6 Guest Management API

This chapter describes the management of guest contexts throughout the guest lifecycle.

6.1 Overview

This chapter describes the SEV context of an SEV-enabled guest.

6.1.1 Guest Context

The SEV firmware maintains guest contexts throughout the lifetime of the platform. The guest context contains data and metadata necessary to implement the SEV API specific to a guest.

Table 42. Guest Context (GCTX) Fields

Field	Field Description
STATE	The current state of this guest in the GSTATE finite state machine
HANDLE	The handle of this guest
ASID	The ASID currently associate with this guest.
ACTIVE	Flag describing whether this guest is currently active
POLICY	This guest's policy
VEK	The memory encryption key of the guest
NONCE	The trusted channel nonce currently associated with this guest
MS	The master secret current associated with this guest
TEK	The transport encryption key currently associated with this guest
TIK	The transport integrity key currently associated with this guest
LD	The launch digest context for this guest

A guest context is denoted GCTX. The GCTX fields are referenced as GCTX.<FIELD>. The GCTX prefix is omitted if it's unnecessary in context.

Table 43. GSTATE Finite State Machine

State	Value	Description	Allowed Guest Commands
UNINIT	0h	The guest is uninitialized.	LAUNCH_START, RECEIVE_START
LUPDATE	1h	The guest is currently being launched and plaintext data and VMCB save areas are being imported.	LAUNCH_UPDATE_DATA, LAUNCH_UPDATE_VMSA, LAUNCH_MEASURE, ACTIVATE, DEACTIVATE, DECOMMISSION, GUEST_STATUS

Table 43. GSTATE Finite State Machine (Continued)

State	Value	Description	Allowed Guest Commands
LSECRET	2h	The guest is currently being launched and ciphertext data are being imported.	LAUNCH_SECRET, LAUNCH_FINISH, ACTIVATE, DEACTIVATE, DECOMMISSION, GUEST_STATUS, ATTESTATION
RUNNING	3h	The guest is fully launched or migrated in, and not being migrated out to another machine.	ACTIVATE, DEACTIVATE, DECOMMISSION, SEND_START, GUEST_STATUS, ATTESTATION
SUPDATE	4h	The guest is currently being migrated out to another machine.	SEND_UPDATE_DATA, SEND_UPDATE_VMSA, SEND_FINISH, ACTIVATE, DEACTIVATE, DECOMMISSION, GUEST_STATUS, ATTESTATION, SEND_CANCEL
RUPDATE	5h	The guest is currently being migrated from another machine.	RECEIVE_UPDATE_DATA, RECEIVE_UPDATE_VMSA, RECEIVE_FINISH, ACTIVATE, DEACTIVATE, DECOMMISSION, GUEST_STATUS
SENT	6h	The guest has been sent to another machine.	DEACTIVATE, DECOMMISSION

Each SEV guest managed by the SEV firmware traverses a finite state machine during operation. The firmware will only execute certain commands in each guest state.

The guest finite state machine is denoted GSTATE. The GSTATE states are referenced as GSTATE.<STATE>. For instance, the uninitialized state is denoted GSTATE.UNINIT.

6.1.2 Activation and Deactivation

The number of ASIDs that can be associated with a VEK and used with SEV are limited. The limit is implementation specific and can be retrieved via CPUID Fn8000_001F[ECX]. To allow for the execution of a larger number of guests, the SEV firmware provides the ability to activate and deactivate guests.

6.2 LAUNCH_START

This command is used to bootstrap a guest by encrypting its memory with a new VEK. This command creates a guest context managed by the SEV firmware which can then be referred to by the handle passed out to the caller.

6.2.1 Actions

A new guest context is created and assigned to an available guest handle.

If the `HANDLE` field is zero, a new VEK is generated for this guest. If the `HANDLE` field is non-zero, the following checks are performed:

- `HANDLE` is a valid guest
- `GUESTS[HANDLE].POLICY` is equal to the `POLICY` field
- `POLICY.NOKS` is zero

If the above checks pass, then the VEK from the guest referred to by `HANDLE` is copied into the new guest context.

The handle of the new guest is written to the `HANDLE` field.

The launch digest stored in `GCTX.LD` field is initialized.

If `POLICY.ES` is set and the platform is configured with SEV-ES, then SEV-ES is enabled. If `POLICY.ES` is set and the platform is not configured with SEV-ES, an error is returned. Otherwise, SEV-ES is disabled.

A version check is performed to ensure that the API version implemented on the platform provides the proper functionality requested by the initiator of the guest VM. The major API version of this platform must be greater than the guest's `POLICY.API_MAJOR` field value, or the major API version is equal to `POLICY.API_MAJOR`, and the minor API version of this platform is greater than or equal to `POLICY.API_MINOR`. If all of these conditions are not met then an error is returned.

If `DH_CERT_PADDR` is equal to 0h, then the `DH_CERT_LEN`, `SESSION_PADDR`, and `SESSION_LEN` fields are ignored. The subsequent launch commands will use 0h for both the `TEK` and `TIK`.

After successful completion of this command, the guest is transitioned to the `GSTATE.LUPDATE` state.

The platform transitions to the `PSTATE.WORKING` state.

6.2.2 Parameters

Table 44 and Table 45 specify the parameters for the LAUNCH_START command.

Table 44. LAUNCH_START Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In, Out	HANDLE	Guest handle.
04h	31:0	In	POLICY	Guest policy.
08h	63:0	In	DH_CERT_PADDR	System physical address of the region containing the guest owner DH certificate. See Appendix C for the layout.
10h	31:0	In	DH_CERT_LEN	Length of the region containing the guest owner DH certificate, in bytes.
14h	31:0	–	–	Reserved. Must be zero.
18h	63:0	In	SESSION_PADDR	System physical address of the region containing the session parameters.
20h	31:0	In	SESSION_LEN	Length of the region containing the session parameters, in bytes.

Table 45. LAUNCH_START Session Data Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	127:0	In	NONCE	Session nonce.
10h	255:0	In	WRAP_TK	The TEK and TIK concatenated and encrypted with the KEK.
30h	127:0	In	WRAP_IV	IV for WRAP_TK encryption.
40h	255:0	In	WRAP_MAC	HMAC for WRAP_TK using KIK.
60h	255:0	In	POLICY_MAC	HMAC for POLICY using TIK.

6.2.3 Status Codes

Table 46 enumerates the possible status codes returned by the LAUNCH_START command.

Table 46. LAUNCH_START Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in PSTATE.INIT or WORKING state.
INVALID_LENGTH	A buffer length is not correct.

Table 46. LAUNCH_START Status Codes (Continued)

Return Value	Reason
INVALID_GUEST	The guest handle is invalid.
POLICY_FAILURE	The guest policy was violated.
BAD_MEASUREMENT	Failed to verify the integrity of a field covered by a MAC.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
UNSUPPORTED	SEV-ES has not been configured.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.3 LAUNCH_UPDATE_DATA

This command is used to encrypt guest data with its VEK.

6.3.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.LUPDATE state. The guest must be activated with an ASID using the ACTIVATE command.

The GCTX.LD is updated with the plaintext contents of the memory region pointed to by PADDR. The plaintext context is then encrypted with the guest's VEK in place.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.3.2 Parameters

Table 47 specifies the parameters for the LAUNCH_UPDATE_DATA command.

Table 47. LAUNCH_UPDATE_DATA Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	PADDR	System physical address of the data to be encrypted. Must be 16 B aligned.

Byte Offset	Bits	In / Out	Name	Description
10h	31:0	In	LENGTH	Length of the data to be encrypted. Must be a multiple of 16 B.

6.3.3 Status Codes

Table 48 enumerates the possible status codes returned by the LAUNCH_UPDATE_DATA command.

Table 48. LAUNCH_UPDATE_DATA Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.LUPDATE state.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.4 LAUNCH_UPDATE_VMSA

This command is used to encrypt the guest VMCB save area with its VEK. This command is only allowed when SEV-ES is enabled for this guest and when SEV-ES is configured for the platform.

6.4.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.LUPDATE state. The guest must be activated with an ASID using the ACTIVATE command.

The GCTX.LD is updated with the plaintext contents of the VMCB save area pointed to by PADDR. The VMCB save area is prepared for SEV-ES usage and is then encrypted with the guest’s VEK in place.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.4.2 Parameters

Table 49 specifies the parameters for the LAUNCH_UPDATE_VMSA command.

Table 49. LAUNCH_UPDATE_VMSA Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	PADDR	System physical address of the data to be encrypted. Must be 16 B aligned.
10h	31:0	In	LENGTH	Length of the data to be encrypted. Must be 4096 B.

6.4.3 Status Codes

Table 50 enumerates the possible status codes returned by the LAUNCH_UPDATE_VMSA command

Table 50. LAUNCH_UPDATE_VMSA Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct
INVALID_GUEST_STATE	The guest state is not in the GSTATE.LUPDATE state.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
UNSUPPORTED	SEV-ES has not been configured.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.5 LAUNCH_MEASURE

This command returns the measurement of the launched guest’s memory pages and VMCB save areas (if ES is enabled). The measurement is keyed with the TIK so the guest owner can use the measurement to verify that the guest was properly launched without tampering.

6.5.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.LUPDATE state.

If the MEASURE_LEN is too small, the required length is written out to that field and an error is returned. Otherwise, the number of bytes written to the buffer are written into MEASURE_LEN.

GCTX.LD is finalized, producing the hash digest of all plaintext data imported into the guest.

The launch measurement is calculated as:

HMAC(0x04 || API_MAJOR || API_MINOR || BUILD || GCTX.POLICY || GCTX.LD || MNONCE; GCTX.TIK)

where “||” represents concatenation. The launch measurement is written into the MEASURE field.

The firmware generates a random MNONCE.

The platform remains in the same state after completion.

The guest transitions to the GSTATE.LSECRET state.

6.5.2 Parameters

Table 51 and Table 52 specify the parameters for the LAUNCH_MEASURE command.

Table 51. LAUNCH_MEASURE Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	MEASURE_PAD DR	System physical address of the region containing the launch measurement. See Table 52 for layout.
10h	31:0	In, Out	MEASURE_LEN	Length of the region containing the launch measurement, in bytes.

Table 52. LAUNCH_MEASURE Measurement Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	255:0	Out	MEASURE	Measurement of the launched guest
20h	127:0	Out	MNONCE	Firmware-generated random nonce

6.5.3 Status Codes

Table 53 enumerates the possible status codes returned by the LAUNCH_MEASURE command.

