Seamless Firmware Servicing (SFS) Specification

Publication # **58604** Revision: **0.71**
Issue Date: **June 2024**

Advanced Micro Devices

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

Trademarks

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

Contents

List of Figures

List of Tables

Revision History

Date	Revision	Description
May 2024	0.70	Initial public release
June 2024	0.71	Added Section 5.2.2

Chapter 1 Introduction

1.1 Purpose and Audience

This document describes the architecture, assumptions, test methodology, and maintenance of the Seamless Firmware Servicing (SFS) support process. This document facilitates customer personnel involved in developing, testing, and deploying SFS support.

1.2 Glossary

AGESA: AMD Generic Encapsulated Software Architecture

APCB: AGESA Platform Configuration Block

SFS: Seamless Firmware Servicing

PSP: Platform Security Processor

ASP: AMD Security Processor, the replacement for the earlier program's PSP

SEV: Secure Encrypted Virtualization

SMM: System Management Module

SNP: Secure Nested Paging

SPDM: Security Protocol and Data Model

TEE: Trusted Execution Environment

Chapter 2 Background

2.1 Motivation for SFS

AMD created SFS as a secure method to allow non-persistent updates to running firmware and settings without requiring a BIOS reflash and/or system reset. This approach improves system stability by allowing patches to address a few selected “high-benefit/low-risk-to-mitigate” issues on running systems. SFS can improve overall system health without increasing maintenance downtime.

In addition, SFS patching can mitigate some security issues which can reduce the frequency of unplanned maintenance events.

SFS does not patch code that is normally part of customer BIOS and runs on x86, such as SMM, AGESA, or UEFI; nor is it able to update run-once code, such as AGESA boot loader (ABL). The intent of SFS is solely for the purposes stated above.

2.2 Scope of SFS Support

AMD AGESA releases typically contain dozens to hundreds of boot-time and runtime improvements in addition to introducing additional capabilities.

AMD limits the scope of SFS to address priority runtime concerns that affect AMD-provided binary firmware components included in an AGESA release.

SFS does not address anything that runs on the x86 processors. Examples that SFS runs on include ASP firmware, modules, and register settings. Other microprocessors can receive updates.

SFS updates are non-persistent and do not last across reboots. The intent is that the SFS update packages are interim steps between BIOS releases. SFS update packages can extend the time for a customer to validate new BIOS releases in preparation for updating the (typically) flash storage that contains the BIOS.

The design of the SFS update packages apply to a particular PI release with a specified base patch and firmware level; this ensures the update package execution process meets the proper base conditions.

SFS update packages include release notes to indicate what the update patches or resolves.

2.3 SFS Flow

The process of creating an SFS update package follows the flow indicated below (Figure 1).

