Secure Encrypted Virtualization Key Management

Technical Preview

Publication # **55766** Revision: **3.00**
Issue Date: **May 2016**

Advanced Micro Devices

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.

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

List of Figures

List of Tables

Revision History

Date	Revision	Description
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 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 is handled by the AMD Secure Processor firmware which exposes an API for these tasks. The key management API 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

1.2.1 Platform Lifecycle

The firmware maintains a state for the platform. To initialize the platform, the INIT command is executed to initialize and configure the platform state. 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 FACTORY_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 signed by the root AMD signing key. A guest owner or remote platform interacting with the firmware can obtain the public component of the AMD signing key to prove the hardware authenticity. The mechanism to obtain the public key is outside the scope of this document.

The chip-unique signing key signs the PEK. This signature connects the PEK to the AMD signing key, 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.

1.3 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: LAUNCH_START, LAUNCH_UPDATE, and LAUNCH_FINISH. These three 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 below).

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

1.3.2 Activate and Deactivate

After the guest has been launched and SEV has been enabled, the guest is ready to be executed. AMD-V requires the guest to be associated with an ASID within its VMCB. 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 keys that the firmware supports is unlimited. However, there are only a limited number of ASIDs that can be associated with keys. The firmware provides to the hypervisor interfaces to manage this association: ACTIVATE, DEACTIVATE, and DF_FLUSH.

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. Once this flushing instruction completes, the hypervisor calls DF_FLUSH to ensure that the data fabric is completely flushed. This command checks that a WBINVD has been executed since the last DEACTIVATE. These steps are necessary 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 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. 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, and SEND_FINISH for the sending or snapshotting platform and RECEIVE_START, RECEIVE_UPDATE, and RECEIVE_FINISH for the receiving or reconstituting platform.

The SEND_START, SEND_UPDATE, and SEND_FINISH commands prepare the guest's memory image for storage or transmission. SEND_START re-encrypted the image within a cryptographic context that is established with the remote platform. Subsequent invocations of SEND_UPDATE may be used to encrypt additional memory regions. When the guest is fully re-encrypted, the SEND_FINISH completes and produces an integrity measurement of the pages the firmware re-encrypted. The integrity measurement will be verified by the receiving platform.

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, 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 command re-encrypts memory regions provided by the hypervisor. The RECEIVE_FINISH finalizes 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 shutdown 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-256	Key agreement
Platform Endorsement Key	PEK	ECDSA curve P-256	Platform authentication and ownership. Signed by CA and CEK.
Chip Endorsement Key	CEK	ECDSA curve P-256	Platform authentication. Signed by ASK
AMD Signing Key	ASK	ECDSA curve P-256	Root of trust for platform authenticity.
Certificate Authority Signing Key	CA	Various	Root of trust for platform ownership
Transport Integrity Key	TIK	HMAC SHA-256	Integrity of transported guest
Transport Encryption Key	TEK	AES 128	Encryption of transported guest
Launch Measurement Key	LMK	HMAC SHA-256	Measurement of guest launch
VM Encryption Key	VEK	AES 128	Guest memory encryption

2.1 Keys

2.1.1 Platform Diffie-Hellman Key

The Platform Diffie-Hellman key (PDH) is an Elliptic Curve Diffie Hellman (ECDH) key using curve P-256 as defined in section D.1.2.3 of [FIPS 186-4] that is unique to each platform. It is generated on system reset or when the PDH_GEN command is invoked. The PDH is stored in private volatile memory accessible only by the AMD Secure Processor firmware. The PDH is used to perform key agreement with remote parties such as guest owners and other instances of the SEV firmware.

The PDH public key is exported via the PDH_CERT_EXPORT command.

2.1.2 Platform Endorsement Key

The Platform Endorsement Key (PEK) is an ECDSA signing key using curve P-256 as defined in section D.1.2.3 of [FIPS 186-4] that is unique to each platform. It is generated when the SEV firmware is provisioned and stored in non-volatile storage available only to the firmware. It will be regenerated when either the INIT command or PEK_GEN command are successfully invoked.

The PEK signs the PDH of its platform. This asserts that the PDH is owned by the hardware that possesses the PEK private key.

The PEK certificates are exported via the PDH_CERT_EXPORT command.

2.1.3 Chip Endorsement Key

The Chip Endorsement Key (CEK) is an ECDSA signing key using curve P-256 as defined in section D.1.2.3 of [FIPS 186-4] that is unique to each platform. It exists for the lifetime of the platform and is stored within the hardware of the AMD Secure Processor. The CEK is signed by the ASK (see below). This signature asserts that the hardware that possesses the CEK private key is authentic AMD hardware and capable of executing SEV enabled guests.

The CEK certificate is exported in the PDH_CERT_EXPORT command.

2.1.4 AMD Signing Key

The AMD Signing Key (ASK) is an ECDSA signing key using curve P-256 as defined in section D.1.2.3 of [FIPS 186-4] that is owned by AMD. It is stored in and protected by an internal key server operated by AMD. The ASK is used as the root of trust that asserts the authenticity of platforms by signing the CEK of every platform.

The ASK certificate and the CEK certificates for each platform are publicly accessible through the AMD certificate authority.

2.1.5 Certificate Authority Signing Key

The platform owner’s Certificate Authority signing key (CA) is protected by the platform owner’s policies and security controls. The platform owner uses the CA to create and sign a certificate for the PEK. It loads the PEK into the firmware via the PEK_CERT_IMPORT command. This certificate asserts ownership of the platform by the platform owner.

The firmware supports RSA, DSA, and ECDSA algorithms.

2.1.6 Transport Integrity Key

The Transport Integrity Key (TIK) is a 256-bit HMAC SHA-256 integrity key. The TIK protects the integrity of the transport session between two firmware instances during image migration or snapshotting. The TIK is agreed upon during a Diffie-Hellman key agreement protocol and is unique to the session.

2.1.7 Transport Encryption Key

The Transport Encryption Key (TEK) is a 128-bit AES-128 encryption key. The TEK protects the confidentiality of the transport session between two firmware instances during image migration or snapshotting. The TEK is agreed upon during a Diffie-Hellman key agreement protocol and is unique to the session.

2.1.8 Launch Measurement Key

The Launch Measurement Key (LMK) is a 256-bit HMAC SHA-256 integrity key. The firmware uses the LMK to measure the launch process and demonstrate that the guest was encrypted without interference. The LMK is agreed upon during a Diffie-Hellman key agreement protocol and is unique to the launch session.

2.1.9 VM Encryption Key

The VM Encryption Key (VEK) is a 128-bit AES-128 key used to encrypt memory of the guest. The VEK is generated by using the LAUNCH_START and RECEIVE_START commands. The lifetime of the VEK lasts until the guest is decommissioned, after using the DECOMMISSION command. This key is not exportable outside of the firmware.

2.2 Usage

2.2.1 Digital Signatures

Three signatures are verified by the firmware: The PEK signature of the PDH of remote platforms, the CEK signature of the PEK, and the CA signatures of the PEK. These verifications occur on invocation of the SEND_START command. The supported signature algorithms are the ones described in [RFC 3279] (excluding MD2 and MD5), [RFC 5480], and [RFC 5758]. The firmware is also capable of verifying ECDSA signatures using SHA-256 and curve P-256 as defined in [FIPS 186-4].

The firmware generates two signatures; it signs the PDH with the PEK, and it signs the PEK with the CEK. The firmware uses the ECDSA signature algorithm with SHA-256 and curve P-256 as defined in [FIPS 186-4].

2.2.2 Key Agreement

A master secret is derived between the firmware and a remote party to establish a trusted channel.

The master secret agreement is accomplished with the Elliptic Curve Diffie-Hellman key agreement algorithm as defined in [SP 800-56A]. The elliptic curve used is P-256 as defined in section D.1.2.3 of [FIPS 186-4]. In both cases, static Diffie-Hellman keys are used as described in section 6.3 of [SP 800-56A]. This key agreement process involves transmitting fresh nonce between the parties to agree on a key.

To produce keys for secure communication based on the master secret, the firmware uses the counter mode KDF defined in section 5.1 of [SP 800-108] using HMAC-SHA-256 as the pseudorandom function. The KDF is supplied the nonce for context. The labels used are ASCII-encoded null-terminated strings.

2.2.3 Launch Measurements

The launch measurement uses an HMAC-SHA-256 keyed with the LMK established with the guest owner. HMAC-SHA-256 is defined in [FIPS 198-1].

The launch measurement key is derived from the master secret that is established between the firmware and the guest owner. The derivation function is specified in Section 2.2.2 on page 21. The label used for the launch measurement key is the ASCII-encoded null-terminated string “sev-launch-measurement-key”.

2.2.4 Key Wrapping

To establish the transport keys, the firmware and the remote party use a key encryption key to wrap the transport encryption key and the transport integrity key during transmission to the remote entity. It also uses a key encryption key to unwrap keys sent to it in the same manner. The key wrapping algorithm is specified in [SP 800-38F], and the firmware uses the AES-based algorithm.

The key encryption key is an AES 128 key derived from the master secret that is established between the firmware and the remote entity sending a guest memory image. The derivation function is specified in Section 2.2.2 on page 21. The label used for the key encryption key is the ASCII-encoded null-terminated string “sev-key-encryption-key”.

2.2.5 Transport Protection

The firmware protects the confidentiality and integrity of a guest during migration or snapshotting. The guest memory image and SEV metadata is encrypted with the TEK and integrity protected with the TIK. The firmware encrypts decrypts guests using AES-128 using the CTR block cipher mode. The firmware provides a message authentication code using HMAC SHA-256 keyed with the TIK.

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. For instance, the POLICY.DBG flag disallows debugging commands for the guest. The POLICY.SEV requires that the guest only be transmitted to platforms that are only SEV capable platforms. 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	DBG	Debugging of the guest is disallowed when set
	1	KS	Sharing keys with other guests is disallowed when set
	2	Reserved. Must be one.	
	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	FW_MAJOR	The guest must not be transmitted to another platform with a lower firmware version.
003h	7:0	FW_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.KS.