Table 53. LAUNCH_MEASURE Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.LUPDATE state
INVALID_GUEST	The guest handle is invalid
INVALID_ADDRESS	A memory region provided contains invalid physical addresses
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.6 LAUNCH_SECRET

This command is used by the guest owner to inject a secret into the guest. This can be done after the launch measurement is retrieved and verified by the guest owner.

6.6.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.LSECRET state.

Before all other processing, the MAC field is verified. The MAC is calculated as follows:

HMAC(0x01 || FLAGS || IV || GUEST_LENGTH || TRANS_LENGTH || DATA || MEASURE; GCTX.TIK)

Where “||” represents concatenation, FLAGS is the full 32-bit FLAGS field, DATA is ciphertext pointed to by TRANS_PADDR, and MEASURE is the value returned by the firmware in the LAUNCH_MEASURE command.

The data pointed to by TRANS_PADDR is decrypted with GCTX.TEK. If FLAGS.COMPRESSED is set to 1, then the resulting plaintext is then decompressed. The result is then written to GUEST_PADDR, encrypted with the guest’s VEK.

The platform remains in the same state after completion. The guest remains in the same state after completion.

6.6.2 Parameters

Table 54 and Table 55 specify the parameters for the LAUNCH_SECRET command.

Table 54. LAUNCH_SECRET Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	HDR_PADDR	System physical address of the region containing the packet header. See Table 55 for the layout.
10h	31:0	In	HDR_LEN	Length of the region containing the packet header in bytes.
14h	31:0	-	-	Reserved. Must be zero.
18h	63:0	In	GUEST_PADDR	System physical of the guest memory region. Must be 16 B aligned.
20h	31:0	In	GUEST_LENGTH	Length of guest memory region. Must be a multiple of 16 B and no more than 16 kB.
24h	31:0	-	-	
28h	63:0	In	TRANS_PADDR	System physical address of the transport memory buffer.
30h	31:0	In	TRANS_LENGTH	Length of the transport memory buffer.

Table 55. LAUNCH_SECRET Packet Header Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	0	In	FLAGS.COMPRESSED	If set, data is compressed.
	31:1	-	-	Reserved. Must be zero.
04h	127:0	In	IV	Initialization vector.
14h	255:0	In	MAC	Integrity protection MAC.

6.6.3 Status Codes

Table 56 enumerates the possible status codes returned by the LAUNCH_SECRET command

Table 56. LAUNCH_SECRET Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.LSECRET state.
INVALID_PARAM	A parameter is invalid.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
BAD_MEASUREMENT	The measurement of the secret is invalid.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.7 LAUNCH_FINISH

After the launch flow is completed, this command is used to transition the guest into a state ready to be run.

6.7.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.LSECRET state.

The following fields of the guest context are zeroed:

- GCTX.TEK
- GCTX.TIK
- GCTX.MS
- GCTX.NONCE
- GCTX.LD

The platform remains in the same state after completion.

The guest transitions to the GSTATE.RUNNING state.

6.7.2 Parameters

Table 57 specifies the parameters for the LAUNCH_FINISH command.

Table 57. LAUNCH_FINISH Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.

6.7.3 Status Codes

Table 58 enumerates the possible status codes returned by the LAUNCH_FINISH command.

Table 58. LAUNCH_FINISH Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.LSECRET state.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.8 ATTESTATON

This command is used to generate an attestation report containing the SHA-256 digest of the guest memory and VMSA passed through the LAUNCH_UPDATE_* commands and signed with the PEK. This digest should match the digest used by the guest owner with LAUNCH_MEASURE. A random MNONCE is provided as input and placed in the report to provide protection.

6.8.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.LSECRET, GSTATE.RUNNING, GSTATE.SUPDATE, or GSTATE.SENT state.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

The LENGTH must be large enough to hold the report or INVALID_LENGTH will be returned and the required length will be placed in the LENGTH field of the Command Buffer.

The firmware will sign the report buffer starting at offset 0h and up to but not including the SIG_USAGE parameter. The value of SIG_USAGE will be PEK, which is defined in Table 113. The value of SIG_ALGO will be ECDSA with SHA-256, which is defined in Table 114. The length of the report will be written to the LENGTH field of the Command Buffer.

6.8.2 Parameters

Table 59 specifies the parameters for the ATTESTATION command.

Table 59. ATTESTATION Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	PADDR	System physical address of the attestation report.
10h	127:0	In	MNONCE	Report nonce.
20h	31:0	In/Out	LENGTH	Input: length of area provided for the attestation report. Output: length of area used by the attestation report.

Table 60. Attestation Report Buffer

Byte Offset	Bits	Name	Description
00h	127:0	MNONCE	Nonce from Command Buffer.
10h	255:0	LAUNCH_DIGEST	SHA-256 digest of launched guest.
30h	31:0	POLICY	Policy the guest was launched with
34h	31:0	SIG_USAGE	Key usage of SIG1 signing key.
38h	31:0	SIG_ALGO	Signature algorithm.
3Ch	31:0	-	Reserved.
40h-CFh		SIG1	Signature.

6.8.3 Status Codes

Table 61 enumerates the possible status codes returned by the ATTESTATION command.

Table 61. ATTESTATION Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.LSECRET, RUNNING, or SUPDATE state.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
INVALID_LENGTH	The length of the buffer for the Attestation report is too small.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.9 SEND_START

This command is used to export a guest from one platform to another. It can be used for saving a guest to disk to be resumed later, or it can be used to migrate a guest across the network to a receiving platform.

6.9.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.RUNNING state.

GCTX.POLICY.NOSEND must be zero. Otherwise, an error is returned.

If GCTX.POLICY.SEV is 1, the PDH, PEK, CEK, ASK, and ARK certificates are validated.

A version check is performed to ensure that the API version implemented on the receiving platform provides the proper functionality requested by the initiator of the guest. The receiving platform API version is read from the PEK certificate. The major API version of the receiving platform must be greater than the guest's POLICY.API_MAJOR field value, or the major API version is equal to POLICY.API_MAJOR, and the minor API version of the receiving platform is greater than or equal to POLICY.API_MINOR. If all of these conditions are not met then an error is returned.

If GCTX.POLICY.SEV is 0, this certificate chain is ignored and no version check is performed.

If GCTX.POLICY.DOMAIN is 1, the PDH-PEK-OCA are validated. If GCTX.POLICY.DOMAIN is 0, this certificate chain is ignored.

When required by the guest policy, the PDH, PEK, OCA, and CEK certificates are validated according to Appendix C on page 118. The ASK and ARK certificates are validated according to Appendix B on page 116.

A fresh nonce is generated and written into the NONCE field. The master secret is calculated with NONCE and PDH_CERT fields and the PCTX.PDH private key according to the key establishment protocol specified in Chapter 2 on page 25. Fresh TEK and TIK transport keys are generated. The transport keys are then wrapped according to Chapter 2 on page 25, and the output is placed in the WRAP_TK, WRAP_IV, and WRAP_MAC fields.

GCTX.POLICY is written to the POLICY field and a MAC of GCTX.POLICY is calculated, keyed with TIK. The MAC is written to the POLICY_MAC field.

The platform remains in the same state after completion.

The guest transitions to the GSTATE.SUPDATE state.

6.9.2 Parameters

Table 62, Table 63, Table 64 and Table 65 specify the parameters for the SEND_START command.

Table 62. SEND_START Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	Out	POLICY	Guest policy
08h	63:0	In	PDH_CERT_PADDR	System physical address of the region containing the PDH certificate. See Appendix C for the layout.

Table 62. SEND_START Command Buffer (Continued)

Byte Offset	Bits	In / Out	Name	Description
10h	31:0	In	PDH_CERT_LEN	Length of the region containing the PDH certificate in bytes.
14h	-	-	-	Reserved. Must be zero.
18h	63:0	In	PLAT_CERTS_PADDR	System physical address of the region containing the platform certificates. See Table 63 for the layout.
20h	31:0	In	PLAT_CERTS_LEN	Length of the region containing the platform certificate in bytes.
24h	-	-	-	Reserved. Must be zero.
28h	63:0	In	AMD_CERTS_PADDR	System physical address of the region containing the AMD certificates. See Table 64 for the layout.
30h	31:0	In	AMD_CERTS_LEN	Length of the region containing the AMD certificates in bytes.
34h	-	-	-	Reserved. Must be zero.
38h	63:0	In	SESSION_PADDR	System physical address of the region containing the session data. Table 65 for the layout.
40h	31:0	In, Out	SESSION_LEN	Length of the region containing the session data in bytes.

Table 63. SEND_START Platform Certificates Buffer

Byte Offset	Bits	In / Out	Name	Description
0000h – 0823h	-	In	PEK_CERT	The PEK certificate. See Appendix C for the layout.
0824h – 1047h	-	In	OCA_CERT	The OCA certificate. See Appendix C for the layout.
1048h – 186Bh	-	In	CEK_CERT	The CEK certificate. See Appendix C for the layout.

Table 64. SEND_START AMD Certificates Buffer

Byte Offset	Bits	In / Out	Name	Description
Variable	-	In	ASK_CERT	The ASK certificate. See Appendix B for the layout.
Variable	-	In	ARK_CERT	The ARK certificate. See Appendix B for the layout.

Table 65. SEND_START Session Data Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	127:0	Out	NONCE	Session nonce.
10h	255:0	Out	WRAP_TK	The TEK and TIK concatenated and encrypted with the KEK.
30h	127:0	Out	WRAP_IV	IV for WRAP_TK encryption.
40h	255:0	Out	WRAP_MAC	HMAC for WRAP_TK using KIK.
60h	255:0	Out	POLICY_MAC	HMAC for POLICY using TIK.

6.9.3 Status Codes

Table 66 enumerates the possible status codes returned by the SEND_START command.

Table 66. SEND_START Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state
INVALID_LENGTH	A buffer length is not correct
INVALID_GUEST_STATE	The guest state is not in the GSTATE.RUNNING state
INVALID_GUEST	The guest handle is invalid
INVALID_CERTIFICATE	A certificate chain is not valid
BAD_SIGNATURE	A signature on the certificates is incorrect
INVALID_ADDRESS	A memory region provided contains invalid physical addresses
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.10 SEND_UPDATE_DATA

This command is used to export guest memory regions to another platform.

6.10.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.SUPDATE state.