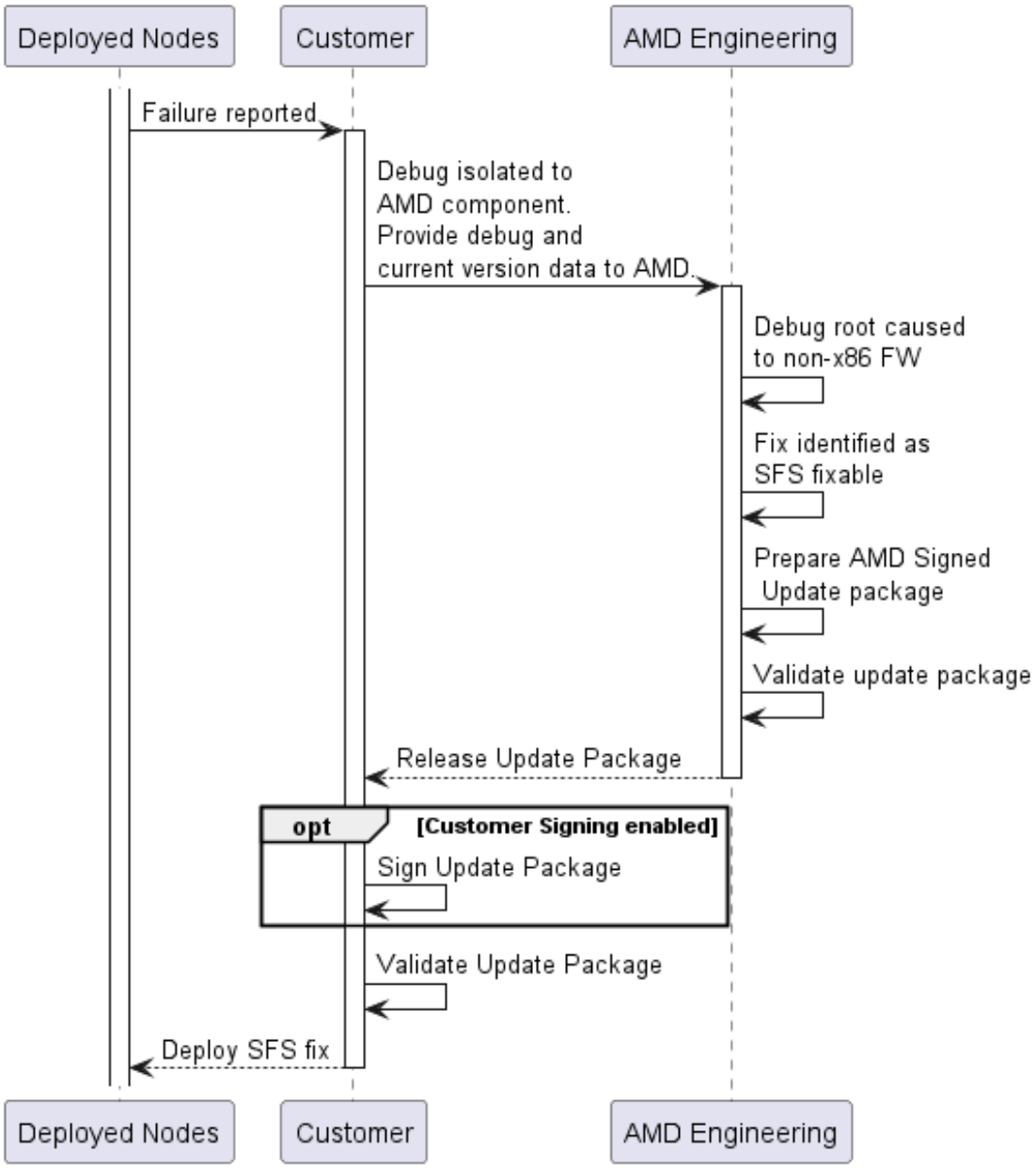

Figure 1. SFS Flow

2.4 Responsibilities

Each customer applying patches is responsible for their decision to proceed with the patches. It is incumbent upon the customer to test and validate any given patch within their own environment. The customer is also responsible for an SFS Deployment Agent (SFSDA) that runs in their environment to initiate and track SFS updates. The SFSDA would utilize the PSP/ASP driver to communicate with AMD's PSP/ASP to execute the patch process.

Chapter 3 SFS Operation

SFS is an AMD firmware feature that facilitates its development and distribution of fixes for selected high priority system issues. Datacenter operators can apply SFS patches without rebooting their system.

AMD works to identify the fixes included in each patch then distributes these patches in a ‘Seamless Firmware Servicing Update Package(s).’

Successfully deploying an SFS update package on a system can cause three distinct types of version numbers to change:

a) The SFS Patch Level (SFSPL) identifies the SFS update package currently applied to the system. An unpatched system has a value of zero. To prevent rollback, only SFS packages with a SFSPL greater than the system's current value can deploy (i.e., the SFSPL must monotonically increase). The SFSPL is the overall runtime patch level. The list of firmware versions combined with the SFSPL uniquely identifies the AMD firmware running on a system.

b) Individual firmware components maintain their own firmware and Security Patch Level (SPL) versions. A newer SFS package will always advance the version, but the functionality introduced in one patch may be reverted in a later patch, if needed.

c) SFS may execute run-once code to modify registers and internal SRAM data that exist outside of a versioned 'firmware component.’ To track these changes, SFS uses the concept of the System Patch Level (SYSPL). The SYSPL is the version number for a virtual firmware component that consists of all run-once code executed by SFS since the last reboot. SYSPL increases monotonically. While the effects of run-once code may be reverted, the history of it executing is indelible.

All that information is available to SFS for it to ensure that any to-be-applied-update applies only to the correct level of existing SOC firmware and operational state. That data is also available to the host OS via an SFS command to get the version levels of the firmware (and SPL, SYSPL, and SFSPL).

For security reasons, AMD cryptographically signs the entire SFS package. As an added feature, customers can opt to co-sign the package. Co-signing enables customer approval of any SFS packages deployed within their datacenters. This requires customers to send (during early boot) a public key to the ASP to apply a patch.

Conceptually, an SFS update package delivers in the form shown below (Table 1). For additional details, see Chapter 5.

Optional SFS Co-Signing Header (Customer)
SFS Package Signing Header (AMD)
SFS Update Package Executable Binary
SFS Package Signature (AMD)
Optional SFS Package Signature (Customer)

Table 1. Patch Signing and General Layout

3.1 SFS Security / Threat Model

The AMD Security Processor (ASP) verifies AMD's signatures on all AMD-owned boot firmware that it retrieves from the ROM. The ASP acts as AMD's root of trust for detection (see *FIPS PUB SP800-193, section 4.1*) for all AMD signed firmware. SFS allows runtime updates to AMD firmware, and SFS extends the ASP's role as the root of trust for detection by requiring the ASP to verify AMD's signature before applying each SFS update package.