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 in order to invoke the SEV key management functions described in this API.

In a nutshell, the driver writes the 7-bit command 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 Mailboxes

The driver communicates with the SEV firmware via three 32-bit MMIO registers and a command buffer in DRAM that contains additional parameters. The first register, CmdResp, has the following layout:

Table 3. CmdResp Layout

Bit#	Description/Purpose
31	0: Command; 1: Response
[30:24]	Reserved
[23:16]	Command ID
[15:0]	Error code; valid when Response (bit #31) is set to 1

The other two registers, CmdBufAddr_Hi and CmdBufAddr_Lo are the high and low bits, respectively, of a 64-bit DRAM system physical address. This address points to the command buffer which contains additional parameters.

4.2 Command Buffer

When the command takes no parameters, CmdBufAddr_Hi and CmdBufAddr_Lo are ignored by firmware. For commands that take one or more parameters, the command buffer has the following format:

Table 4. Command Buffer Layout

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	In: Number of bytes allocated to command buffer Out: The number of bytes used by the command
CBUF_LEN	Command specific.		

For every command that takes parameters in the command buffer, the firmware checks that there is enough memory allocated to complete the necessary operations for this command. There are several commands, for instance, that output data whose length are not determined until the command is invoked, such as exporting X.509 certificates.

If the firmware does not have enough memory allocated in the command buffer, no action is taken and the amount of bytes needed for this command is written to CBUF_LEN. The firmware then exits with error code CBUF_TOO_SMALL.

If the firmware does have enough memory allocated in the command buffer for the command, after the command completes successfully, the firmware writes the number of bytes of the buffer that were used into the CBUF_LEN field, and returns a success code.

Note: The above CBUF_LEN parameter is reflected in all of the command descriptions below for clarity.

4.3 Command Identifiers

Table 5 summarizes the platform management, guest management, and debugging commands. See the command definitions for further details.

Table 5. Command Identifiers

Command	ID	Description
Error! Reference ource not found.	0x01	Initialize the platform
SHUTDOWN	0x07	Shut down the platform
FACTORY_RESET	0x08	Delete the persistent platform state
PLATFORM_STATUS	0x09	Return status of the platform
PEK_GEN	0x0A	Generate a new PEK
PEK_CSR	0x0B	Generate a PEK certificate signing request
PEK_CERT_IMPORT	0x0C	Import the signed PEK certificate
PDH_GEN	0x0D	Generate a new PDH and PEK signature
PDH_CERT_EXPORT	0x0E	Export the PDH and its certificate chains
LAUNCH_START	0x02	Begin to launch a new SEV enabled guest
LAUNCH_UPDATE	0x03	Encrypt guest memory for launch
LAUNCH_FINISH	0x04	Complete launch of guest
SEND_START	0x0F	Begin to send guest to new remote platform
SEND_UPDATE	0x10	Re-encrypt guest memory for transmission
SEND_FIINISH	0x11	Complete sending guest to remote platform
RECEIVE_START	0x12	Begin to receive guest from remote platform

RECEIVE_UPDATE	0x13	Re-encrypt guest memory from transmission

Table 5. Command Identifiers (Continued)

Command	ID	Description
RECEIVE_FINISH	0x14	Complete receiving guest from remote platform
GUEST_STATUS	0x15	Query the status and metadata of a guest
ACTIVATE	0x05	Load a guest's key into the memory controller
DEACTIVATE	0x16	Unload a guest's key into the memory controller
DF_FLUSH	0x06	Flush the data fabric
DECOMMISSION	0x17	Delete the guest's SEV context in firmware
DBG_DECRYPT	0x18	Decrypt guest memory region for debugging
DBG_ENCRYPT	0x19	Encrypt guest memory region for debugging

4.4 Status Codes

Table 6 summarizes the status codes that can be returned by the firmware commands. See the command definitions for further details.

Table 6. 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
CMDBUF_TOO_SMALL	0x0004	The command 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
DFFLUSH_REQUIRED	0x000F	DF_FLUSH invocation required
INVALID_GUEST	0x0010	The guest handle is invalid

4.5 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 elliptic curve integral values too, such as the QX and QY components of a curve point and the r and s components of the ECDSA signature.

Chapter 5 Platform Management API

The firmware maintains the state of the SEV platform. Valid platform states are as follows:

- Uninitialized
- Initialized
- Working

The uninitialized state is the default state the firmware enters when the system is initially booted. When the platform is in the initialized state, it is able to provide support for guest management commands. The working state indicates that one or more guests are being managed by the firmware.

Commands related to altering the identity of the platform are restricted to the initialized state. This requirement is due to the fact that regenerating identities while those guests are active (Working state) within the system may potentially violate the security invariants expected by the guests.

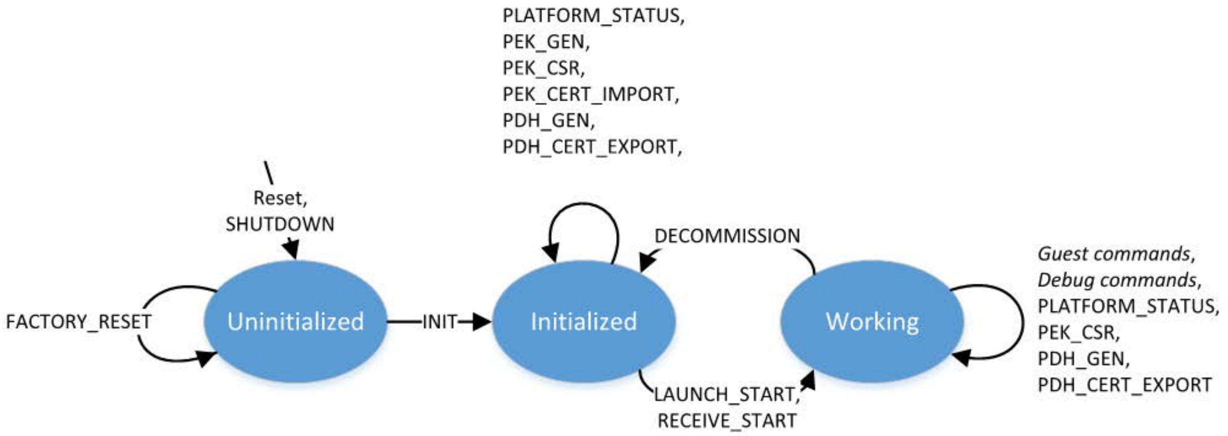


Figure 1. Platform Management State Machine

See Figure 1 for an illustration of the state machine. The LAUNCH_START, RECEIVE_START, and DECOMMISSION commands are part of the guest management interface. If an edge does not exist in a state for a command, no action is taken and an error is returned to the caller.

The platform is in the working state when it is maintains state for guests. The first LAUNCH_START or RECEIVE_START commands transition the platform into the working state. When the last guest is decommissioned with the DECOMMISSION command, the platform transitions back to the initialized state.

The firmware manages platform-level data. These include:

- The platform Diffie-Hellman key (PDH) used to establish keys with external parties
- The platform endorsement key (PEK) that signs the PDH
- The chip endorsement key (CEK) that signs the PEK to prove authenticity of the part
- The certificate authority (CA) and a certificate chain linking the PEK to the CA root of trust
- The API version
- The platform serial number

The owner of the CA private key is considered to be the owner of the platform. The platform can generate its own CA key pair, which defines the platform as self-owned. If the platform imports an external CA certificate, the platform is then defined to be owned by that CA.

The firmware, via specific policy, controls whether guests are allowed to be migrated to platforms with different owners. This depends on the migration policy of both the guest and the platform.

5.1 INIT

Initialize the platform.

5.1.1 Usage

This command is used to transition the platform to the initialized state. The INIT command must be called before any other platform management commands are called (except FACTORY_RESET). The firmware will only accept the INIT command from the uninitialized state.

5.1.2 Actions

The firmware first loads the persistent state into its private memory, and then performs the following actions:

- The Chip Endorsement Key (CEK) is derived from the chip unique values.
- If no CA certificate exists, a CA signing key is generated and a self-signed CA X.509 certificate is created. The signing key and certificate are written to persistent memory.
- If no PEK exists or the CA was just regenerated, a PEK signing key is generated and a PEK X.509 certificate is created and signed by the CA. The PEK and its certificate are written to persistent memory.
- A new PDH key is generated. Its public key and platform metadata are signed by the PEK and the CEK.
- All SEV-related ASIDs on all cores are marked invalid. See ACTIVATE and DEACTIVATE for further details.

The platform must be in the uninitialized state. Otherwise an error is returned.

5.1.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	FLAGS	Flags describing various status; Reserved for future use. Must be zero.

5.1.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the uninitialized state
INVALID_CONFIG	The configuration is invalid or unsupported
CMDBUF_TOO_SMALL	The command buffer is too small
INVALID_ADDRESS	An address is invalid

5.2 SHUTDOWN

Securely de-initialize the platform.

5.2.1 Usage

This command is used to transition the platform to the uninitialized state.

5.2.2 Actions

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

This command is valid when the platform is in any state. After completion, the platform transitions into the uninitialized state.

5.2.3 Parameters

None.

5.2.4 Return Values

Return Value	Reason
SUCCESS	Successful completion

5.3 FACTORY_RESET

Reset the persistent state information of the platform.

5.3.1 Usage

This command is invoked when the platform is:

- decommissioned
- transferred to a new owner
- re-allocated to another domain

5.3.2 Actions

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

This platform must be in the uninitialized state. Otherwise, a SHUTDOWN command must first be invoked to transition the platform into the uninitialized state.

5.3.3 Parameters

None.

5.3.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the uninitialized state

5.4 PLATFORM_STATUS

Returns the current status of the platform.

5.4.1 Usage

Used to collect the current state of the platform.

5.4.2 Actions

If the platform is in the uninitialized state, then FLAGS, CERT_STATUS, and GUEST_COUNT values are not written out.

The CERT_STATUS[0] bit is set to 1 if the platform is owned by a domain. The firmware checks domain ownership by determining if the CA certificate that signed its PEK was generated by the platform or whether the CA certification was imported by the domain owner.