If the HDR_LEN is too small, the required length is written out to that field and an error is returned. Otherwise, the number of bytes written to the buffer are written into HDR_LEN.

If FLAGS.COMPRESSED is 1, the data pointed to by GUEST_PADDR will first be compressed before encryption and integrity protection. If FLAGS.COMPRESSED is 0, no compression is performed.

A fresh IV is generated and written to the IV field. The data pointed to by GUEST_PADDR is encrypted with the GCTX.TEK. The result is then written to the TRANS_PADDR field. If the input TRANS_LENGTH field is greater than output length of the compressed and encrypted plaintext, the new length is written into the TRANS_LENGTH field. If TRANS_LENGTH is less than the output length of the compressed and encrypted plaintext, an error is returned.

A MAC is computed to integrity protect the data. The MAC is calculated as follows:

HMAC(0x02 || FLAGS || IV || GUEST_LENGTH || TRANS_LENGTH || DATA;
GCTX.TIK)

Where “||” represents concatenation, FLAGS is the full 32-bit FLAGS field, and DATA is ciphertext pointed to by TRANS_PADDR. TRANS_LENGTH is the new length written out to the TRANS_LENGTH field. The MAC is written out to the MAC field.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.10.2 Parameters

Table 67 specifies the parameters for the SEND_UPDATE_DATA command.

Table 67. SEND_UPDATE_DATA Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	HDR_PADDR	System physical address of the region containing the packet header.
10h	31:0	In, Out	HDR_LEN	Length of the region containing the packet header in bytes.
14h	31:0	-	-	Reserved. Must be zero
18h	63:0	In	GUEST_PADDR	System physical of the guest memory region. Must be 16 B aligned with the C-bit set.

Table 67. SEND_UPDATE_DATA Command Buffer (Continued)

Byte Offset	Bits	In / Out	Name	Description
20h	31:0	In	GUEST_LENGTH	Length of guest memory region. Must be a multiple of 16 B and no more than 16 kB.
24h	31:0	-	-	
28h	63:0	In	TRANS_PADDR	System physical address of the transport memory buffer.
30h	31:0	In, Out	TRANS_LENGTH	Length of the transport memory buffer.

Table 68. SEND_UPDATE_DATA Packet Header Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	0	Out	FLAGS.COMPRESSED	If set, data is compressed.
	31:1	-	-	Reserved. Must be zero.
04h	127:0	Out	IV	Initialization vector.
14h	255:0	Out	MAC	Integrity protection MAC.

6.10.3 Status Codes

Table 69 enumerates the possible status codes returned by the SEND_UPDATE_DATA command.

Table 69. SEND_UPDATE_DATA Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	A buffer length is not correct.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.SUPDATE state.
INVALID_PARAM	A parameter is invalid.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.11 SEND_UPDATE_VMSA

This command is used to export guest VMCB save areas to another platform.

6.11.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.SUPDATE state.

If the HDR_LEN is too small, the required length is written out to that field and an error is returned. Otherwise, the number of bytes written to the buffer are written into HDR_LEN.

If FLAGS.COMPRESSED is 1, the data pointed to by GUEST_PADDR will first be compressed before encryption and integrity protection. If FLAGS.COMPRESSED is 0, no compression is performed.

A fresh IV is generated and written to the IV field. The data pointed to by GUEST_PADDR is encrypted with the GCTX.TEK. The result is then written to the TRANS_PADDR field. If the input TRANS_LENGTH field is greater than output length of the compressed and encrypted plaintext, the new length is written into the TRANS_LENGTH field. If TRANS_LENGTH is less than the output length of the compressed and encrypted plaintext, an error is returned.

A MAC is computed to integrity protect the data. The MAC is calculated as follows:

HMAC(0x03 || FLAGS || IV || GUEST_LENGTH || TRANS_LENGTH || DATA;
GCTX.TIK)

Where “||” represents concatenation, FLAGS is the full 32-bit FLAGS field, and DATA is ciphertext pointed to by TRANS_PADDR. TRANS_LENGTH is the new length written out to the TRANS_LENGTH field. The MAC is written out to the MAC field.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

The vCPU represented by this VMSA should no longer be run by the hypervisor. Future versions of firmware may enforce this constraint.

6.11.2 Parameters

Table 70 and Table 71 specifies the parameters for the SEND_UPDATE_VMSA command.

Table 70. SEND_UPDATE_VMSA Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	HDR_PADDR	System physical address of the region containing the packet header.
10h	31:0	In, Out	HDR_LEN	Length of the region containing the packet header in bytes.
14h	31:0	-	-	Reserved. Must be zero.
18h	63:0	In	GUEST_PADDR	System physical of the guest memory region. Must be 16 B aligned with the C-bit set.
20h	31:0	In	GUEST_LENGTH	Length of guest memory region. Must be a multiple of 16 B and no more than 16 kB.
24h	31:0	-	-	
28h	63:0	In	TRANS_PADDR	System physical address of the transport memory buffer.
30h	31:0	In, Out	TRANS_LENGTH	Length of the transport memory buffer.

Table 71. SEND_UPDATE_VMSA Packet Header Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	0	Out	FLAGS.COMPRESSED	If set, data is compressed.
	31:1	-	-	Reserved. Must be zero.
04h	127:0	Out	IV	Initialization vector.
14h	255:0	Out	MAC	Integrity protection MAC.

6.11.3 Status Codes

Table 72 enumerates the possible status codes returned by the SEND_UPDATE_VMSA command.

Table 72. SEND_UPDATE_VMSA Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_LENGTH	A buffer length is not correct.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.SUPDATE state.
INVALID_PARAM	A parameter is invalid.
INVALID_GUEST	The guest handle is invalid.
UNSUPPORTED	SEV-ES is not configured.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.12 SEND_FINISH

This command finalizes the send operational flow.

6.12.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.SUPDATE state.

The following fields of the guest context are zeroed:

- GCTX.TEK
- GCTX.TIK
- GCTX.MS
- GCTX.NONCE

The platform remains in the same state after completion.

The guest transitions to the GSTATE.SENT state.

6.12.2 Parameters

Table 73 specifies the parameters for the SEND_FINISH command.

Table 73. SEND_FINISH Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.

6.12.3 Status Codes

Table 74 enumerates the possible status codes returned by the SEND_FINISH command.

Table 74. SEND_FINISH Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state
INVALID_GUEST_STATE	The guest state is not in the GSTATE.SUPDATE state
INVALID_GUEST	The guest handle is invalid
INVALID_ADDRESS	A memory region provided contains invalid physical addresses
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.13 SEND_CANCEL

This command cancels the send operational flow.

6.13.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.SUPDATE state.

The following fields of the guest context are zeroed:

- GCTX.TEK
- GCTX.TIK
- GCTX.MS
- GCTX.NONCE

The platform remains in the same state after completion.

The guest transitions to the GSTATE.RUNNING state.

6.13.2 Parameters

Table 75 specifies the parameters for the SEND_CANCEL command.

Table 75. SEND_CANCEL Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.

6.13.3 Status Codes

Table 76 enumerates the possible status codes returned by the SEND_CANCEL command.

Table 76. SEND_CANCEL Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.SUPDATE state.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.14 RECEIVE_START

This command is used to import a guest from one platform to another. It can be used for restoring a guest from disk, or it can be used to migrate a guest across the network from a sending platform.

6.14.1 Actions

The platform must not be in the PSTATE.UNINIT state.

A new guest context is created and assigned to an available guest handle.

The MAC of GCTX.POLICY is calculated, keyed with TIK. The MAC is compared to the POLICY_MAC field to verify the integrity of the guest policy.

If the HANDLE field is zero, a new VEK is generated for this guest. If the HANDLE field is non-zero, the following checks are performed:

- HANDLE is a valid guest
- GUESTS[HANDLE].POLICY is equal to the POLICY field
- The MAC of the POLICY is valid
- POLICY.NOKS is zero

If the above checks pass, then the VEK from the guest referred to by HANDLE is copied into the new guest context.

The handle of the new guest is written to the HANDLE field.

The master secret is calculated with NONCE and PDH_CERT fields and the PCTX.PDH private key according to the key establishment protocol specified in Chapter 2 on page 25. The transport keys in the WRAPPED_TK field are then unwrapped according to Chapter 2 on page 25 using the WRAP_IV and WRAP_MAC fields as the IV and MAC parameters, respectively.

A version check is performed to ensure that the API version implemented on the platform provides the proper functionality requested by the initiator of the guest. The major API version of this platform must be greater than the guest's POLICY.API_MAJOR field value, or the major API version is equal to POLICY.API_MAJOR, and the minor API version of this platform is greater than or equal to POLICY.API_MINOR. If all of these conditions are not met then an error is returned.

If POLICY.ES is set and the platform is configured with SEV-ES, then SEV-ES is enabled. If POLICY.ES is set and the platform is not configured with SEV-ES, an error is returned. Otherwise, SEV-ES is disabled.

The platform remains in the same state after completion.

The guest transitions to the GSTATE.RUPDATE state.

6.14.2 Parameters

Table 77 and Table 78 specify the parameters for the RECEIVE_START command.

Table 77. RECEIVE_START Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In, Out	HANDLE	Guest handle
04h	31:0	In	POLICY	Guest policy
08h	63:0	In	PDH_CERT_PADDR	System physical address of the region containing the PDH certificate.
10h	31:0	In	PDH_CERT_LEN	Length of the region containing the PDH certificate in bytes.
14h	31:0	-	-	Reserved. Must be zero

Byte Offset	Bits	In / Out	Name	Description
18h	63:0	In	SESSION_PADDR	System physical address of the region containing the session data.
20h	31:0	In	SESSION_LEN	Length of the region containing the session data in bytes.

Table 78. RECEIVE_START Session Data Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	127:0	In	NONCE	Session nonce.
10h	255:0	In	WRAP_TK	The TEK and TIK concatenated and encrypted with the KEK.
30h	127:0	In	WRAP_IV	IV for WRAP_TK encryption.
40h	255:0	In	WRAP_MAC	HMAC for WRAP_TK using KIK.
60h	255:0	In	POLICY_MAC	HMAC for POLICY using TIK.

6.14.3 Status Codes

Table 79 enumerates the possible status codes returned by the RECEIVE_START command.