AMD's policy has been that the system designer (OEM/ODM) owns the firmware root of trust for update for all the firmware, not AMD (see *FIPS PUB SP800-193 section 4.1* for the difference between root of trust for detection vs update). Unless AMD-provided Platform Secured Boot (PSB) solution is enabled, the system designer (OEM/ODM) is also responsible for root of trust for detection of the system designer owned/signed/provided firmware.

When systems first begin executing OEM/ODM-provided x86 code (i.e., early BIOS code), that code has full read/write access to the ROM's firmware store. This early OEM/ODM trusted code secures the system by enabling restrictions on ROM chip access. Enabling features such as AMD ROM Armor ensures only the OEM/ODM-controlled SMM handler can write to the ROM chip. Only after this lockdown does the OEM's firmware allow untrusted third-party code to execute (such as adapter card ROMs, disk-based boot loaders, and network PXE images). In later stages of execution, OEM/ODM-provided logic then verifies updates before allowing the ROM to apply the updates.

Since SFS offers a means to update running firmware, SFS provides the OEM/ODM the option to co-sign update packages so the OEM/ODM can maintain their role as root of trust for update. During the early boot phase, when the ROM is still in read/write mode, the OEM/ODM may present their public signing key to SFS. Once presented, the ASP will require updates signed by AMD and countersigned by the OEM/ODM's signing key. Maintenance of customer co-signing keys is the responsibility of the customer. AMD does not countersign the customer keys.

Note: *If an attacker manages to subvert the early trusted code, such that they could replace the co-signing key used, the attacker could easily use their escalated privileges to overwrite the*

baseline firmware components in the ROM, which negates any advantage to being able to co-sign updates.

If AMD Platform Secure Boot (PSB) is enabled, the code that installs the SFS co-signature should be part of the code protected by PSB.

3.2 SFS Behavior

Systems boot in an unpatched state, with a baseline firmware level (whatever signed AMD firmware is running). At runtime, customers may choose to apply SFS patches.

3.2.1 Application of SFS Packages

Application of SFS packages must occur at run-time and needs to be reapplied after any system reboot. SFS does not persist over reset. In multi-socket systems, the first sockets' ASP receives the patch communicated to it. Each patch determines if it is on a single or multi-socket system then ensures the patch applies to the local socket correctly, and (as needed) to the additional sockets.

If there are multiple updates, application of the packages must occur sequentially and in numerical order.

3.2.1.1 Enablement

BIOS developers enable SFS by compiling their BIOS with appropriate values for: `APCB_TOKEN_UID_PSP_SFS_ENABLE` and `APCB_TOKEN_UID_PSP_SFS_COSIGN_REQUIRED`.

Both options are off by default. Customers must opt-in and configure the BIOS to allow SFS and if desired co-sign the update packages.

The security processor firmware consumes this information when authenticating whether to allow application of the patch. The table below explains the combinations of these two flags.

SFS enabled?	Co-signing enabled?	Behavior
No	N/A	SFS will not accept commands. BIOS will not accept a public key.
Yes	No	SFS will accept update packages and requires no customer co-signature (SFS will refuse a co-signed package)
Yes	Yes	SFS will accept update packages and require co-signing

Table 2. SFS/Co-signing Flags

In early boot, it is a requirement that the customer BIOS pass their signing key to the ASP, if setup. If both keys are enabled, the SFS package authenticates first against the customer key, and then against the AMD SFS key. This allows customers to create a BIOS with a co-signing key that runs against (for example) a test set of systems and then another one with a different co-signing key that runs on production systems only. This separation prevents SFS update packages, used for the test phase, from deploying on production systems.

Prior to OS start, the customer BIOS provides a public signing key by sending a message (below) to the BIOS-PSP interface. The firmware verifies that the signature of an SFS update package matches the customer in control of the machine during early boot.

3.2.1.2 SET_CUST_SFS_SIGNATURE

Signing requires the customer provide their public key (aka RSA public key) used for SFS package verification during early boot, specifically during, or immediately after, setting up SMM (prior to BOOT_DONE). The customer cannot update the signature after this stage.

The x86 should pass in a pointer to an `MBOX_BUFFER_HEADER`, defined as:

```
typedef struct
{
  UINT32    TotalSize;  ///< Total Size of MBOX_BUFFER (including this field)
  UINT32    Status;     //< Status value
} MBOX_BUFFER_HEADER;
```

The ReqBuffer follows immediately in memory behind the MBOX_BUFFER_HEADER as defined below:

```
typedef struct OEM_CO_SIGN_KEY
{
    UINT32 KeySize; //!< Indicator of 2K or 4K key in bits.
    //!< Struture which holds OEM Co sign key and its data
    UINT8  OemCoSignKey[SIZE_4K_OEM_CO_SIGN_KEY];
} OEM_CO_SIGN_KEY;
```

3.2.1.3 RSA_TOKEN_FORMAT

Customers define the `OemCoSignKey` using the `RSA_TOKEN_FORMAT`.