The CERT_STATUS[1] bit is set to 1 if the certificate chain is valid. The firmware checks that the signatures on the certificates are valid and that the validity dates on the certificate are valid. The firmware also reports an invalid certificate chain if one of the certificates is known to the firmware to have been revoked.

The FLAGS parameter is equal to the value passed to the INIT command.

5.4.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
1	Out	API_MAJOR	Major API version
1	Out	API_MINOR	Minor API version.
1	Out	STATE	Current platform state: 0x00: Uninitialized 0x01: Initialized 0x02: Working All other encodings are reserved.
1	Out	CERT_STATUS	Bit field describing certificate status: Bit 0: Domain owned Bit 1: Valid certificate chain Bits 7:2 are reserved.
4	Out	FLAGS	Flags describing configuration. All bits reserved.
4	Out	GUEST_COUNT	Number of valid guests maintained by the firmware

5.4.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
CMDBUF_TOO_SMALL	The command buffer is too small

5.5 PEK_GEN

Generates a new PEK signing key and certificate.

5.5.1 Usage

This command is used to generate a new PEK. This should be invoked only once during the provisioning of the platform.

5.5.2 Actions

Various keys and certificates are securely deleted from volatile and non-volatile storage including the current certificate chain, CA signing key (if it exists), and the PEK. The current PDH is also deleted from volatile storage.

A new CA signing key and self-signed certificate are generated and stored in non-volatile storage. The CA is an ECDSA key pair using the NIST curve P-256.

A new PEK signing key and certificate signed by the new CA are generated and stored in non-volatile storage. The PEK is an ECDSA key pair using curve P-256.

This call automatically invokes a regeneration of the PDH. See PDH_GEN section for regeneration details.

If a CSR has been generated for the previous PEK, it is deleted. See PEK_CSR for more details.

The platform must be in the initialized state.

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

5.5.3 Parameters

None.

5.5.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the initialized state

5.6 PEK_CSR

Generates a certificate signing request (CSR) for the PEK signing key.

5.6.1 Usage

During the provisioning process, the platform can be joined to a domain 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 an X.509 certificate with the information from the CSR and signing it with its key.

5.6.2 Actions

A CSR is generated in the format of PKCS #10. The subject of the CSR has the common name "SEV-PEK-"concatenated with the serial number of the platform character encoded. The subject also has a serial number attribute which is set to the serial number of the platform. The CSR contains the PEK public key and is signed by the PEK private key.

Each CSR produced is stored in volatile storage. Subsequent invocations of this command return the same CSR until a new PEK is generated or until the platform transitions to the uninitialized state.

The platform must be in the initialized or working states.

5.6.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
Variable	Out	CSR	Certificate signing request formatted with PKCS #10

5.6.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the initialized or working states
CMDBUF_TOO_SMALL	The command buffer is too small

5.7 PEK_CERT_IMPORT

Import the X.509 certificate of the PEK and its certificate chain.

5.7.1 Usage

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 an X.509 certificate with the information from the CSR and signing it with its key. This command imports the X.509 certificate of the PEK and its X.509 certificate chain 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.

5.7.2 Actions

The certificate chain consists of one or more certificates, where the first certificate is signed the PEK and the last certificate provided is the root CA certificate. The platform validates the certificate chain to ensure that each signature is valid and that each certificate has not expired. The certificates must be X.509 certificates in the DER format.

The PEK X.509 certificate is checked to ensure that it matches the CSR exported via the PEK_CSR command. Also, the firmware checks that the public key in the PEK certificate matches its PEK public key.

The platform must be self-owned or an error is returned. A successful call to this command transitions the platform from a self-owned platform to a domain-owned platform, where the owner is the owner of the root CA signing key. The current CA signing key, CA certificate, and PEK certificate are overwritten in volatile and non-volatile storage.

This call automatically invokes a regeneration of the PDH. See Section 5.8, PDH_GEN, on page 37.

The platform must be in the initialized state.

5.7.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	N	Number of certificates in the certificate chain (excluding the PEK certificate).
Variable	In	PEK_CERT	PEK certificate signed by the domain CA. Signed by CERT1.
Variable	In	CERT1	First certificate in the chain. Signed by CERT2.
Variable	In	CERT2	Second certificate in the chain. Signed by CERT3.
…			
Variable	In	CERTn	nth certificate in the chain and the root CA

5.7.4 Return Values

Return Value	Reason
SUCCESS	Successful completion

INVALID_CONFIG	The configuration is invalid or unsupported
CMDBUF_TOO_SMALL	The command buffer is too small
ALREADY_OWNED	The platform is already owned.
INVALID_CERTIFICATE	The certificate is improperly formatted. The certificate’s public key does not match the PEK.

5.8 PDH_GEN

Regenerates the PDH.

5.8.1 Usage

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.

5.8.2 Actions

Generates a new PDH key pair. The PDH is an ECDH key agreement key with NIST curve P-256. After generation of the PDH key, the firmware then signs the PDH public key with the CEK and the PEK along with other metadata. Specifically, this operation is a signature of the following values (where `||’ is the concatenation operator)

PDH_PUB_QX || PDH_PUB_QY || API_MAJOR || API_MINOR || SERIAL

where PDH_PUB_QX and PDH_PUB_QY are the public component of the PDH, the API_MAJOR and API_MINOR are the major and minor API versions of this platform, and SERIAL is the serial number of this platform.

The platform must be in the initialized or working states or an error is returned. The PDH is never stored in non-volatile or other persistent storage and is regenerated after each call to INIT, PEK_GEN, and PEK_IMPORT_CERT.

5.8.3 Parameters

None.

5.8.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform must be in the initialized state.

5.9 PDH_CERT_EXPORT

Exports the PDH public key and all the signatures and certificates necessary to authenticate the PDH.

5.9.1 Usage

This command is used to 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.

5.9.2 Actions

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

- The PDH public key and its metadata
- The CEK signature of the PDH public key and its metadata
- The PEK signature of the PDH public key and its metadata
- The PEK certificate chain

The CEK and PEK signatures exported by this command are generated by the PDH_GEN command.

The platform must be in the initialized or working state.

5.9.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
1	Out	API_MAJOR	API major version
1	Out	API_MINOR	API minor version
2	Reserved.		
4	Out	SERIAL	Platform serial number
32	Out	PDH_PUB_QX	The Qx parameter of the PDH public key
32	Out	PDH_PUB_QY	The Qy parameter of the PDH public key
32	Out	PEK_SIG_R	The r component of the PEK signature
32	Out	PEK_SIG_S	The s component of the PEK signature
32	Out	CEK_SIG_R	The r component of the CEK signature
32	Out	CEK_SIG_S	The s component of the CEK signature
32	Out	CEK_PUB_QX	The Qx parameter of the CEK public key
32	Out	CEK_PUB_QY	The Qy parameter of the CEK public key
4	Out	N	Number of certificates in the certificate chain (excluding the PEK certificate
Variable	Out	PEK_CERT	PEK certificate signed by the domain CA. Signed by CERT1.
Variable	Out	CERT1	First certificate in the chain. Signed by CERT2.
Variable	Out	CERT2	Second certificate in the chain. Signed by CERT3.
…			
Variable	Out	CERTn	nth certificate in the chain and the root CA

5.9.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the initialized or working states
CMDBUF_TOO_SMALL	The command buffer is too small

Chapter 6 Guest Management API

The firmware maintains state of each guest. Valid guest states are as follows:

- Invalid
- Launching
- Running
- Sending
- Receiving

A guest can also be in the off, booted, or sent states as shown below, but these states are not tracked by the firmware; in these cases, the firmware reports invalid for guest state status requests via GUEST_STATUS. A guest is identified by its guest handle, which is provided by the firmware to the hypervisor through which it manages the guest.

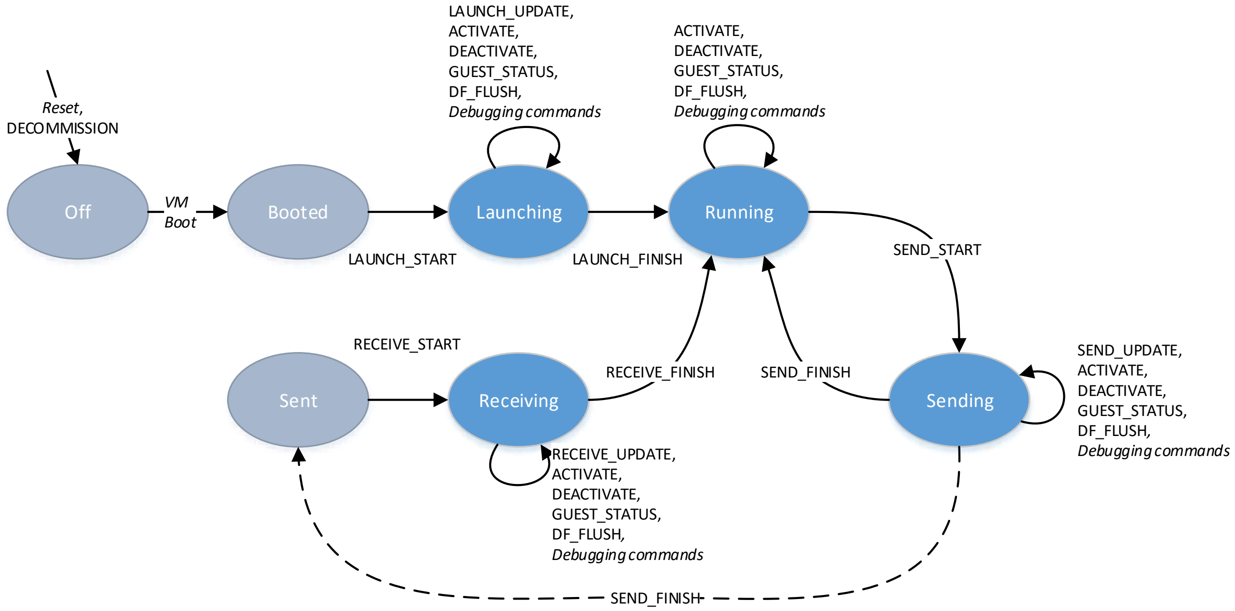


Figure 2. Guest Management State Machine

See Figure 2 for an illustration of the state machine. If an edge does not exist in a state for a command, no action is taken and an error is returned to the caller.