Table 79. RECEIVE_START Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state
INVALID_LENGTH	A buffer length is not correct
INVALID_GUEST	The guest handle is invalid
INVALID_CERTIFICATE	A certificate chain is not valid
POLICY_FAILURE	A guest policy requirement cannot be honored.
BAD_SIGNATURE	A signature on the certificates is incorrect
BAD_MEASUREMENT	The measurement of the wrapped key or policy is invalid
INVALID_ADDRESS	A memory region provided contains invalid physical addresses
UNSUPPORTED	SEV-ES has not been configured
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.15 RECEIVE_UPDATE_DATA

This command is used to import guest memory from a sending platform.

6.15.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.RUPDATE state.

A MAC is computed to verify the integrity of the data area. The MAC is calculated as follows:

```
HMAC(0x02 || FLAGS || IV || GUEST_LENGTH || TRANS_LENGTH || DATA;  
GCTX.TIK)
```

Where “||” represents concatenation, FLAGS is the full 32-bit FLAGS field, and DATA is ciphertext pointed to by TRANS_PADDR. The MAC is compared against the MAC field which must be equal.

The ciphertext data pointed to by TRANS_PADDR is decrypted with the GCTX.TEK and the IV field. If FLAGS.COMPRESSED is 1, the resulting plaintext data will be decompressed. If FLAGS.COMPRESSED is 0, no compression is performed. The result is then encrypted with GCTX.VEK and written to the memory pointed to by GUEST_PADDR field.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.15.2 Parameters

Table 80 and Table 81 specify the parameters for the RECEIVE_UPDATE_DATA command.

Table 80. RECEIVE_UPDATE_DATA Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	HDR_PADDR	System physical address of the region containing the packet header.
10h	31:0	In	HDR_LEN	Length of the region containing the packet header in bytes.
14h	31:0	-	-	Reserved. Must be zero.
18h	63:0	In	GUEST_PADDR	System physical address of the guest memory region. Must be 16 B aligned.
20h	31:0	In	GUEST_LENGTH	Length of guest memory region. Must be a multiple of 16 B and no more than 16 kB.
24h	31:0	-	-	Reserved. Must be zero.
28h	63:0	In	TRANS_PADDR	System physical address of the transport memory buffer.
30h	31:0	In	TRANS_LENGTH	Length of the transport memory buffer.

Table 81. RECEIVE_UPDATE_DATA Packet Header Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	0	In	FLAGS.COMPRESSED	If set, data is compressed.
	31:1	-	-	Reserved. Must be zero.
04h	127:0	In	IV	Initialization vector.
14h	255:0	In	MAC	Integrity protection MAC.

6.15.3 Status Codes

Table 82 enumerates the possible status codes returned by the RECEIVE_UPDATE_DATA command.

Table 82. RECEIVE_UPDATE_DATA Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.RUPDATE state.
INVALID_PARAM	A parameter is invalid.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
BAD_MEASUREMENT	The measurement of the secret is invalid.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.16 RECEIVE_UPDATE_VMSA

This command is used to import a VMCB save area from another platform.

6.16.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.RUPDATE state.

A MAC is computed to verify the integrity of the VMCB save area. The MAC is calculated as follows:

```
HMAC(0x03 || FLAGS || IV || GUEST_LENGTH || TRANS_LENGTH || DATA;
GCTX.TIK)
```

Where “||” represents concatenation, FLAGS is the full 32-bit FLAGS field, and DATA is ciphertext pointed to by TRANS_PADDR. The MAC is compared against the MAC field which must be equal.

The ciphertext data pointed to by TRANS_PADDR is decrypted with the GCTX.TEK and the IV field. If FLAGS.COMPRESSED is 1, the resulting plaintext data will be decompressed. If FLAGS.COMPRESSED is 0, no compression is performed. The result is then encrypted with GCTX.VEK and written to the memory pointed to by GUEST_PADDR field.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.16.2 Parameters

Table 83 and Table 84 specify the parameters for the RECEIVE_UPDATE_VMSA command.

Table 83. RECEIVE_UPDATE_VMSA Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	HDR_PADDR	System physical address of the region containing the packet header.
10h	31:0	In	HDR_LEN	Length of the region containing the packet header in bytes.
14h	31:0	-	-	Reserved. Must be zero.
18h	63:0	In	GUEST_PADDR	System physical of the VMCB save area. Must be 16 B aligned.
20h	31:0	In	GUEST_LENGTH	Length of guest memory region. Must be a multiple of 16 B and no more than 16 kB.
24h	31:0	-	-	Reserved. Must be zero.
28h	63:0	In	TRANS_PADDR	System physical address of the transport memory buffer.
30h	31:0	In	TRANS_LENGTH	Length of the transport memory buffer.

Table 84. RECEIVE_UPDATE_VMSA Packet Header Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	0	In	FLAGS.COMPRESSED	If set, data is compressed.
	31:1	-	-	Reserved. Must be zero.
04h	127:0	In	IV	Initialization vector.
14h	255:0	In	MAC	Integrity protection MAC.

6.16.3 Status Codes

Table 85 enumerates the possible status codes returned by the RECEIVE_UPDATE_VMSA command.

Table 85. RECEIVE_UPDATE_VMSA Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.RUPDATE state.
INVALID_PARAM	A parameter is invalid.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
UNSUPPORTED	SEV-ES is not configured
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
BAD_MEASUREMENT	The measurement of the secret is invalid.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.17 RECEIVE_FINISH

This command is used to complete the receive work flow and bring the guest to a state that can be executed.

6.17.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest must be in the GSTATE.RUPDATE state.

The following fields of the guest context are zeroed:

- GCTX.TEK
- GCTX.TIK
- GCTX.MS
- GCTX.NONCE

The platform remains in the same state after completion.

The guest transitions to the GSTATE.RUNNING state.

6.17.2 Parameters

Table 86 specifies the parameters for the RECEIVE_FINISH command.

Table 86. RECEIVE_FINISH Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.

6.17.3 Status Codes

Table 87 enumerates the possible status codes returned by the RECEIVE_FINISH command.

Table 87. RECEIVE_FINISH Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST_STATE	The guest state is not in the GSTATE.RUPDATE state.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.18 GUEST_STATUS

This command is used to retrieve status information about an SEV-enabled guest.

6.18.1 Actions

The platform must be in the PSTATE.INIT or PSTATE.WORKING states.

The guest may be in any state.

If the HANDLE field is invalid, the STATE field is set to 0h and all other parameters are untouched.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.18.2 Parameters

Table 88 specify the parameters for the GUEST_STATUS command.

Table 88. GUEST_STATUS Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	Out	POLICY	Guest policy.
08h	31:0	Out	ASID	Current ASID. If not active, set to 0h.
0Ch	7:0	Out	STATE	Current guest state. See Table 43.

6.18.3 Status Codes

Table 89 enumerates the possible status codes returned by the GUEST_STATUS command.

Table 89. GUEST_STATUS Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.19 ACTIVATE

This command is used to inform the firmware that the guest is bound to a particular ASID. The firmware then loads the guest's VEK into the memory controller at the key slot for that ASID. When the guest is RUNNING, all Cache Core Complexes (CCXs) will be allowed to execute the guest.

6.19.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest may be in any state.

The guest must be inactive. The ASID parameter must not be zero and must be a valid SEV ASID. The ASID must not be currently used by another guest.

If the guest is SEV-ES enabled, then the ASID must be at least 1h and at most (MIN_SEV_ASID-1). If the guest is not SEV-ES enabled, then the ASID must be at least MIN_SEV_ASID and at most the maximum SEV ASID available. The MIN_SEV_ASID value is discovered by CPUID Fn8000_001F[EDX]. The maximum SEV ASID available is discovered by CPUID Fn8000_001F[ECX].

A successful DF_FLUSH must be executed since the last DEACTIVATE command was issued for the ASID.

The VEK of the guest identified by HANDLE is loaded into the memory controller in the slot identified by ASID. The ASID will be marked valid on all cores when the guest transitions to RUNNING state, or immediately if already in RUNNING state.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.19.2 Parameters

Table 90 specifies the parameters for the ACTIVATE command.

Table 90. ACTIVATE Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	In	ASID	ASID to activate the guest with.

6.19.3 Status Codes

Table 91 enumerates the possible status codes returned by the ACTIVATE command.

Table 91. ACTIVATE Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the working state.
ACTIVE	The guest is currently active.
INVALID_GUEST	The guest handle is invalid.
ASID_OWNED	The ASID is already in use by another guest.
INVALID_ASID	The ASID is invalid.

Return Value	Reason
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
DF_FLUSH_REQUIRED	A DF_FLUSH has not been invoked since the last DEACTIVATE invocation.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.20 ACTIVATE_EX

This command is used to inform the firmware that the guest is bound to a particular ASID and designate which specific threads (identified by APIC ID) will be allowed to run the guest. The firmware then loads the guest's VEK into the memory controller at the key slot for that ASID. When the guest is RUNNING, all threads in the CCX(s) selected by the provided APIC ID(s) will be allowed to execute the guest. That is, designation of a single APIC ID on a CCX will allow all threads in that CCX to execute the guest, and all such threads – whether the guest was run on them or not – must have a WBINVD instruction performed between DEACTIVATE and DF_FLUSH. The benefit of ACTIVATE_EX over ACTIVATE is that by excluding ALL APIC IDs residing on a CCX, that CCX's threads do not need a WBINVD between DEACTIVATE and DF_FLUSH.

6.20.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest may be in any state.

The guest may be active or inactive. The ASID parameter must not be zero and must be a valid SEV ASID. The ASID must not be currently used by a different guest. If the guest is already active, the ASID must be the ASID the guest is using.

If the guest was launched with SEV-ES enabled, then the ASID must be at least 1h and at most (MIN_SEV_ASID-1). If the guest was not SEV-ES enabled, then the ASID must be at least MIN_SEV_ASID and at most the maximum SEV ASID available. The MIN_SEV_ASID value is discovered by CPUID Fn8000_001F[EDX]. The maximum SEV ASID available is discovered by CPUID Fn8000_001F[ECX].

If the guest is inactive, DF_FLUSH (and its pre-requisite WBINVDs) must be executed since INIT, or the last DEACTIVATE command was issued for the ASID, whichever is later.

The VEK of the guest identified by HANDLE is loaded into the memory controller in the slot identified by ASID. The ASID will be marked valid on the selected CCXs when the guest transitions to RUNNING state, or immediately if currently in RUNNING state.