To co-sign an update package received from AMD, the customer generates an RSA key pair (either 2048-bit or 4096-bit) and signs the package with private key using RSA-PSS signature scheme (`RSASSA_PKCS1_PSS_MGF1_SHA256` for a 2048-bit key, or `RSASSA_PKCS1_PSS_MGF1_SHA384` for a 4096-bit key).

Byte Offset	Content	Description
00h	Version_ID	Version for the key format structure. Set the version as one.
04h	This_KEY_ID	Customer-created key ID
14h	Reserved	Must be zero
24h	Reserved	Must be zero
28h	Reserved	Must be zero
37h	Reserved	Must be zero
38h	EXP_SIZE	Public exponent size
3Ch	MOD_SIZE	Modulus size in bits
40h	EXP	N = 256 or 512 bytes depending on public exponent size
40h + N	MOD	N = 256 or 512 bytes depending on modulus size

Table 3. RSA Token Format

3.2.2 Versioning

Each executable patch checks the existing System Patch Level and/or specific firmware version(s) then determines if it is applicable to that existing patch/firmware level and the other firmware running in the system.

3.2.3 Patchable Items

An update package can patch a register or other value, replace a complete firmware image in a microprocessor, or update portions (e.g., drivers) of a firmware package.

3.2.3.1 SEV/SNP Update Note

AMD SFS automatically commits any SEV update which prevents rollback to previous images. Reverting an SFS updated SEV firmware image requires a system reboot.

Additionally, if SNP is enabled, all pages used for the update package must be `HV-FIXED` pages using the `RMPUPDATE` command. The command is issued by the driver/software sending the package to the SFS interface.

3.2.4 Security and Attestation Note

For attestation requirements, the OS or system/platform management software should request attestation data/evidence via appropriate interfaces & protocols (viz., side-band interface/SPDM or in-band/DPE. SFS provides required measurements to the appropriate attestation data managers for reporting via the above mechanisms.

Some attestation schemes prevent the update of certain firmware components of AMD-provided firmware. AMD will not attempt to modify that firmware.

3.2.5 Resiliency

Each update package is responsible for ensuring that the proper versions of the firmware updated and/or influenced by a patch exist on the system prior to beginning the update process. This ensures the firmware in the system does not get out-of-sync if there are any inter-firmware dependencies.

If any portion of a package cannot fully apply its updates, then application of the package should not occur. Otherwise, the only choice may be to force a reset, which is an undesirable outcome. System testing by AMD and the customer's site will verify that a reset is not necessary.

Chapter 4 Boot Time Application Overview

Update packages are non-persistent across system reboots (warm or cold), so it is important to define a mechanism to apply patches. Customers have the option to apply update patches as part of OS boot, immediately after system boot, or at any time during system operation.

For example, the Linux OS places firmware updates in a location known to the OS, then loads the firmware from that location during system boot. SFS update patches have version numbers and a boot service (or driver install) and can apply patches sequentially by placing the update package files in a location known to the file system.

See section 5.2.1 for information on package naming.

Chapter 5 Sending SFS Commands

5.1 Issuing SFS Commands

Issuing SFS commands requires the following:

1. Memory alignment
    a. The page used for the TEE_EXT_CMD_BUFFER must be at the start of a 2MB aligned page that is 2MB in size.
    b. For the GET_FW_VERSIONS command, the output page must follow the TEE_EXT_CMD_BUFFER address + 4k.
    c. For the UPDATE_FW command, the update package must start at the address of the TEE_EXT_CMD_BUFFER + 4k. If the customer has co-signed the update package, the customer co-signing header must start at TEE_EXT_CMD_BUFFER + 4k.
2. Customer-provided SFS Deployment Agent
    a. Requirements
        i. The TEE_EXT_CMD_BUFFER must be 2MB aligned.
    b. Process
        i. Create a TEE_EXT_CMD_BUFFER (see Table 4) in DRAM.
        ii. Appends an SFS co-signing header in DRAM (see below for definition) if co-signing.
        iii. Appends the entire AMD provided update package.
        iv. Appends the co-signing signature (if used) at the end of the AMD provided update package.
        v. Communicates to the primary ASP firmware the request for SFS by putting an SFS command in the TEE mailbox interface and passing the physical address of the SFS TEE_EXT_CMD_BUFFER in memory into additional mailboxes (see section 5.1.5).

The process of applying an SFS update package follows the flow indicated below (Figure 2).