The reset edge into the off state represents a power cycle or reboot taken by the system that causes volatile storage to be cleared.

The debugging commands are described in Chapter 7 on page 61. These commands allow the hypervisor to decrypt guest memory. The guest's policy may disallow debugging.

The SEND_FINISH command returns the guest back to the running state. The dashed edge in the state machine from the sending state to the sent state reflects the fact that the guest is essentially copied, where one copy continues running and the other is sent. If the hypervisor needs to migrate the guest, it could invoke DECOMMISSION immediately afterward to remove the running guest from the platform.

6.1 LAUNCH_START

Command to initiate the process to launch a guest into SEV-enabled mode.

6.1.1 Usage

This command is used to prepare a guest for the transition into SEV-enabled mode.

This command produces a new guest handle and generates a new VEK.

6.1.2 Actions

The firmware establishes a launch measurement key using

- The guest owner's public ECDH key (DH_PUB_QX and DH_PUB_QY)
- The NONCE generated by the guest owner
- The platform's PDH private key

This master secret establishment is described in Section 2.2.2 on page 21. Section 2.2.3 on page 22 describes the derivation of the launch measurement key from the master secret.

A guest handle is returned to the caller in the HANDLE field. This guest handle is unique to all guests on the platform and is used by the hypervisor to manage the guest from this point onward. The guest handle refers to the firmware's internal structures that describe the current state of the guest. The guest's VM Encryption Key (VEK) is generated at this point and stored in the guest state structure. The guest's policy is also stored in this state structure.

If the hypervisor requests this guest to share keys with another guest (FLAGS.KS=1), then the VEK of the other guest specified in HANDLE is used for this guest. For key sharing to be allowed, POLICY.KS must be clear and POLICY.DBG, POLICY.DOMAIN, and POLICY.SEV should be identical for both guests. If key sharing is not requested (FLAGS.KS=0), HANDLE is ignored and a newly generated VEK is used for memory encryption.

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.

After successful completion of this command, the guest is transitioned to the launching state.

If the platform is in the initialized state, the successful completion of this command transitions the platform into the working, respectively. The platform must be in the initialized or working states.

6.1.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In, Out	HANDLE	In: If FLAGS.KS is set to 1, share key with guest referenced by this handle. Otherwise ignored. Out: New guest handle. Nothing written on failure.
4	In	FLAGS	Guest configuration flags: Bit 0: (KS) Key sharing is requested Bits 31:1 are reserved. SBZ
4	In	POLICY	Guest flags describing its policy. See Chapter 3 on page 23 for the policy format.
32	In	DH_PUB_QX	The Qx parameter of the owner’s ECDH public key
32	In	DH_PUB_QY	The Qy parameter of the owner’s ECDH public key
16	In	NONCE	A nonce freshly generated by guest owner

6.1.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the initialized or working states
CMDBUF_TOO_SMALL	The command buffer is too small
INVALID_GUEST	The guest handle is invalid

6.2 LAUNCH_UPDATE

This command is used to continue required steps in the launching process for the target guest.

6.2.1 Usage

For the LAUNCH_UPDATE command, the platform must be in the working state, and the guest must be in the launching state, else an error is returned. The LAUNCH_UPDATE command provides a method for physical pages of the guest to be encrypted securely by the firmware.

During launch, the firmware builds a measurement of the memory contents that is encrypted. This measurement can then be used by the guest owner (or the guest owner's delegate) to verify the guest was successfully launched on the SEV enabled platform.

6.2.2 Actions

The launch measurement is updated with the contents of each memory region in the order contained within the LAUNCH_UPDATE command. The key used for the measurement process is the launch measurement key of the guest associated with the HANDLE parameter. See Section 2.2.3 on page 22 for details concerning the measurement algorithm.

This command requires that the guest be already activated using the ACTIVATE command. If the guest has not been activated, this command fails without altering the provided memory regions.

Once the measurement process has completed, each memory region is then encrypted in place with the VEK of the guest associated with the HANDLE parameter.

The system physical addresses must be 16-byte aligned. It must also be a valid physical address.

6.2.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being launched
4	In	N	Number of memory regions to encrypt
8	In	PADDR1	System physical address of first memory region to encrypt and measure. Must be 16-byte aligned.
4	In	LENGTH1	Length of PADDR1 region. Must be divisible by 16.
…			
8	In	PADDRN	System physical address of Nth memory region to encrypt and measure. Must be 16-byte aligned.
4	In	LENGTHN	Length of PADDRN region. Must be divisible by 16.

6.2.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the launching state
CMDBUF_TOO_SMALL	The command buffer is too small
INACTIVE	The guest is currently inactive

Return Value	Reason
INVALID_ADDRESS	An address is invalid
INVALID_GUEST	The guest handle is invalid

6.3 LAUNCH_FINISH

Finalize launching the guest.

6.3.1 Usage

After the hypervisor has finished encrypting the guest's memory (via the firmware LAUNCH_UPDATE command), the hypervisor concludes the launch operation with this command. The LAUNCH_FINISH command returns a measurement value to the hypervisor. This measurement value can be handed to the guest owner (or the guest owner's delegate) to verify that the guest was launched successfully.

The guest has one or more VMCBs managed by the hypervisor. The register save state areas of these VMCBs represents the current state of the virtual CPUs (VCPU) of the guest. Each of these save state areas are provided by the hypervisor to the firmware. The firmware includes these save state areas in the measurement value.

6.3.2 Actions

The measurement value is updated with the selected bytes of each VCPU in the order provided in the LAUNCH_FINISH command issued by the hypervisor. The *i*th byte of the VCPU is selected if the *i*th bit of the VCPU mask is set to one (1), where bits are ordered from least to most significant within each byte. In other words, if the *j*th bit of the *k*th byte of the VCPU mask is set then the (*j+k*8*) byte of the VCPU is included in the measurement.

The measurement is then updated with the VCPU_COUNT value. The measurement is then finalized and provided in the MEASUREMENT field. The key used for the measurement is the launch measurement key of the guest established during the LAUNCH_START command. See Section 2.2.3 on page 22 for more details regarding the launch measurement.

When the LAUNCH_FINISH command is issued, the platform must be in the working state, and the guest must be in the launching state or an error is returned. After successful completion of this command, the guest is transitioned into the running state.

The system physical addresses must be 16-byte aligned. It must also be a valid physical address.

6.3.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes

4	In	HANDLE	The guest being launched
32	In	MEASUREMENT	Host physical address to store measurement
4	In	VCPU_LENGTH	The length of each VCPU in bytes
8	In	VCPU_MASK_ADDR	A pointer to the mask of bytes to include in the measurement. L is equal to ceiling (VCPU_LENGTH / 8). The remaining bits must be zero.
4	In	VCPU_COUNT	Number of VCPUs for this guest
8	In	VCPU1	System physical address of first VCPU. This must be the bootstrap CPU
8	In	VCPU2	System physical address of second VCPU
…			
8	In	VCPUn	System physical address of nth VCPU, where n is VCPU_COUNT.

6.3.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the launching state
CMDBUF_TOO_SMALL	The command buffer is too small
INVALID_ADDRESS	An address is invalid
INVALID_GUEST	The guest handle is invalid

6.4 SEND_START

Begins the guest send process.

6.4.1 Usage

The hypervisor plans to send the guest to a target platform either for the purpose of saving to disk or to migrate the guest to a target platform.

There are four possible target platform types:

- SEV capable platforms inside this platform’s domain
- SEV capable platforms outside this platform’s domain
- Non-SEV platforms inside this platform’s domain

- Non-SEV platforms outside this platform's domain

The guest has the ability to choose which of these targets are allowed with its policy. The hypervisor indicates which target type it is and the firmware ensures the guest's policy is enforced.

6.4.2 Actions

If FLAGS.DOMAIN is set to 1, the PEK signature (PEK_SIG_R and PEK_SIG_S) is verified using the public key found in PEK_CERT. This is a signature of the following values (where `||' is the concatenation operator)

DH_PUB_QX || DH_PUB_QY || API_MAJOR || API_MINOR || SERIAL

Then, the certificate chain of the PEK (CERT1, CERT2, …, CERTN) is validated. This involves verifying each of their signatures, checking the expiration dates on each of the certificates, and ensuring that the root certificate (CERTN) matches the root certificate of this platform. The PEK signature and the certificate chain must be valid.

If FLAGS.NOSEND is set to 1, this command returns an error.

If FLAGS.DOMAIN is clear, the PEK signature and certificate chain are not checked. POLICY.DOMAIN must be clear in the policy for the guest associated with HANDLE.

If FLAGS.SEV is set to 1, the CEK signature (CEK_SIG_R and CEK_SIG_S) is verified using the CEK public key (CEK_PUB_QX and CEK_PUB_QY). This is a signature of the following values (where `||' is the concatenation operator)

DH_PUB_QX || DH_PUB_QY || API_MAJOR || API_MINOR || SERIAL

Then, the ASK signature (ASK_SIG_R and ASK_SIG_S) is verified using the ASK public key that is retrieved from internal firmware storage. This is a signature of the following values (where `||' is the concatenation operator)

CEK_PUB_QX || CEK_PUB_QY

The ASK and CEK signatures must be valid.

If FLAGS.SEV is clear, the CEK and ASK signatures are not checked. POLICY.SEV must be clear in the policy for the guest associated with HANDLE.

API_MAJOR must be greater than the guest's POLICY.API_MAJOR, or API_MAJOR is equal to POLICY.API_MAJOR and API_MINOR is greater than or equal to POLICY.API_MINOR.

The firmware generates a 16-byte nonce and outputs it to NONCE. The firmware then derives the master secret using the guest owner's public ECDH key (DH_PUB_QX and DH_PUB_QY), the NONCE generated by the firmware, and the platform's PDH private key. This key establishment is described in Section 2.2.2 on page 21.