CCXs are identified by the SEV firmware by APIC ID, which is a thread-level identifier. The hypervisor must be aware that ALL threads available on a CCX with one or more activated APIC IDs, are subject to the WBINVD/DF_FLUSH requirements.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.20.2 Parameters

Table 92 specifies the parameters for the ACTIVATE command. If the EX_LEN field is not recognized, the command is invalid.

Table 92. ACTIVATE_EX Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	EX_LEN	Length of the command buffer. 18h for 0.22 of the API
04h	31:0	In	HANDLE	Guest handle.
08h	31:0	In	ASID	ASID to activate the guest with.
0Ch	31:0	In	NUMIDS	Number of APIC IDS in IDS_PADDR list
10h	63:0	In	IDS_PADDR	System physical address of the list of APIC IDs

6.20.3 Status Codes

Table 93 enumerates the possible status codes returned by the ACTIVATE_EX command.

Table 93. ACTIVATE_EX Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the working state.
INVALID_GUEST	The guest handle is invalid.
ASID_OWNED	The ASID is already in use by another guest.
INVALID_ASID	The ASID is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
DF_FLUSH_REQUIRED	A DF_FLUSH has not been invoked since the last DEACTIVATE invocation for the ASID.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.

Return Value	Reason
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.21 DEACTIVATE

This command is used to dissociate the guest from its current ASID. The firmware will uninstall the guest's key from the memory controller and disallow use of the ASID by all CCXs.

6.21.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest may be in any state.

The firmware uninstalls the guest's key from the memory controller and records that a DF_FLUSH is required for this ASID. The ASID is marked invalid on all cores.

The platform remains in the same state after completion.

The guest remains in the same state after completion.

6.21.2 Parameters

Table 94 specifies the parameters for the DEACTIVATE command.

Table 94. DEACTIVATE Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.

6.21.3 Status Codes

Table 95 enumerates the possible status codes returned by the DEACTIVATE command.

Table 95. DEACTIVATE Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_ASID	The ASID is invalid.
INVALID_GUEST	The guest handle is invalid.

Return Value	Reason
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.22 DF_FLUSH

The x86 system software invokes this command after deactivating one or more guests. The x86 system software must execute a WBINVD on the hardware threads that the previous guest was active on before invoking the DF_FLUSH command.

6.22.1 Actions

The platform may be in any state.

Each core must have executed a WBINVD instruction since the last DEACTIVATE command was invoked.

The data fabric write buffers are flushed for each core. The firmware also records that a flush has been performed for all deactivated ASIDs.

The platform remains in the same state after completion.

6.22.2 Parameters

None. The CmdBuf_Lo and CmdBuf_Hi mailbox registers are ignored.

6.22.3 Status Codes

Table 96 enumerates the possible status codes returned by the DF_FLUSH command.

Table 96. DF_FLUSH Command Status Codes

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the PSTAET.INIT or PSTATE.WORKING states
WBINVD_REQUIRED	WBINVD has not been executed
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.23 DECOMMISSION

The hypervisor no longer intends to run the guest. Invoking this command deletes all guest context for this guest. Upon successful completion of this command, HANDLE is no longer a valid guest handle. Guests that shared their VEK with this guest are not affected.

6.23.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest handle must be valid. The guest may be in any state.

The guest must not be active. The hypervisor must invoke DEACTIVATE for the guest before invoking DECOMMISSION.

All guest context is securely deleted from firmware memory. After completion, the guest handle will not be valid.

The platform transitions to the PSTATE.INIT state if PCTX.GUEST_COUNT is zero. Otherwise, the platform remains in the PSTATE.WORKING state.

6.23.2 Parameters

Table 97 specifies the parameters for the DECOMMISSION command.

Table 97. DECOMMISSION Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.

6.23.3 Status Codes

Table 98 enumerates the possible status codes returned by the DECOMMISSION command.

Table 98. DECOMMISSION Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
ACTIVE	The guest is currently active.
INVALID_GUEST	The guest handle is invalid.

Return Value	Reason
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

6.24 COPY

The hypervisor wishes to move encrypted guest page(s) from one place in system physical memory to another place in system physical memory.

6.24.1 Actions

The platform must be in the PSTATE.WORKING state.

The guest handle must be valid. The guest may be in any state. The guest must be active.

LENGTH must be a multiple of 4096. SRC_PADDR must be aligned to a 4096-byte boundary. DST_PADDR must be aligned to a 4096-byte boundary.

LENGTH encrypted bytes are copied from SRC_PADDR to DST_PADDR.

The platform and guest remain in the same state.

6.24.2 Parameters

Table 99 specifies the parameters for the COPY command.

Table 99. COPY Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	In	LENGTH	Number of bytes to be copied. Multiple of 4096.
08h	63:0	In	SRC_PADDR	Physical address of the bytes to be copied.
10h	63:0	In	DST_PADDR	Physical address for the destination of the copy.

6.24.3 Status Codes

Table 100 enumerates the possible status codes returned by the COPY command.

Table 100. COPY Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INACTIVE	The guest is not active.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
INVALID_LENGTH	The LENGTH parameter is not a multiple of 4096, or too large.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.
INVALID_PARAM	A parameter is invalid.

6.25 SWAP_OUT

The hypervisor wishes to move an encrypted guest page from system physical memory to some off-line storage medium (e.g., a disk swap area for the guest.) Note that the page can only be restored as long as the guest is active. This facility cannot be used to save and restore a guest to any other machine or after the guest has been decommissioned.

6.25.1 Actions

The platform must be in the PSTATE.WORKING state. INVALID_PLATFORM_STATE is returned if the platform is in any other state.

SRC_PADDR and DST_PADDR must be aligned to a 4096-byte boundary if PAGE_SIZE indicates a 4 KB page or aligned to a 2097152-byte boundary if PAGE_SIZE indicates a 2 MB page. MDATA_PADDR must be 64-byte aligned. INVALID_ADDRESS is returned if any of these requirements are not met.

If PAGE_TYPE indicates a VMSA page, PAGE_SIZE must be 4 KB or INVALID_PARAM is returned.

SRC_PADDR, DST_PADDR, and MDATA_PADDR must be valid addresses, otherwise INVALID_ADDRESS is returned.

MDATA_PADDR must not overlap SRC_PADDR or DST_PADDR, otherwise INVALID_PARAM is returned.

The guest handle must be valid. INVALID_GUEST is returned if the guest is not valid. The guest may be in any state except UNINIT. The guest must be active. INACTIVE is returned if the guest is not active.

If SRC_PADDR points to a VMSA page and PAGE_TYPE does not equal VMSA or if SRC_PADDR does not point to a VMSA page and PAGE_TYPE equals VMSA, INVALID_PARAM is returned.

The page at SRC_PADDR will be decrypted with the guest’s VEK. A new, random 64-bit IV is generated and stored in the Metadata Entry. The SOFTWARE_DATA, PAGE_SIZE and PAGE_TYPE fields are also stored in the Metadata entry. If the source page was a VMSA page, the VMSA_FLAG is set in the Metadata entry. AES-256 GCM is used to encrypt the decrypted guest memory using the IV with the Metadata entry (with AUTH_TAG zeroed) as additional authenticated data. The generated Auth Tag is stored in the Metadata entry.

The platform and guest remain in the same state.

6.25.2 Parameters

Table 101 specifies the parameters for the SWAP_OUT command.

Table 101. SWAP_OUT Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	0	In	PAGE_SIZE	Indicates page size. 0 indicates a 4 KB page. 1 indicates a 2 MB page.
	2:1	In	PAGE_TYPE	Indicates the page type of the source page. 0h indicates a normal data page. 2h indicates a VMSA page. Other encodings are reserved.
	31:3	-	-	Reserved. Must be zero.
08h	63:0	In	SRC_PADDR	Physical address of the page to be swapped out. Must be 4 KB or 2 MB aligned, per PAGE_SIZE.
10h	63:0	In	DST_PADDR	Physical address for the destination of the copy. Must be 4 KB or 2 MB aligned, per PAGE_SIZE.
18h	63:0	In	MDATA_PADDR	Physical address used to store the page’s METADATA_ENTRY. Must be 64-byte aligned. See Table 102 for format.
20h	63:0	In	SOFTWARE_DATA	64 bits of arbitrary data that will be preserved and authenticated (but not encrypted) by the METADATA_ENTRY.

Table 102. SWAP_OUT & SWAP_IN Metadata Entry

Byte Offset	Bits	Name	Description
00h	63:0	SOFTWARE_DATA	SOFTWARE_DATA from SWAP_OUT command.
08h	63:0	AES_GCM_IV	IV for AES_GCM encryption randomly generated by SWAP_OUT command.
10h	127:0	AUTH_TAG	AES-GCM Authentication Tag generated by SWAP_OUT command.
20h-3Bh	7:0	-	All bytes in this range must be zero.
3Ch	0	PAGE_SIZE	Indicates page size. 0 indicates a 4 KB page. 1 indicates a 2 MB page.
	2:1	PAGE_TYPE	Indicates the page type of the source page. 0h indicates a normal data page. 2h indicates a VMSA page. Other encodings are reserved.
	31:3	-	Reserved. Must be zero.

6.25.3 Status Codes

Table 103 enumerates the possible status codes returned by the SWAP_OUT command.

Table 103. SWAP_OUT Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INACTIVE	The guest is not active.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses or address that are not aligned.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.
INVALID_PARAM	A parameter is invalid.

6.26 SWAP_IN

The hypervisor wishes to restore an encrypted guest page to system physical memory from some off-line storage medium (e.g., a disk swap area for the guest.) Note that the page can only be

restored to the same guest that used SWAP_OUT. This facility cannot be used to save and restore a guest.

6.26.1 Actions

The platform must be in the PSTATE.WORKING state. INVALID_PLATFORM_STATE is returned if the platform is in any other state.

SRC_PADDR and DST_PADDR must be aligned to a 4096-byte boundary if SIZE_FLAG equals a 4 KB page or aligned to a 2097152-byte boundary if SIZE_FLAG equals a 2 M page. MDATA_PADDR must be 64-byte aligned. INVALID_ADDRESS is returned if any of these requirements are not met.

SRC_PADDR, DST_PADDR, and MDATA_PADDR must be valid addresses, otherwise INVALID_ADDRESS is returned.

MDATA_PADDR must not overlap SRC_PADDR or DST_PADDR, otherwise INVALID_PARAM is returned.