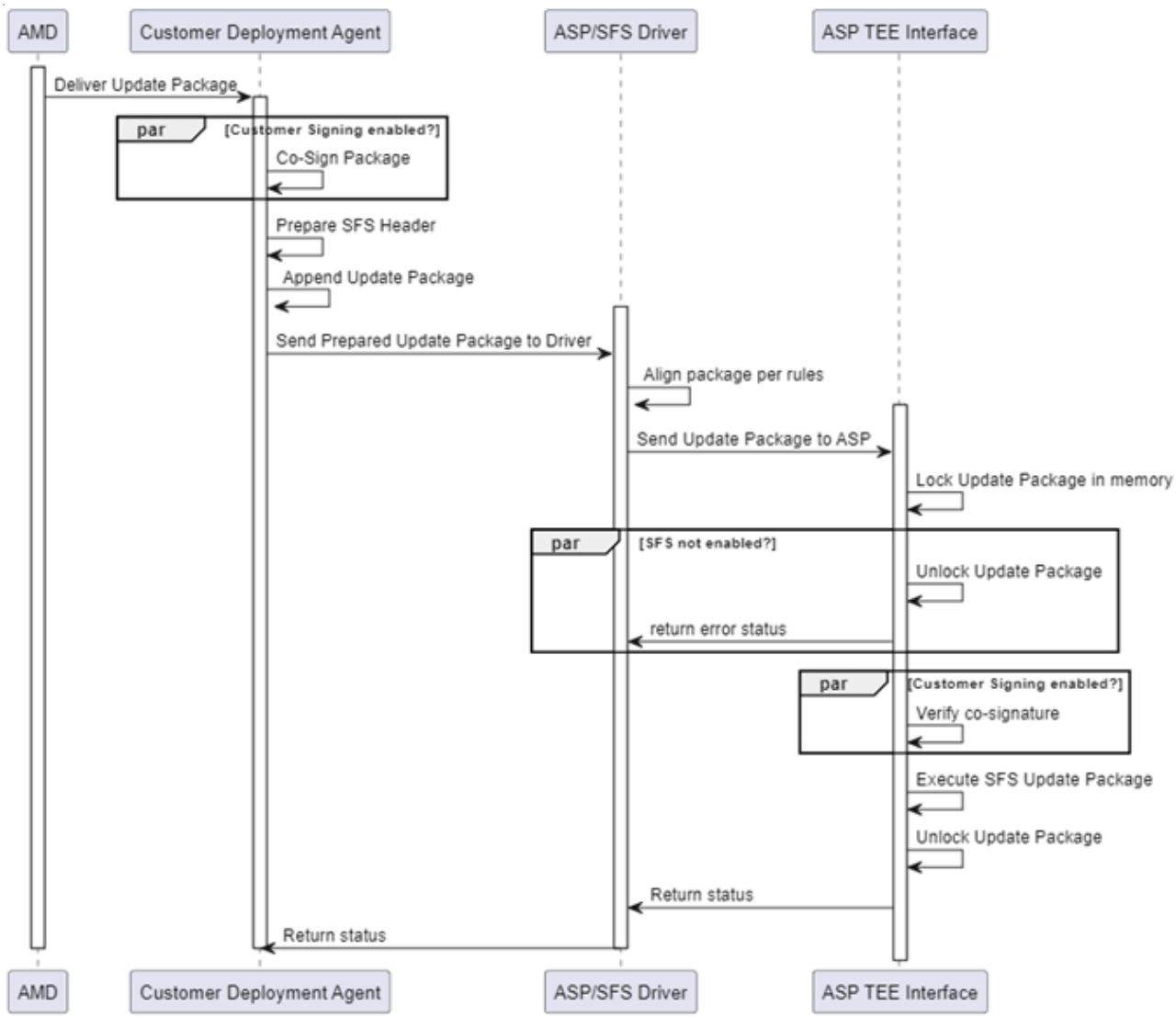

Figure 2. SFS Update Flow

For a detailed verbal description of applying an SFS update package, see below (Figure 3).