The firmware generates a transport encryption nonce, a transport encryption key, and a transport integrity key for the purpose of protecting the guest memory image during transmission. The keys are generated according to Section 2.2.5 on page 22 and wrapped as described in 2.2.4. The wrapped keys are output in the WRAPPED_TEK and WRAPPED_TIK fields. The transport encryption nonce is input into the TEN field.

The IV field is output from the firmware and used by the RECEIVE_UPDATE command as the IV input to the block cipher.

The guest's policy is output to POLICY, along with an integrity measurement of the policy which is output to POLICY_MEAS. The measurement uses an HMAC-SHA-256 keyed with the transport integrity key.

The platform must be in the initialized or working states. The guest must be in the running state. After successful completion of this command, the guest is transitioned into the sending state.

6.4.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
16	Out	NONCE	Nonce generated by firmware for KDF freshness
4	Out	POLICY	Guest flags describing its policy. See Chapter 3 on page 23 for the policy format.
32	Out	POLICY_MEAS	HMAC of POLICY keyed with the transport integrity key.
24	Out	WRAPPED_TEK	The wrapped transport encryption key
8	Reserved.		
24	Out	WRAPPED_TIK	The wrapped transport integrity key
8	Reserved		
16	Out	TEN	The transport encryption nonce for key wrapping
16	Out	IV	The IV for the transport encryption block cipher
4	In	HANDLE	The guest being sent
4	In	FLAGS	Flags for this command: Bit 0: (DOMAIN) Target entity is part of the domain Bit 1: (SEV) Target entity is SEV capable platform Bits 31:2 are reserved. MBZ
1	In	API_MAJOR	API major version of target platform
1	In	API_MINOR	API minor version of target platform
2	Reserved.		

Size (bytes)	In / Out	Name	Description
4	In	SERIAL	Platform serial number
32	In	DH_PUB_QX	The Qx parameter of the target's DH public key
32	In	DH_PUB_QY	The Qy parameter of the target's DH public key
32	In	PEK_SIG_R	The r component of the PEK signature
32	In	PEK_SIG_S	The s component of the PEK signature
32	In	CEK_SIG_R	The r component of the CEK signature
32	In	CEK_SIG_S	The s component of the CEK signature
32	In	CEK_PUB_QX	The Qx parameter of the CEK public key
32	In	CEK_PUB_QY	The Qy parameter of the CEK public key
4	In	N	Number of certificates in the certificate chain (excluding the PEK certificate)
Variable	In	PEK_CERT	PEK certificate signed by the domain CA. Signed by CERT1.
Variable	In	CERT1	First certificate in the chain. Signed by CERT2.
Variable	In	CERT2	Second certificate in the chain. Signed by CERT3.
…			
Variable	In	CERTn	nth certificate in the chain and the root CA
32	In	ASK_SIG_R	The r component of the ASK signature.
32	In	ASK_SIG_R	The r component of the ASK signature.

6.4.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the running state
CMDBUF_TOO_SMALL	The command buffer is too small
INVALID_CERTIFICATE	The remote PEK or its certificates are not formatted correctly.
POLICY_FAILURE	The guest policy disallows sending this guest to the remote platform
INACTIVE	The guest is currently inactive
BAD_SIGNATURE	The remote PEK or its certificates do not have valid signatures
INVALID_GUEST	The guest handle is invalid

6.5 SEND_UPDATE

Continue re-encrypting the guest's encrypted pages.

6.5.1 Usage

The hypervisor provides a handle to the guest being sent as well as a list of physical addresses of encrypted pages to perform the send operations on. The firmware generates a new IV, decrypts each page with the VEK and then re-encrypts the page with the transport encryption key. This process is performed in such a way that there is no race condition in which the hypervisor can take advantage to access the decrypted data. After this command successfully completes, each of the destination pages contain the re-encrypted contents of the source page.

6.5.2 Actions

The firmware initializes a new measurement context. The measurement used is HMAC-SHA-256 keyed with the transport integrity key. The firmware also initializes a new encryption context. The encryption uses AES-128 in CTR mode with the transport encryption key. The nonce used was derived from the master secret in SEND_START.

This command requires that the guest be already activated using the ACTIVATE command. If the guest has not been activated, this command fails without altering the provided memory regions.

For each source memory region in order provided by the hypervisor, the plaintext contents of that region (starting at SRC_PADDRi and extending LENGTHi bytes) are encrypted using the transport encryption context and saved into the destination memory region (starting at DST_PADDRi and extending LENGTHi bytes). Then, the measurement is updated with the contents of the destination memory region.

The system physical addresses must be 16-byte aligned. It must also be a valid physical address.

The platform must be in the initialized or working states, and the guest must be in the sending state or an error is returned.

6.5.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being sent
4	In	N	Number of memory regions to encrypt
8	In	SRC_PADDR1	System physical address of first memory region to re-encrypt from. Must be 16-byte aligned.
8	In	DST_PADDR1	System physical address of first memory region to re-encrypt into. Must be 16-byte aligned.

4	In	LENGTH1	Length of PADDR1 region. Must be divisible by 16.
…			
8	In	SRC_PADDRN	System physical address of last memory region to re-encrypt from. Must be 16-byte aligned.
8	In	DST_PADDRN	System physical address of last memory region to re-encrypt into. Must be 16-byte aligned.
4	In	LENGTHN	Length of PADDRN region. Must be divisible by 16.

6.5.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the sending state
CMDBUF_TOO_SMALL	The command buffer is too small
INACTIVE	The guest is currently inactive
INVALID_ADDRESS	An address is invalid
INVALID_GUEST	The guest handle is invalid

6.6 SEND_FINISH

Finish sending the guest.

6.6.1 Usage

The hypervisor informs the firmware that the hypervisor has completed re-encrypting the memory of the guest.

6.6.2 Actions

The measurement of the re-encrypted memory is finalized and output into MEASUREMENT. The transport keys are securely deleted from firmware memory.

The platform must be in the working state, and the guest must be in the sending state or an error is returned. After this call completes successfully, the guest transitions to the running state.

6.6.3 Parameters

Size (bytes)	In / Out	Name	Description

4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being sent
32	Out	MEASUREMENT	The measurement of the re-encrypted guest memory

6.6.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the sending state
CMDBUF_TOO_SMALL	The command buffer is too small
INACTIVE	The guest is currently inactive
INVALID_ADDRESS	An address is invalid
INVALID_GUEST	The guest handle is invalid

6.7 RECEIVE_START

Begin the process to receive an SEV-enabled guest.

6.7.1 Usage

The hypervisor has received a guest from an external entity, and initiates a receive process with the RECEIVE_START command. The RECEIVE_START command requests the firmware to generate a guest context. This generated guest context includes a VEK, the provided guest policy, transport encryption and integrity keys derived from the provided ECDH public keys of the originator of the image, and a measurement context. As the receive process continues, the hypervisor provides portions of the guest to be re-encrypted, integrity checked, and prepared to start as an SEV enabled guest.

6.7.2 Actions

The firmware establishes a master secret using the origin's public ECDH key (DH_PUB_QX and DH_PUB_QY), the NONCE generated by the guest owner, and the platform's PDH private key. This key establishment is described in Section 2.2.2 on page 21.

The firmware receives a wrapped transport encryption key and a transport integrity key for the purpose of protecting the guest memory image during transmission. These keys are wrapped as described in Section 2.2.4 on page 22. The wrapped keys are input in the WRAPPED_TEK and WRAPPED_TIK fields. The transport encryption nonce is input into the TEN field.

An integrity measurement of POLICY is generated and compared with POLICY_MEAS. The measurement uses an HMAC-SHA-256 keyed with the transport integrity key. The POLICY_MEAS must be successfully verified, or an error is returned.

Upon command success, a guest handle is returned to the caller in HANDLE. The guest handle is unique to all guests on the platform and is used by the hypervisor to manage the guest from this point onward. The guest handle refers to the firmware's internal structures that describe the current state of the guest. The guest's VM Encryption Key (VEK) is generated at this point and stored in the guest state structure. The guest's policy is also stored in this guest state structure.

If the hypervisor requests this guest to share keys with another guest (FLAGS.KS=1), the VEK of the other guest specified in HANDLE is used for this guest. For key sharing to be enabled, POLICY.KS must be clear and POLICY.DBG, POLICY.DOMAIN, and POLICY.SEV should be identical for both guests. If key sharing is not requested (FLAGS.KS=0), HANDLE is ignored and a newly generated VEK is used for memory encryption.

The major API version of this platform must be greater than the guest's POLICY.API_MAJOR, 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 any of these conditions fail, then an error code is returned.

After successful completion of this command, the guest is transitioned to the receiving state.

If the platform is in the initialized state, the successful completion of this command transitions the platform into the working, respectively. The platform must be in the initialized or working states. See Figure 1 on page 29 in Chapter 5 for more details.

6.7.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In, Out	HANDLE	In: If FLAGS.KS is set to 1, share key with guest referenced by this handle. Otherwise ignored. Out: New guest handle.
4	In	FLAGS	Flags for the receive process: Bit 0: (KS) Key sharing requested Bits 31:1 are reserved. SBZ
4	In	POLICY	Guest flags describing its policy. See Chapter 3 on page 23 for the policy format.
32	In	POLICY_MEAS	HMAC of POLICY keyed with the transport integrity key.
24	In	WRAPPED_TEK	The wrapped transport encryption key
8	Reserved.		

Size (bytes)	In / Out	Name	Description
24	In	WRAPPED_TIK	The wrapped transport integrity key
8	Reserved		
16	In	TEN	The transport encryption nonce
32	In	DH_PUB_QX	The Qx parameter of the origin's ECDH public key
32	In	DH_PUB_QY	The Qy parameter of the origin's ECDH public key
16	In	NONCE	Nonce generated by the origin

6.7.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the running state
CMDBUF_TOO_SMALL	The command buffer is too small
POLICY_FAILURE	The guest policy does not allow key sharing
INACTIVE	The guest is currently inactive
INVALID_ADDRESS	An address is invalid
BAD_MEASUREMENT	The policy measurement is invalid

6.8 RECEIVE_UPDATE

Continue re-encrypting the guest's encrypted pages.