PAGE_SIZE and PAGE_SOURCE must match the corresponding values in the metadata entry or else INVALID_PARAM is returned. SWAP_IN_PLACE must be 0 if PAGE_TYPE equals VMSA, otherwise INVALID_PARAM is returned. If SWAP_IN_PLACE is 1, then SRC_PADDR must match DST_PADDR.

The guest handle must be valid. INVALID_GUEST is returned if the guest is not valid. The guest may be in any state except UNINIT. The guest must be active. INACTIVE is returned if the guest is not active.

The page at SRC_PADDR will be decrypted with the guest's OEK and the IV from the Metadata entry. The AUTH_TAG field is saved and zeroed and AES-256 GCM is used to decrypt the encrypted memory to PSP private memory using the Metadata entry as additional authenticated data. If authentication against the original AUTH_TAG value is successful, the decrypted data will be stored at DST_PADDR with the guest's VEK.

The platform and guest remain in the same state.

6.26.2 Parameters

Table 104 specifies the parameters for the SWAP_IN command.

Table 104. SWAP_IN Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	0	In	PAGE_SIZE	Indicates page size. 0 indicates a 4 KB page. 1 indicates a 2 MB page.
	2:1	In	PAGE_TYPE	Indicates the page type of the source page. 0h indicates a normal data page. 2h indicates a VMSA page. Other encodings are reserved.
	3	In	SWAP_IN_PLACE	If set, then SRC_PADDR and DST_PADDR are equal and the page will be swapped in place.
	31:4	-	-	Reserved. Must be zero.
08h	63:0	In	SRC_PADDR	Physical address of the encrypted data to be swapped in. Must be 4 KB aligned.
10h	63:0	In	DST_PADDR	Physical address for the destination of the copy. Must be 4 KB aligned.
18h	63:0	In	MDATA_PADDR	Physical address used to store the page's METADATA_ENTRY. Must be 64-byte aligned. See Table 102 for format.

6.26.3 Status Codes

Table 105 enumerates the possible status codes returned by the SWAP_IN command.

Table 105. SWAP_IN Status Codes

Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INACTIVE	The guest is not active.
INVALID_GUEST	The guest handle is invalid.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

Return Value	Reason
INVALID_PARAM	A parameter is invalid.

Chapter 7 Debugging API

The debugging API provides a simple interface to decrypt and encrypt memory with a guest's VEK. This allows developers of guest kernels and hypervisor to troubleshoot bugs. The commands in this chapter can be (and, in production environments, should be) disabled via the Guest Policy bit NODBG (see Table 2. Guest Policy Structure) established in LAUNCH_START.

7.1 DBG_DECRYPT

This command enables developers of hypervisors and guest kernels to access encrypted memory.

7.1.1 Actions

The guest policy must allow debugging.

The contents of the source region starting at SRC_PADDR and extending LENGTH bytes are decrypted using GCTX.VEK and saved into the destination memory region starting at DST_PADDR and extending LENGTH bytes.

The guest and platform states are unaffected.

7.1.2 Parameters

Table 106 specifies the parameters for the DBG_DECRYPT command.

Table 106. DBG_DECRYPT Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	SRC_PADDR	System physical address of data to decrypt. Must be 16 B aligned.
10h	63:0	In	DST_PADDR	System physical address of destination. Must be 16 B aligned.
18h	31:0	In	LENGTH	Length of regions. Must be 16 B aligned.

7.1.3 Status Codes

Table 107 enumerates the possible status codes returned by the `DBG_DECRYPT` command.

Table 107. `DBG_DECRYPT` Status Codes

Return Value	Reason
<code>SUCCESS</code>	Successful completion.
<code>INVALID_PLATFORM_STATE</code>	The platform is not in the <code>PSTATE.WORKING</code> state.
<code>INVALID_LENGTH</code>	A buffer length is not correct.
<code>INVALID_GUEST</code>	The guest handle is invalid.
<code>INACTIVE</code>	The guest is inactive.
<code>INVALID_ADDRESS</code>	A memory region provided contains invalid physical addresses.
<code>POLICY_FAILURE</code>	The policy disallows debugging.
<code>HWERROR_PLATFORM</code>	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
<code>HWERROR_UNSAFE</code>	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

7.2 `DBG_ENCRYPT`

This command allows a developer to encrypt memory for debugging purposes.

7.2.1 Actions

The guest policy must allow debugging.

The contents of the source region, starting at `SRC_PADDR` and extending `LENGTH` bytes, are encrypted using `GCTX.VEK` and saved into the destination memory region, starting at `DST_PADDR` and extending `LENGTH` bytes.

The guest and platform states are unaffected.

7.2.2 Parameters

Table 108 specifies the parameters for the DBG_ENCRYPT command.

Table 108. DBG_ENCRYPT Command Buffer

Byte Offset	Bits	In / Out	Name	Description
00h	31:0	In	HANDLE	Guest handle.
04h	31:0	-	-	Reserved. Must be zero.
08h	63:0	In	SRC_PADDR	System physical address of data to encrypt. Must be 16 B aligned.
10h	63:0	In	DST_PADDR	System physical address of destination. Must be 16 B aligned.
18h	31:0	In	LENGTH	Length of regions. Must be 16 B aligned.

7.2.3 Status Codes

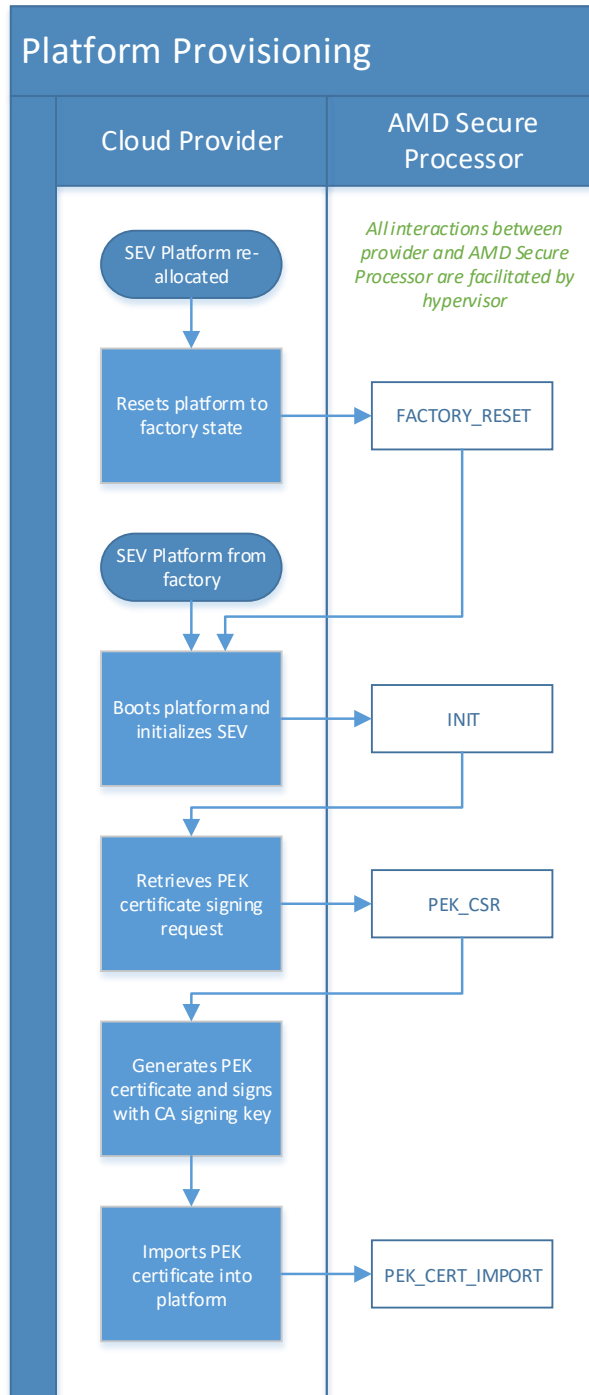
Table 109 enumerates the possible status codes returned by the DBG_ENCRYPT command.