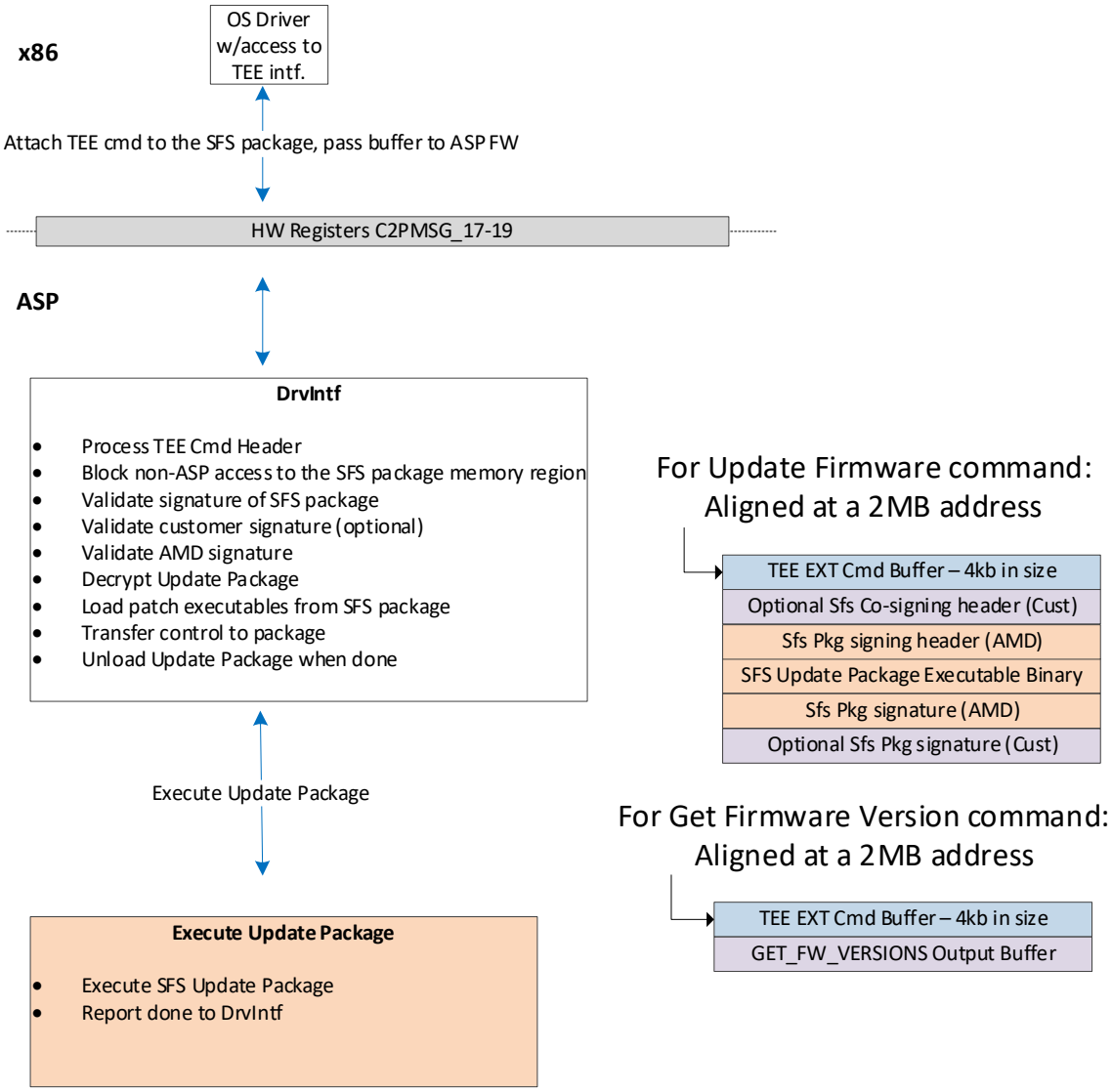

Figure 3. Patch Application Process

5.1.1 TEE Extended Command Buffer (TEE_EXT_CMD_BUFFER)

The structure consists of a `TEE_EXT_CMD` padded to a 4k size; represented by the following example:

```
typedef struct TEE_EXT_CMD_BUFFER
{
    TEE_EXT_CMD Command;
    // For alignment and future expansion
    TEE_UINT8   Reserved[ TEE_EXT_CMD_MAX_SIZE - sizeof(TEE_EXT_CMD) ];
} TEE_EXT_CMD_BUFFER;
```

5.1.2 TEE Extended Command Structure (TEE_EXT_CMD)

This structure is as follows. For SFS, use the following structure for the `TEE_EXT_SUB_COMMAND`.

```
typedef struct TEE_EXT_CMD
{
    TEE_EXT_CMD_HEADER   Header;
    TEE_EXT_SUB_COMMAND  ExtSubCmd;
} TEE_EXT_CMD;
```

5.1.3 TEE Extended Sub Command Header (TEE_EXT_SUB_COMMAND)

Byte Offset	Bits	Dir	Name	Description
00h	31:0	In	TotalSize	Total Size of EXT_CMD_BUFFER (including this header) in bytes
04h	31:0	In	SubCmdId	See TEE SFS Sub-Command ID values below (Table 5)
08h	31:0	Out	Status	Command Execution Status
0ch	31:0	Out	EXT_STATUS	Extended Status (Please report these values to AMD if Status != SFS_SUCCESS)
10h-19ch	31:0	-	Reserved	

Table 4. TEE Sub-Command Header

TEE also supports additional SFS-specific commands.

Command ID Value	Command ID Name	Description
1h	TEE_SUB_CMD_SFS_GET_FW_VERSIONS	
2h	TEE_SUB_CMD_SFS_UPDATE	

Table 5. TEE SFS Sub-Command ID Values

SFS reports the following statuses using a full 32-bit word value.