6.8.1 Usage

The hypervisor provides a handle to the guest being received as well as a list of physical addresses of pages to re-encrypt. After this command successfully completes, each of the destination pages contain the re-encrypted contents of the source page.

6.8.2 Actions

The firmware initializes a new measurement context. The measurement used is HMAC-SHA-256 keyed with the transport integrity key. The firmware also initializes a new encryption context. The encryption uses AES-128 in CTR mode with the transport encryption key. The nonce used was derived from the master secret in SEND_START.

This command requires that the guest be already activated using the ACTIVATE command. If the guest has not been activated, this command fails without altering the provided memory regions.

For each memory region contained within the RECEIVE_UPDATE command, the contents of that region (starting at PADDRi and extending LENGTHi bytes) are measured using the transport integrity context. The firmware processes the memory regions in the order provided by the hypervisor. The memory regions are then decrypted using the transport encryption context and re-encrypted with the VEK in place. This is performed in such a way that the hypervisor cannot race to the decrypted data.

The system physical addresses must be 16-byte aligned. It must also be a valid physical address.

The platform must be in the working state, and the guest must be in the receiving state or an error is returned.

6.8.3 Parameters.

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being received
16	In	IV	The initialization vector for this block of memory
4	In	N	Number of memory regions to re-encrypt
8	In	PADDR1	System physical address of first memory region to re-encrypt. Must be 16-byte aligned.
4	In	LENGTH1	Length of PADDR1 region. Must be divisible by 16
…			
8	In	PADDRN	System physical address of Nth memory region to re-encrypt. Must be 16-byte aligned.
4	In	LENGTHN	Length of PADDRN region. Must be divisible by 16

6.8.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the receiving state
CMDBUF_TOO_SMALL	The command buffer is too small
INACTIVE	The guest is currently inactive
INVALID_ADDRESS	An address is invalid
INVALID_GUEST	The guest handle is invalid

6.9 RECEIVE_FINISH

Finish receiving the guest.

6.9.1 Usage

This command is used to complete the receipt of a transmitted guest.

6.9.2 Actions

The transport keys are securely deleted from firmware memory. The measurement provided by the sending platform is input into the MEASUREMENT field. The firmware verifies the measurement using the TIK established in the RECEIVE_START key agreement. If the measurement fails, then an error is returned, the guest handle is invalidated, and the VM encryption keys are deleted.

The platform must be in the initialized or working states, and the guest must be in the receiving state or an error is returned. After this command completes successfully, the guest is transitioned to the running state.

6.9.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being received
32	In	MEASUREMENT	The measurement of the transported guest

6.9.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_GUEST_STATE	The guest is not in the receiving state
CMDBUF_TOO_SMALL	The command buffer is too small
BAD_MEASUREMENT	The policy measurement is invalid
INVALID_GUEST	The guest handle is invalid

6.10 GUEST_STATUS

Returns the current status of the guest.

6.10.1 Usage

Used by the hypervisor to collect the current state of the guest.

6.10.2 Actions

If the HANDLE is invalid, then STATE=0x00 and all other parameters are untouched.

The platform must be in the initialized or working states.

6.10.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	Guest handle
4	Out	POLICY	Guest flags describing its policy. See Chapter 3 on page 23 for the policy format.
4	Out	ASID	Current ASID. If not active, set to 0.
1	Out	STATE	Current platform state: 0x00: Invalid 0x01: Launching 0x02: Receiving 0x03: Sending 0x04: Running All other encodings are reserved.

6.10.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
CMDBUF_TOO_SMALL	The command buffer is too small
INVALID_GUEST	The guest handle is invalid

6.11 ACTIVATE

Activate a guest.

6.11.1 Usage

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.

6.11.2 Actions

The guest must either be inactive or its ASID must be equal to the ASID parameter. The ASID parameter must not be zero and must be a valid ASID. The ASID must not be currently used by another guest.

A WBINVD and subsequent call to DF_FLUSH is required to be executed before the hypervisor executes the ACTIVATE instruction. A WBINVD is required on all cores. The firmware checks that a DF_FLUSH has been performed. If it has not, an error is returned.

The platform must be in the working state. The guest must be in the launching, sending, receiving, or running state.

6.11.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	Guest handle
4	In	ASID	ASID to activate the guest with.

6.11.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
CMDBUF_TOO_SMALL	The command buffer is too small
ACTIVE	Guest is already active
ASID_OWNED	ASID is already owned by another guest
INVALID_ASID	ASID is not valid
WBINVD_REQUIRED	WBINVD has not been executed
DFFLUSH_REQUIRED	DF_FLUSH has not been invoked
INVALID_GUEST	The guest handle is invalid

6.12 DEACTIVATE

Deactivate a guest.

6.12.1 Usage

This command is used to dissociate the guest from its current ASID. The firmware will uninstall the guest's key from the memory controller.

6.12.2 Actions

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

The platform must be in the initialized or working states. The guest must be in the launching, sending, receiving, or running state.

6.12.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	Guest handle

6.12.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
CMDBUF_TOO_SMALL	The command buffer is too small
INACTIVE	Guest is already inactive
INVALID_GUEST	The guest handle is invalid

6.13 DF_FLUSH

Flush the internal data fabric write buffers.

6.13.1 Usage

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

6.13.2 Actions

The firmware checks that the hypervisor executed a WBINVD instruction on all cores. If not, an error is returned.

The firmware executes a data fabric flush which causes all internal CPU write buffers to be flushed. The firmware also records that a flush has been performed for all ASIDs.

The platform must be in the initialized or working states. Otherwise an error is returned.

6.13.3 Parameters

None.

6.13.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
ACTIVE	The guest is active
WBINVD_REQUIRED	WBINVD has not been executed
INVALID_GUEST	The guest handle is invalid

6.14 DECOMMISSION

Decommission a guest.

6.14.1 Usage

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.

6.14.2 Actions

The guest must not be active. Otherwise, an error is returned. The hypervisor must invoke DEACTIVATE for the guest before invoking DECOMMISSION.

All guest context is securely deleted from firmware memory.

The platform must be in the initialized or working states. The guest must be in any state but the invalid state; if this command is invoked during launch, receive, or send operations, all context is lost.

6.14.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	Guest handle

6.14.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state
ACTIVE	The guest is currently active
CMDBUF_TOO_SMALL	The command buffer is too small
INVALID_GUEST	The guest handle 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.

7.1 DBG_DECRYPT

Decrypts a page of guest memory.

7.1.1 Usage

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

7.1.2 Actions

The guest policy must allow debugging.

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

The system physical addresses must be 16-byte aligned. It must also be a valid physical address.

7.1.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being debugged
8	In	SRC_PADDR	System physical address of memory region to debug from. Must be 16-byte aligned.
8	In	DST_PADDR	System physical address of first memory region to write decrypted contents into. Must be 16-byte aligned.
4	In	LENGTH	Length of PADDR region. Must be divisible by 16

7.1.4 Return Values

Return Value	Reason
SUCCESS	Successful completion
INVALID_PLATFORM_STATE	The platform is not in the working state

INVALID_ADDRESS	An address is invalid
CMDBUF_TOO_SMALL	The command buffer is too small
POLICY_FAILURE	Debugging is not allowed by guest policy
INVALID_GUEST	The guest handle is invalid

7.2 DBG_ENCRYPT

Encrypts a region of guest memory.

7.2.1 Usage

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

7.2.2 Actions

The guest policy must allow debugging.

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

The system physical addresses must be 16-byte aligned.

7.2.3 Parameters

Size (bytes)	In / Out	Name	Description
4	In, Out	CBUF_LEN	Command buffer length in bytes
4	In	HANDLE	The guest being debugged
8	In	SRC_PADDR	System physical address of memory region to debug from. Must be 16-byte aligned.
8	In	DST_PADDR	System physical address of first memory region to write encrypted contents into. Must be 16-byte aligned.
4	In	LENGTH	Length of PADDR region. Must be divisible by 16