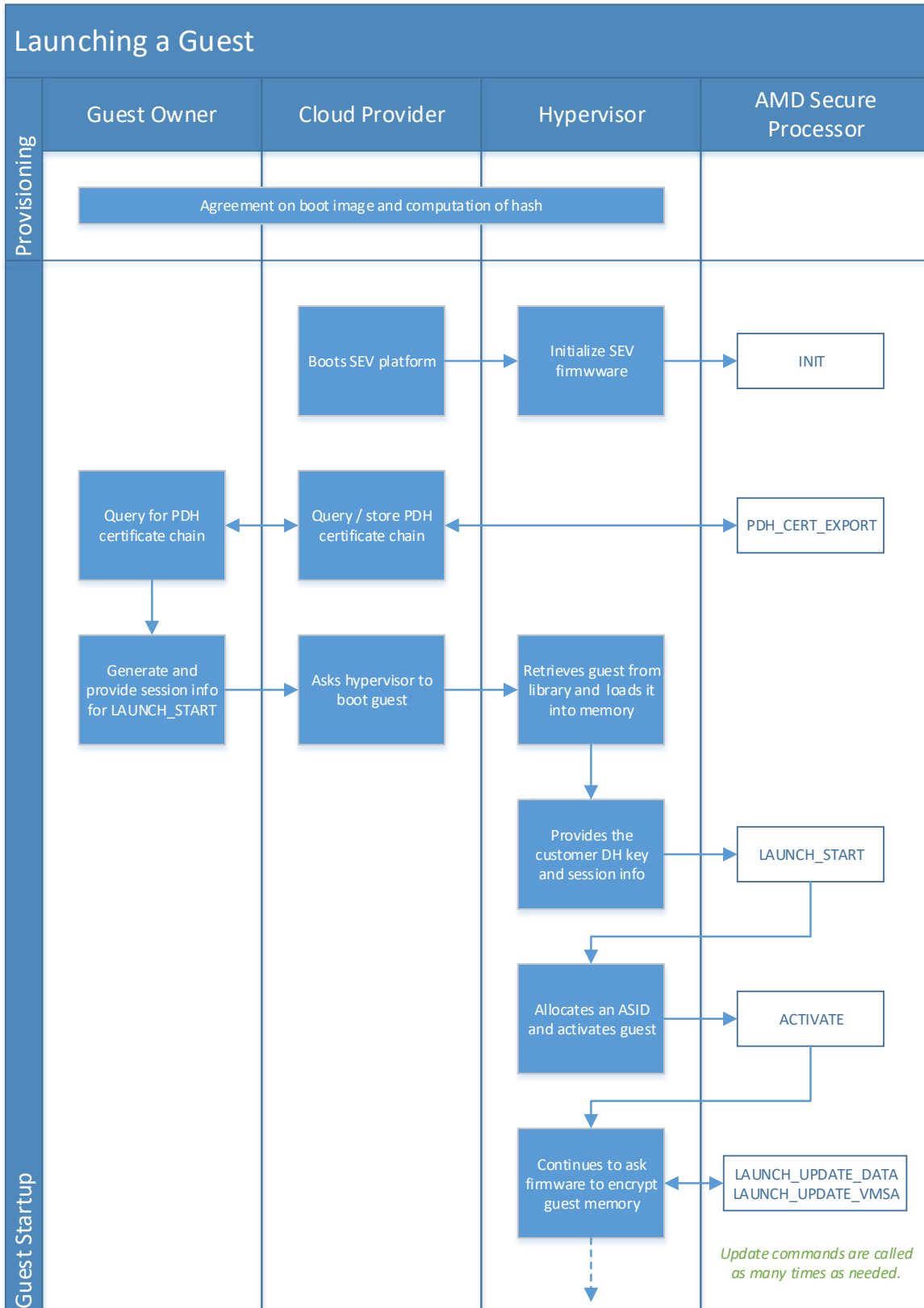
Table 109. DBG_ENCRYPT Status Codes

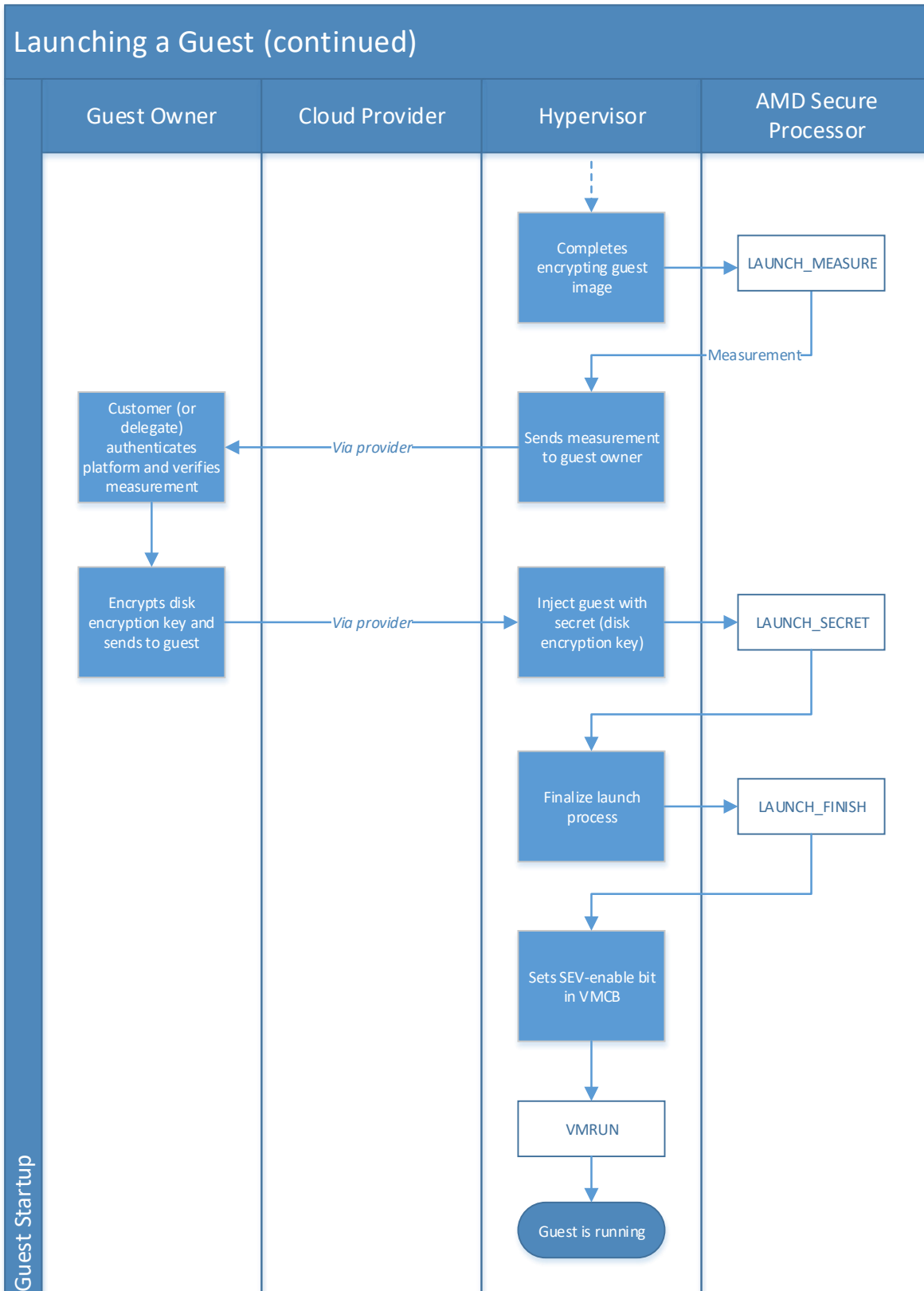
Return Value	Reason
SUCCESS	Successful completion.
INVALID_PLATFORM_STATE	The platform is not in the PSTATE.WORKING state.
INVALID_LENGTH	A buffer length is not correct.
INVALID_GUEST	The guest handle is invalid.
INACTIVE	The guest is inactive.
INVALID_ADDRESS	A memory region provided contains invalid physical addresses.
POLICY_FAILURE	The policy disallows debugging.
HWERROR_PLATFORM	A hardware condition has occurred affecting the platform. It is safe to re-allocate parameter buffers.
HWERROR_UNSAFE	A hardware condition has occurred affecting the platform. It is unsafe to re-allocate parameter buffers.

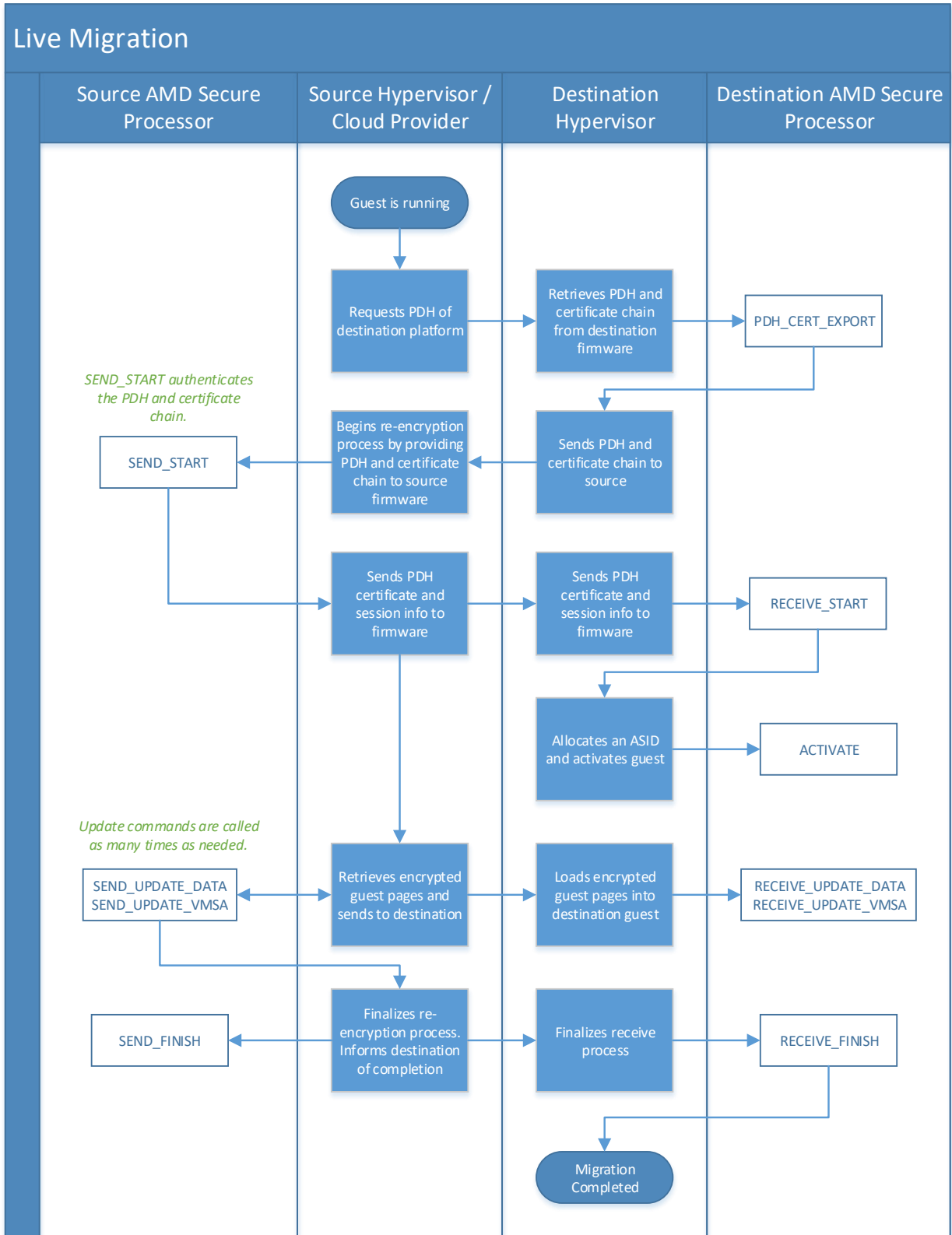
Appendix A Usage Flows

The following flow charts are provided to illustrate the how the usage of the SEV API might be implemented. Note that these are only examples and there may be other implementation strategies.









Appendix B AMD Certificate Authority Certificates

AMD maintains a certificate authority to authenticate AMD hardware and software for a number of applications. The root of trust of the certificate authority is the AMD Root Signing Key (ARK). This key then signs the AMD SEV Signing Key (ASK), which is an intermediate certificate authority specifically used for SEV related authentication.

B.1 Certificate Format

Table 110 specifies the format of the certificates for the ASK and ARK. This version of the API supports version 01h of the certificate format. Table 111 enumerates the valid key usage values for the KEY_USAGE field.