Status Code	Name	Description
00h	SFS_SUCCESS	Success – patch applied
01h	Reserved	

Status Code	Name	Description
02h	SFS_INVALID_TOTAL_SIZE	Invalid TotalSize
03h	Reserved	
04h	SFS_INVALID_PKG_SIZE	Invalid Image size
05h	SFS_DISABLED	Patching not allowed
06h	SFS_INVALID_CUSTOMER_SIGNATURE	Invalid Customer Signature
07h	SFS_INVALID_AMD_SIGNATURE	Invalid AMD signature
08h	SFS_INTERNAL_ERROR	Please report the extended status to AMD.
09h	SFS_CUSTOMER_SIGNING_ NOT_ALLOWED	Customer signed but not allowed
0ah	SFS_INVALID_BASE_PATCH_LEVEL	Invalid base patch level – Base FW Version mismatch
0bh	SFS_INVALID_CURRENT_PATCH_LEVEL	Invalid current SFS patch level – current patch level mismatch
0ch	SFS_INVALID_NEW_PATCH_LEVEL	Invalid new SFS patch level – less than current patch level
0dh	SFS_INVALID_SUBCOMMAND	Invalid SFS subcommand
0eh	SFS_PROTECTION_FAIL	Payload cannot be protected - possibly not aligned on 2MB boundary.
0fh	SFS_BUSY	Busy – SFS cannot update with this package
10h	SFS_FW_VERSION_MISMATCH	The uploaded FW is less than the existing FW.
11h	SFS_SYSTEM_VERSION_MISMATCH	The current SYS patch level and new SYS patch level are not one apart.
12h	SFS_SEV_STILL_INITIALIZED	SEV clients and SEV SHUTDOWN need to have happened to update SEV.

Table 6. SFS Status Values

5.1.4 Sending a Customer Co-Signed Update Package

This consists of the optional customer co-signature header, followed by the mandatory AMD signature, and the SFS update package itself. The required signatures follow - AMD (and optionally the co-signature).

Byte Offset	Name	Description
00h	Reserved1	Must be zero.
10h	Cookie	Required. Must be set to 5453_5543h
14h	SizeFWSigned	Size of the package in bytes. This would be the entire AMD update package.
18h to FCh	Reserved2	Must be zero.

Table 7. Optional SFS Co-signing Header

5.1.5 Sending the SFS Command to the ASP

Simplified algorithm for sending this command (Table 8). Access to the ASP TEE registers is via the ASP PCI driver.

1. Software needs to interrogate Bit 3 (SFS_ENABLED) of the ASP Feature Register (MP0_C2PMSG63) to determine if SFS is enabled at all.
2. Client polls on Ready flag of Command/Status Register until set to 1.
3. Write the physical address of the SFS Command Header into the CmdRspBufAddr_Lo and CmdRspBufAddr_Hi registers.
4. Write the TEE_IF_EXT_CMD_ID into the Command/Status Register[Command ID] field and write zero to all other fields and update the Command/Status Register.
5. Loop on Ready flag of Command/Status Register while 0 (command in progress).
6. At this point, the Status field of the Reload Firmware Header is valid and combined with the Status in the Command/Status Register, reflects the Reload operations' status.

Register Name	Bits	Name	Description
Command/Status Register	31	Ready	Set by the target to indicate the mailbox interface state 0 – Not ready to handle commands (or handling previous command) 1 – Ready to handle next command

Register Name	Bits	Name	Description
	30:20	-	reserved
	19:16	Command ID	0x0E – TEE Extended Command
	15:0	Status	Set by the ASP to indicate the execution status of the last command
CmdRspBufAddr_Lo	31:0	Addr_Lo	Lower 32-bits of physical address of Command/Response Buffer (TEE EXT CMD Header)
CmdRspBufAddr_Hi	31:0	Addr_Hi	Upper 32-bits of physical address of Command/Response Buffer (TEE EXT CMD Header)

Table 8. ASP Register Definitions

Relative to the base address assigned by BIOS to the PSP device's memory mapped I/O space, Command/Status is register 17 (offset 68), CmdRspBufAddr_Lo is register 18 (offset 72), and CmdRspBufAddr_Hi is register 19 (offset 76).

5.1.6 ASP Process of SFS

1. ASP checks that the DRAM address can be protected from x86 access.
2. ASP verifies the customer signature (if enabled and provided to ASP).
3. ASP verifies the AMD signature.
4. ASP loads and runs the executable patch.
5. ASP updates the current patch level information.
6. ASP returns the status to the software agent.

5.1.7 SFS Sub-Commands

SFS supports the following sub-commands.

5.1.7.1 TEE_SUB_CMD_SFS_GET_FW_VERSIONS

The ASP interface can provide the current level of base firmware for the ASP (and other microprocessors) as well as the current patch level(s).