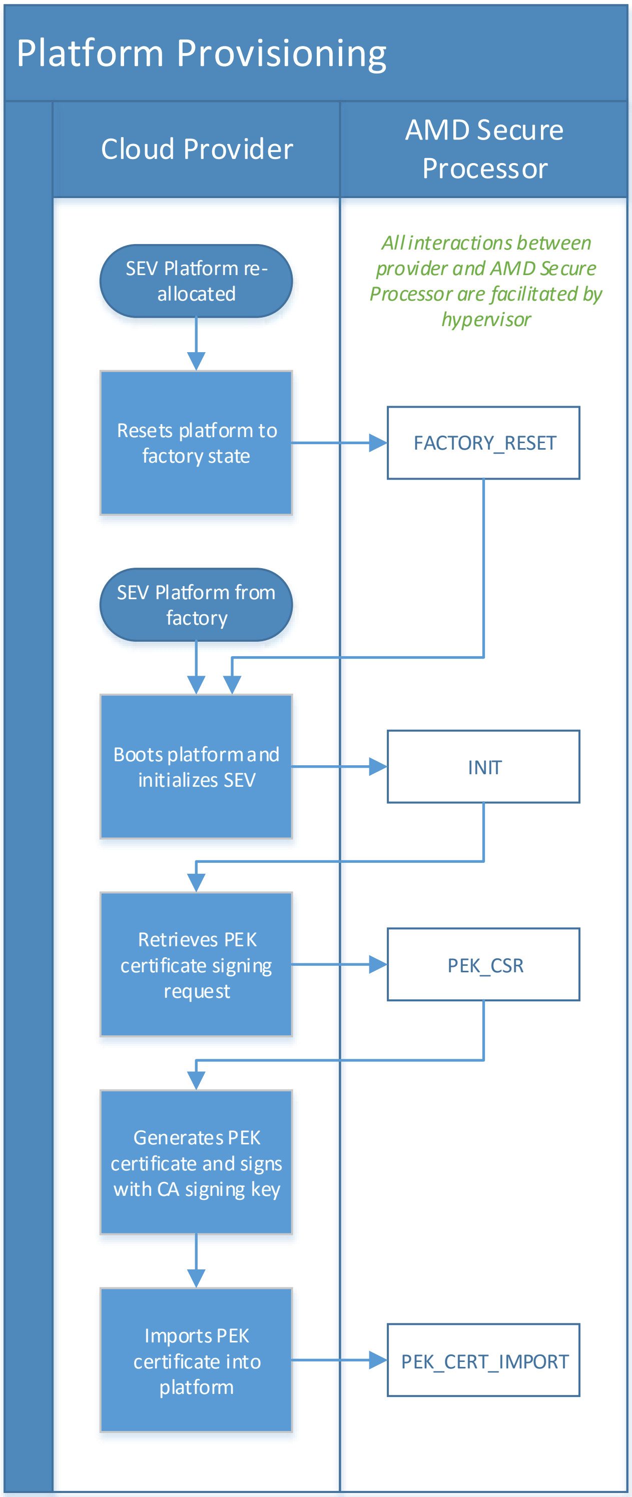
7.2.4 Return Values

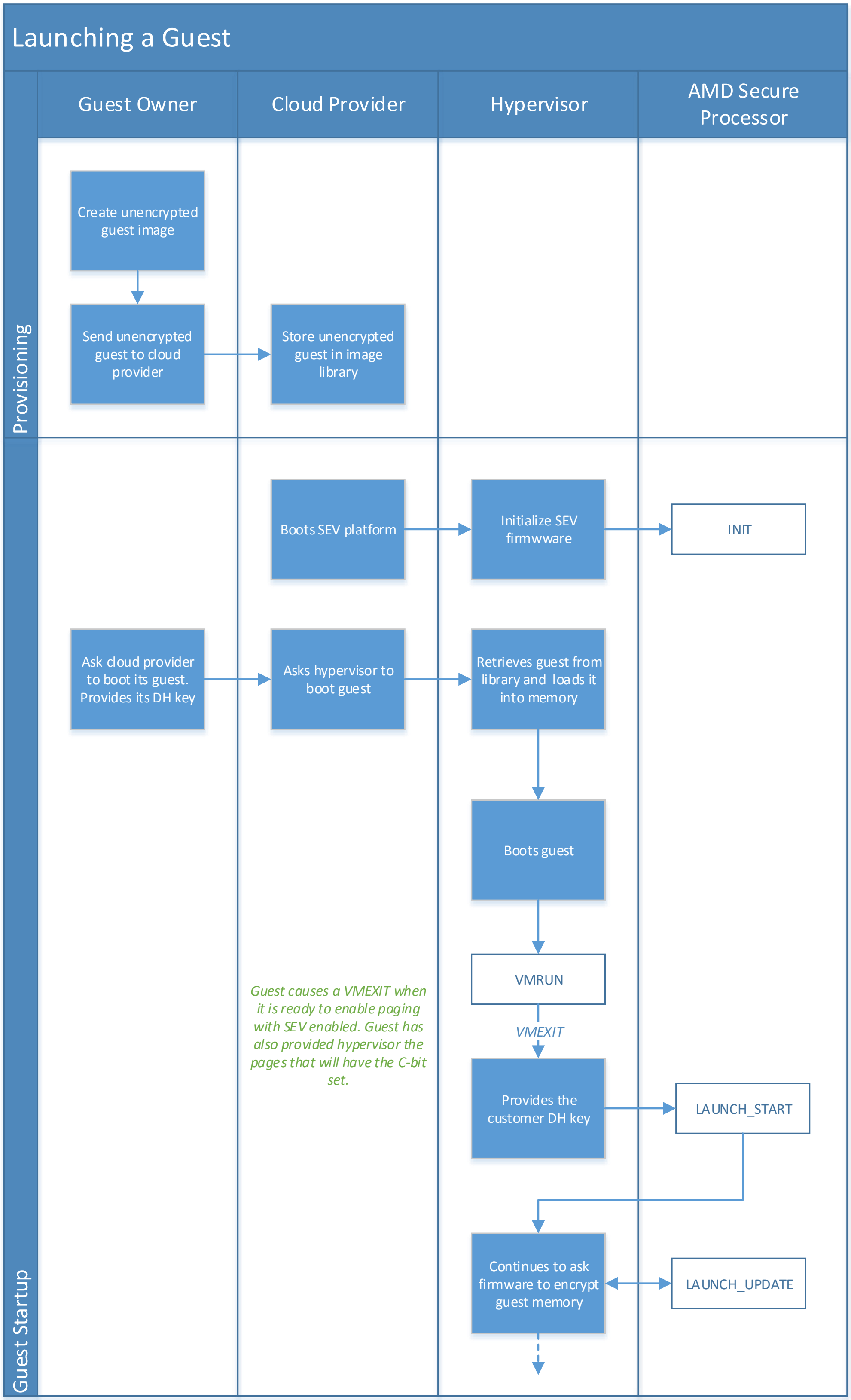
Return Value	Reason
SUCCESS	Successful completion

INVALID_PLATFORM_STATE	The platform is not in the working state
INVALID_ADDRESS	An address is invalid
CMDBUF_TOO_SMALL	The command buffer is too small
POLICY_FAILURE	Debugging is not allowed by guest policy
INVALID_GUEST	The guest handle is invalid

Appendix A Usage Flows

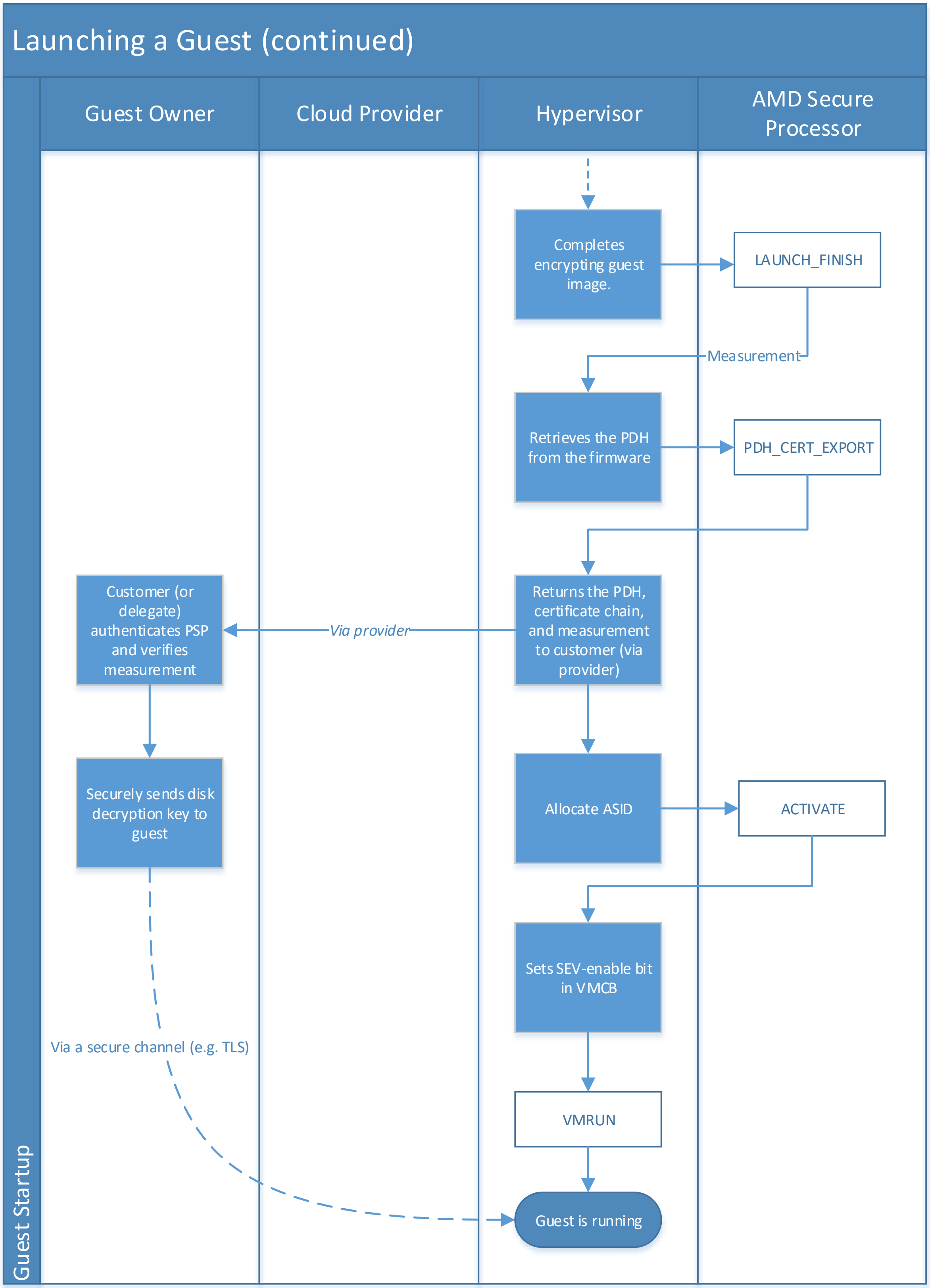
The following flow charts are provided to illustrate the how the usage of the SEV API might be implemented. Note that these are only examples and there may be other implementation strategies.