Table 110. AMD Signing Key Certificate Format

Byte Offset	Bits	Name	Description
00h	31:0	VERSION	Certificate format version. The API expects a 1.
04h	127:0	KEY_ID	The unique ID for this key.
14h	127:0	CERTIFYING_ID	The unique ID for the key that signed this certificate. If this certificate is self-signed, then this field equals the KEY_ID field.
24h	31:0	KEY_USAGE	Usage of the key. See Table 111.
28h	127:0	-	Reserved. Set to zero.
38h	31:0	PUBEXP_SIZE	Size of the public exponent in bits. Must be 2048 or 4096.
3Ch	31:0	MODULUS_SIZE	Size of the public modulus in bits. Must be 2048 or 4096.
40h	2047:0 or 4095:0	PUBEXP	Public exponent of this key. The size of this field in bits is equal to PUBEXP_SIZE.
-	2047:0 or 4095:0	MODULUS	Public modulus of this key. The size of this field in bits is equal to MODULUS_SIZE.
-	2047:0 or 4095:0	SIGNATURE	Signature of this key. The size of this field in bits is equal to MODULUS_SIZE in.

Table 111. Key Usage Encoding

Key Usage	Description
00h	AMD Root Signing Key.
13h	AMD SEV Signing Key.

B.2 Certificate Signature

The ARK and ASK certificates both carry RSASSA-PSS signatures. EPYC 7001 generation of microprocessors uses SHA-256 as the digest algorithm with 2048-bit keys. EPYC 7002 generation of microprocessors uses SHA-384 as the digest algorithm with 4096-bit keys.

B.3 Certificate Validation

The following procedure validates an AMD SEV Signing Key certificate:

1. Verify that the ASK VERSION field is a version supported by this API version (1).
2. Verify that the ASK KEY_USAGE field is equal to the ASK key usage encoding.
3. Verify that the ASK PUBEXP_SIZE field is equal to 4096 or 2048.
4. Verify that the ASK MOD_SIZE field is equal to 4096 or 2048.
5. Verify that the ARK VERSION field is a version supported by this API version (1).
6. Verify that the ARK KEY_USAGE field is equal to the ARK key usage encoding.
7. Verify that the ARK PUBEXP_SIZE field is equal to 4096 or 2048.
8. Verify that the ARK MOD_SIZE field is equal to 4096 or 2048.
9. Verify that the ARK KEY_ID field is equal to the ASK CERTIFYING_ID field.
10. Verify that the ASK SIGNATURE field is a valid signature by the ARK.
11. Verify that the ARK SIGNATURE field is a valid signature by the ARK.

If any of the above fail, the certificate is invalid.

Appendix C SEV Certificates

The PEK, CEK, OCA, and PDH key pairs all have certificates used to authenticate the platform. The certificates for the PEK, CEK, OCA, and PDH are distinct from the certificate used within the AMD certificate authority described in Appendix B.

C.1 Certificate Format

An SEV certificate consists of a certificate version, public key, and zero, one, or two signatures. Table 112 specifies the layout of a SEV certificate.

Table 112. SEV Certificate Format

Byte Offset	Bits	Name	Description
000h	31:0	VERSION	Certificate version, set to 01h.
004h	7:0	API_MAJOR	If PEK, set to API minor version Otherwise, zero.
005h	7:0	API_MINOR	If PEK, set to API minor version Otherwise, zero.
006h	7:0	-	Reserved. Set to zero.
007h	7:0	-	Reserved. Set to zero.
008h	31:0	PUBKEY_USAGE	Public key usage
00Ch	31:0	PUBKEY_ALGO	Public key algorithm
010h – 413h	-	PUBKEY	Public key
414h	31:0	SIG1_USAGE	Key usage of SIG1 signing key
418h	31:0	SIG1_ALGO	First signature algorithm
41Ch – 61Bh	-	SIG1	First signature
61Ch	31:0	SIG2_USAGE	Key usage of SIG2 signing key
620h	31:0	SIG2_ALGO	Second signature algorithm
624h – 823h	-	SIG2	Second signature

Table 113. USAGE Enumeration (All other encodings are reserved)

Value	Enumeration
0000h	ARK
0013h	ASK
1000h	<i>Invalid</i>
1001h	OCA
1002h	PEK
1003h	PDH
1004h	CEK

Table 114. ALGO Enumeration (All other encodings are reserved)

Value	Enumeration
0h	Invalid
1h	RSA with SHA-256
2h	ECDSA with SHA-256
3h	ECDH with SHA-256
101h	RSA with SHA-384
102h	ECDSA with SHA-384
103h	ECDH with SHA-384

The PUBKEY field contains the public key represented by this certificate. The format of the PUBKEY field is determined by the PUBKEY_ALGO field. The PUBKEY_ALGO field must be one of the valid algorithms enumerated in Table 114 on page 119. See Section C.3 for the format of the public key for each algorithm.

The PUBKEY_USAGE field specifies the SEV specific usage for the public key represented by this certificate. Table 113 on page 118 enumerates the possible usage identifiers.

The certificate may have zero, one, or two signatures. SIG1_USAGE and SIG2_USAGE describe the key usage for the certifying key that signed the certificate. If SIG1_USAGE is 1000h, SIG1 is not present and the SIG1_ALGO and SIG1 fields are ignored. If SIG2_USAGE is 1000h, SIG2 is not present and the SIG2_ALGO and SIG2 fields are ignored.

The SIG1 and SIG2 fields are formatted according to the SIG1_ALGO and SIG2_ALGO fields, respectively. SIG1_ALGO and SIG2_ALGO must each specify a signing key algorithm from Table 114. See Section C.4, on page 121, for the format of each signature algorithm.

C.2 Elliptic Curve Enumeration

Table 115. CURVE Enumeration (All other encodings are reserved)

Value	Enumeration
0h	Invalid
1h	P256
2h	P384

The above table describes elliptic curves used by the SEV firmware. Note that the NIST P-256 curve is not currently supported but reserved for future usage.

C.3 Public Key Formats

The following subsections describe the public key formats for the SEV certificates.

C.3.1 RSA Public Key

The ALGO.RSA public key format is specified in Table 116. This format supports RSA keys up to 4096 bits in size.

Table 116. RSA Public Key

Byte Offset	Bits	Name	Description
000h	31:0	MODULUS_SIZE	Size of modulus in bits.
004h	4095:0	PUBEXP	Public exponent.
204h	4095:0	MODULUS	Modulus.

C.3.2 ECDSA Public Key

The ALGO.ECDSA public key format is specified in Table 117. The CURVE field specifies which elliptic curve that the public key is defined on. The supported curves are defined in Section C.2 on page 119.

Table 117. ECDSA Public Key

Byte Offset	Bits	Name	Description
000h	31:0	CURVE	Curve ID.
004h	575:0	QX	x component of the public point Q.
04Ch	575:0	QY	y component of the public point Q.
094h – 403h	-	-	Reserved. Must be zero.

C.3.3 ECDH Public Key

The ALGO.ECDH public key format is specified in Table 118. The CURVE field specifies which elliptic curve that the public key is defined on. The supported curves are defined in Section C.2 on page 119.

Table 118. ECDH Public Key

Byte Offset	Bits	Name	Description
000h	31:0	CURVE	Curve ID.
004h	575:0	QX	x component of the public point Q.
04Ch	575:0	QY	y component of the public point Q.
094h – 403h	-	-	Reserved. Must be zero.

C.4 Signature Formats

The following subsections describe the signature formats for the SEV certificates.

C.4.1 RSA Signature

The ALGO.RSA signature format is specified in Table 119. This format supports signatures by RSA keys up to 4096 bits in size.

Table 119. RSA Signature

Byte Offset	Bits	Name	Description
000h – 1FFh	4095:0	S	Signature.

C.4.2 ECDSA Signature

The ALGO.ECDSA signature format is specified in Table 120.

Table 120. ECDSA Signature

Byte Offset	Bits	Name	Description
000h – 047h	575:0	R	R component of the signature.
048h – 08Fh	575:0	S	S component of the signature.
090h – 1FFh	-	-	Reserved. Must be zero.

C.5 Certificate Validation

The following procedure validates an SEV certificate chain:

1. PDH certificate
 - a. Verify that the VERSION field is supported by this API version
 - b. Verify that the PUBKEY_USAGE field is equal to the PDH key usage encoding
 - c. Verify that SIG1_USAGE field is equal to the PEK key usage encoding
 - d. Verify that SIG1_ALGO field is equal to the PUBKEY_ALGO field of the PEK certificate
 - e. Verify that the SIG1 field is a valid signature by the PEK certificate's public key
2. PEK certificate
 - a. Verify that the VERSION field is supported by this API version
 - b. Verify that the PUBKEY_USAGE field is equal to the PEK key usage encoding
 - c. Verify that SIG1_USAGE field is equal to the CEK key usage encoding

- d. Verify that SIG1_ALGO field is equal to the PUBKEY_ALGO field of the CEK certificate
 - e. Verify that the SIG1 field is a valid signature by the CEK certificate's public key
 - f. Verify that SIG2_USAGE field is equal to the OCA key usage encoding
 - g. Verify that SIG2_ALGO field is equal to the PUBKEY_ALGO field of the OCA certificate
 - h. Verify that the SIG2 field is a valid signature by the OCA certificate's public key
 - i. Note: SIG1 and SIG2 can be swapped. That is, SIG1 could be the OCA certificate and SIG2 could be the CEK certificate.
3. OCA certificate
 - a. Verify that the VERSION field is supported by this API version
 - b. Verify that the PUBKEY_USAGE field is equal to the OCA key usage encoding
 - c. Verify that SIG1_USAGE field is equal to the PEK key usage encoding
 - d. Verify that SIG1_ALGO field is equal to the PUBKEY_ALGO field
 - e. Verify that the SIG1 field is a valid self signature
 4. CEK certificate
 - a. Verify that the VERSION field is supported by this API version
 - b. Verify that the PUBKEY_USAGE field is equal to the CEK key usage encoding
 - c. Verify that SIG1_USAGE field is equal to the ASK key usage encoding
 - d. Verify that SIG1_ALGO field is equal to the PUBKEY_ALGO field of the ASK certificate
 - e. Verify that the SIG1 field is a valid signature by the ASK certificate's public key
 5. ASK certificate – see Section B
 6. ARK certificate – see Section B

If any of the above fail, the certificate is invalid