This command consists of two pages sent to the ASP interface. The first page is the TEE_EXT_CMD_BUFFER which must be exactly one page (4096 bytes). The output data from the command (also 4096 bytes) uses the second page.

To check the firmware versions, the GET_SFS_VERSION_INFO command passes into the SFS handler in the ASP as a subcommand in the SFS Command Header. The address values passed in would point to a contiguous two-page area. The output buffer needs to initialize to C7h in every byte. For SNP systems, the page needs to be an HV-FIXED page. On successful command completion, that buffer contains the data as specified below (All reserved fields are zero).

Items not supported have a 0000_0000h value in the item's field.

Note: *Table 9 is subject to change.*

Offset	Bits	Name	Description
00h	31:0	SFS_API_VERSION	The version of the API that this command supports
04h	31:0	CURRENT_PATCH_LVL	The current SFS patch level as an integer (SFSPL)
08h	31:0	SYS_PATCH_LVL	The current System Patch Level as an integer (SYSPL)
0Ch	31:0	NUM_SUPPORTED_FWS	Number of FW Versions captured out of MAX (MAX = 300)
10h-E20h	31:0	FW_VERSION_INFO	See below
E24h-FFFh	31:0	Reserved	

Table 9. GET_SFS_VERSION_INFO Entry

Each FW_VERSION_INFO entry is a triplet consisting of:

Offset	Bits	Name	Description
00h	31:0	VERSION_TYPE	The Enum of the version type (see Table 11)
04h	31:0	VERSION	The version of the indicated VERSION_TYPE
08h	31:0	SEC_PATCH_LVL	The current Security Patch Level (SPL) as an integer

Table 10. FW_VERSION_INFO Entry

Firmware Enums for VERSION_TYPE:

Note: *Not all these items are currently SFS updatable.*

FW_NAME	Enum Value	Description
DRV_SYS_VERSION	001h	ASP System Driver Version
DRV_SOC_VERSION	002h	ASP SOC Driver Version
DRV_HAD_VERSION	003h	ASP High Availability Driver (Debug) Version
DRV_BOOT_VERSION	004h	ASP Boot Driver Version
DRV_INTF_VERSION	005h	ASP Interface Driver Version
DRV_RAS_VERSION	006h	ASP RAS Driver Version
DRV_SEV_VERSION	007h	ASP SEV Driver Version
DRV_SPDM_VERSION	008h	ASP SPDM Driver Version
DRV_IPKEYMGR_VERSION	009h	ASP IP Key Manager Version
DRV_SFS_VERSION	00Ah	SFS Driver Version
AGESA_BL_VERSION	100h	AGESA Boot Loader Version
ASP_VERSION	101h	ASP Secure OS Version
PMFW_VERSION	102h	Power Management Firmware Version
TMPM_VERSION	103h	Tiered Memory Page Migration Firmware Version
RESERVED	104h	Reserved
MP5_VERSION	105h	MP5 Firmware Version
MPIO_VERSION	106h	MPIO Firmware Version

Table 11. Firmware Enums for VERSION_TYPE

5.1.7.2 TEE_SUB_CMD_SFS_UPDATE

This sub-command tells the ASP to load, verify, and execute the SFS package. The package does not apply any updates until all parts of the update package complete validation as appropriate.

This command consists of a buffer the size of the TEE_EXT_CMD_BUFFER (1 page) followed by the SFS Update Package sent to the ASP interface.

5.2 SFS Package Release

5.2.1 Patch Naming

SFS packages are binary objects that a customer maintains and aligns to AGESA releases.

Package names follow this example:

`fam_<family>-<AGESA-Release>-SFS-<n>.pkg` (e.g., `fam_19h-1006-SFS-1.pkg`)

where:

- `<AGESA-Release>` is AMD's PI release number (e.g., 1002 or 1006).
- `<n>` is a monotonically increasing counting number (i.e., integer) starting at 1.

5.2.2 Release Mechanism

Platform provider and large-scale data fleet operators can work through their AMD customer representatives for SFS update packages.

Chapter 6 Operational Considerations

Update packages have a specific set of firmware and system patch level versions to which they apply; therefore, any update package released must match the specific combination of firmware running on a given system to apply the package.

The update process runs in parallel with x86 and DMA traffic. The goal of package design is that updates have a minimal impact on system performance, at most requiring a brief quiescence of certain activities. Optimally, updates require no quiescing from the OS perspective; therefore, any updates, module, and micro-processor firmware replacements will look at worst like a small slowdown in response – well within the latency windows of the AMD data fabric and customer expectations.

There are very few SFS operations that would require quiescing OS operations or portions thereof. One example of a more impactful update (but still not requiring a system restart) is that SEV or SEV/ES virtual machines need to stop and then restart for an SEV firmware update. SEV/SNP virtual machines are capable of continuing operation across an SFS update to the SEV firmware.