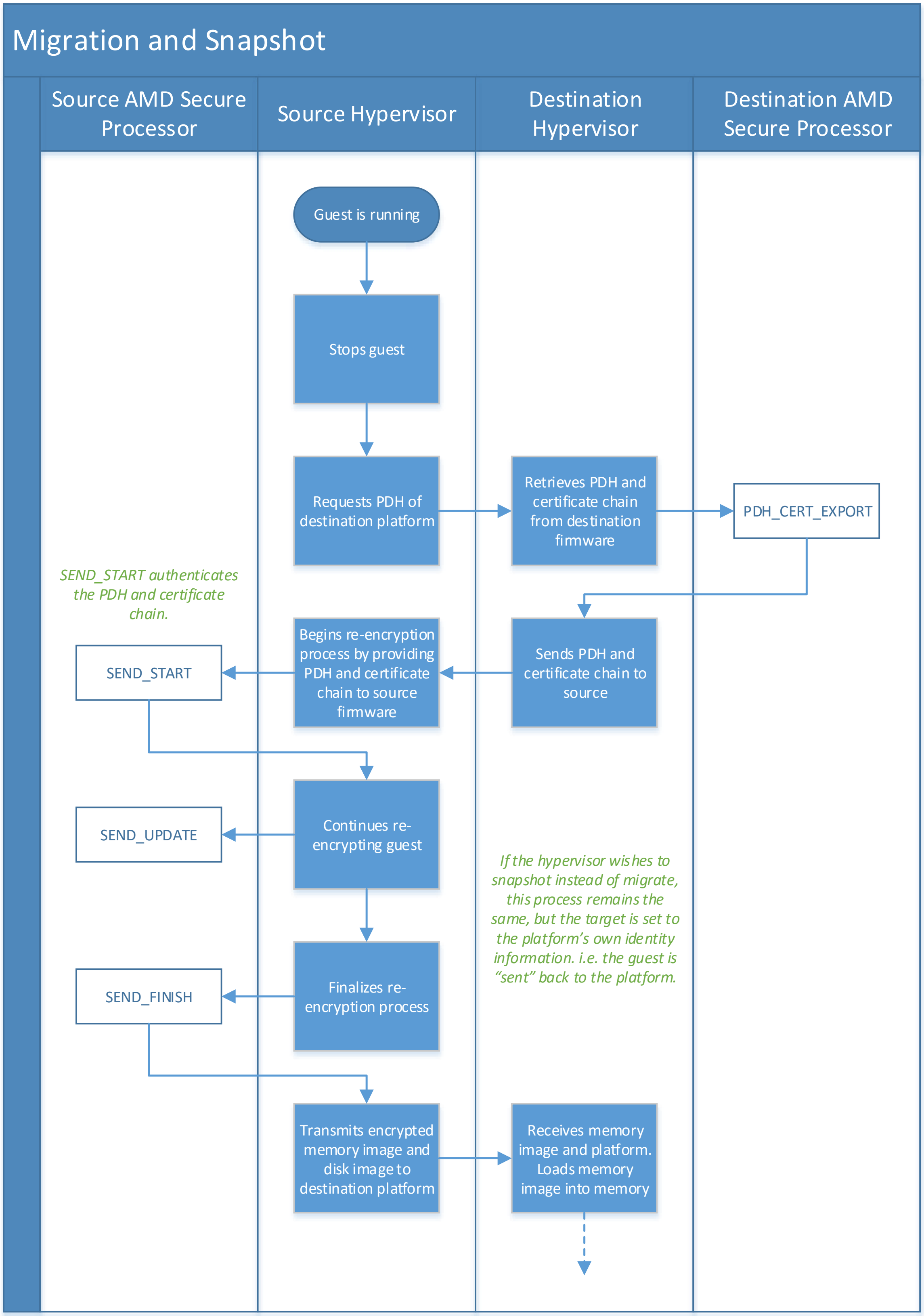

Platform Provisioning
Cloud Provider
AMD Secure Processor
All interactions between provider and AMD Secure Processor are facilitated by hypervisor
SEV Platform re-allocated
Resets platform to factory state
FACTORY_RESET
SEV Platform from factory
Boots platform and initializes SEV
INIT
Retrieves PEK certificate signing request
PEK_CSR
Generates PEK certificate and signs with CA signing key
Imports PEK certificate into platform
PEK_CERT_IMPORT

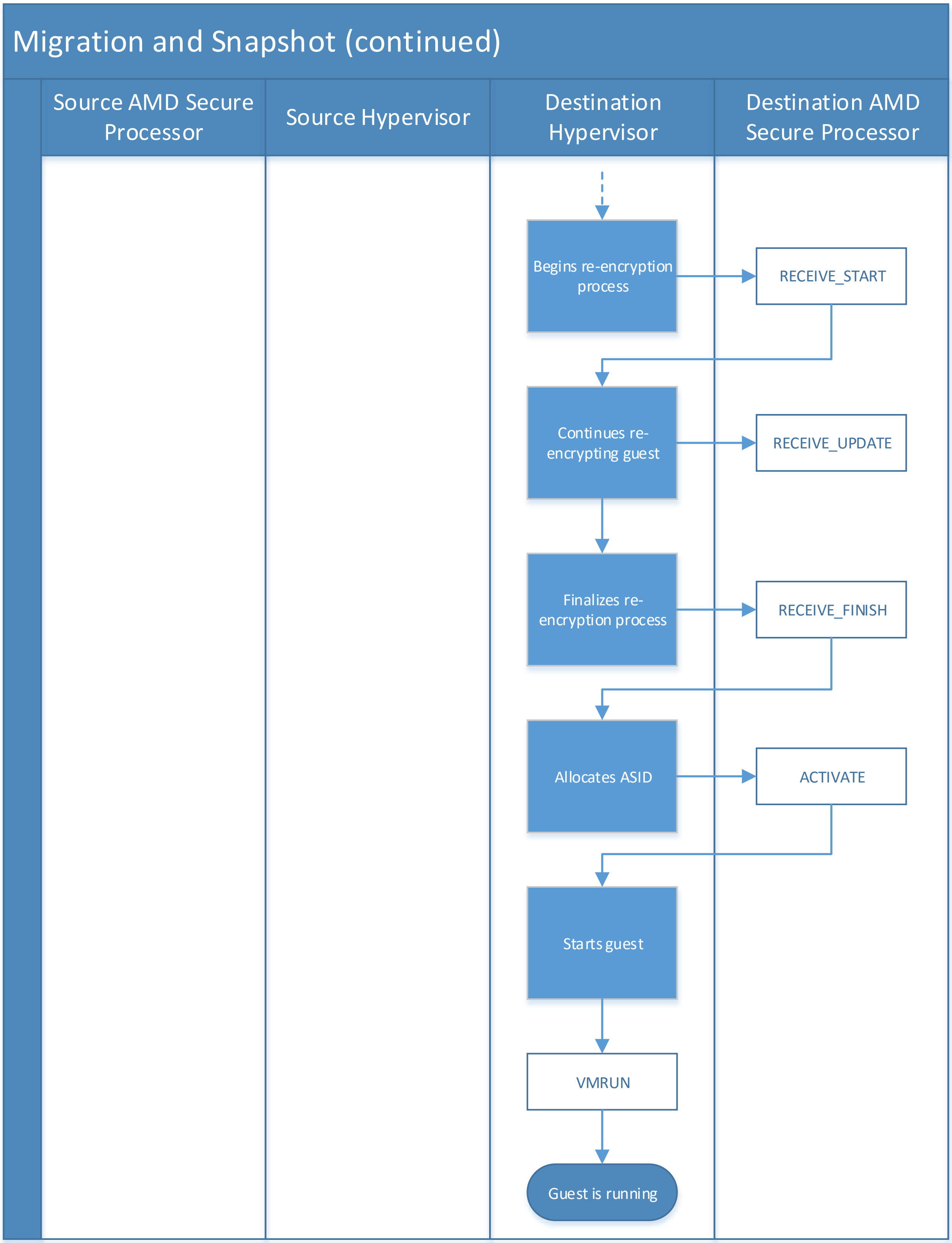

Launching a Guest

Guest Owner
Cloud Provider
Hypervisor
AMD Secure Processor
Provisioning
Guest Startup
Create unencrypted guest image
Send unencrypted guest to cloud provider
Store unencrypted guest in image library
Boots SEV platform
Initialize SEV firmwware
INIT
Ask cloud provider to boot its guest. Provides its DH key
Asks hypervisor to boot guest
Retrieves guest from library and loads it into memory
Boots guest
VMRUN
VMEXIT
Provides the customer DH key
LAUNCH_START
Continues to ask firmware to encrypt guest memory
LAUNCH_UPDATE

Guest causes a VMEXIT when it is ready to enable paging with SEV enabled. Guest has also provided hypervisor the pages that will have the C-bit set.


Launching a Guest (continued)
Guest Startup
Guest Owner
Cloud Provider
Hypervisor
AMD Secure Processor
Completes encrypting guest image.
LAUNCH_FINISH
Measurement
Retrieves the PDH from the firmware
PDH_CERT_EXPORT
Returns the PDH, certificate chain, and measurement to customer (via provider)
Via provider
Customer (or delegate) authenticates PSP and verifies measurement
Securely sends disk decryption key to guest
Allocate ASID
ACTIVATE
Sets SEV-enable bit in VMCB
VMRUN
Via a secure channel (e.g. TLS)
Guest is running


Migration and Snapshot
Source AMD Secure Processor
Source Hypervisor
Destination Hypervisor
Destination AMD Secure Processor
Guest is running
Stops guest
Requests PDH of destination platform
Retrieves PDH and certificate chain from destination firmware
PDH_CERT_EXPORT
SEND_START authenticates the PDH and certificate chain.
SEND_START
Begins re-encryption process by providing PDH and certificate chain to source firmware
Sends PDH and certificate chain to source
SEND_UPDATE
Continues re-encrypting guest
If the hypervisor wishes to snapshot instead of migrate, this process remains the same, but the target is set to the platform's own identity information. i.e. the guest is "sent" back to the platform.
SEND_FINISH
Finalizes re-encryption process
Transmits encrypted memory image and disk image to destination platform
Receives memory image and platform. Loads memory image into memory


Migration and Snapshot (continued)
Source AMD Secure Processor
Source Hypervisor
Destination Hypervisor
Destination AMD Secure Processor
Begins re-encryption process
RECEIVE_START
Continues re-encrypting guest
RECEIVE_UPDATE
Finalizes re-encryption process
RECEIVE_FINISH
Allocates ASID
ACTIVATE
Starts guest
VMRUN
Guest is running