Latest Implementation of AMD SEV-SNP in OVMF

r1chard-lyu
SUSE Labs Developer for EFI
FOSDEM 2025

Whoami

Richard Lyu

Taipei, Taiwan

Work at SUSE

SUSE Labs Developer for EFI

Maintain **OVMF** in SLES and openSUSE

Overview

- **Background**
  - Confidential Computing
  - AMD SEV-SNP
  - OVMF
- **Upstream Status**
  - AMD SEV-SNP in Open Source
  - Commits
- **Integration in Virtualization**
  - Integration
  - SEV Driver

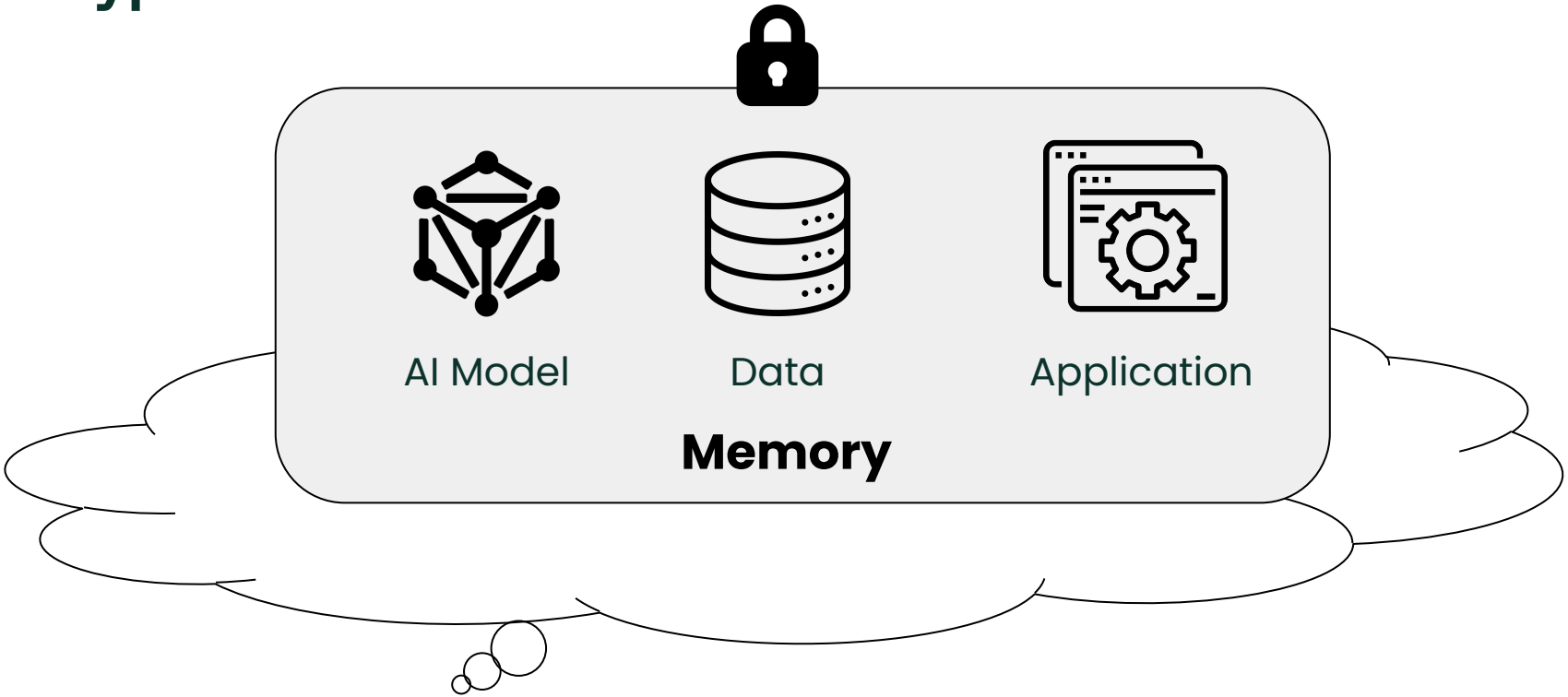
SUSE

SUSE

Background

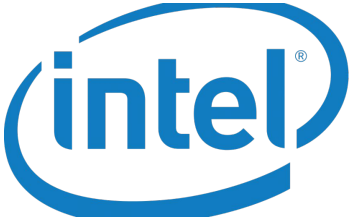
Confidential Computing

Encryption



Providers

- Require a combination of hardware and software



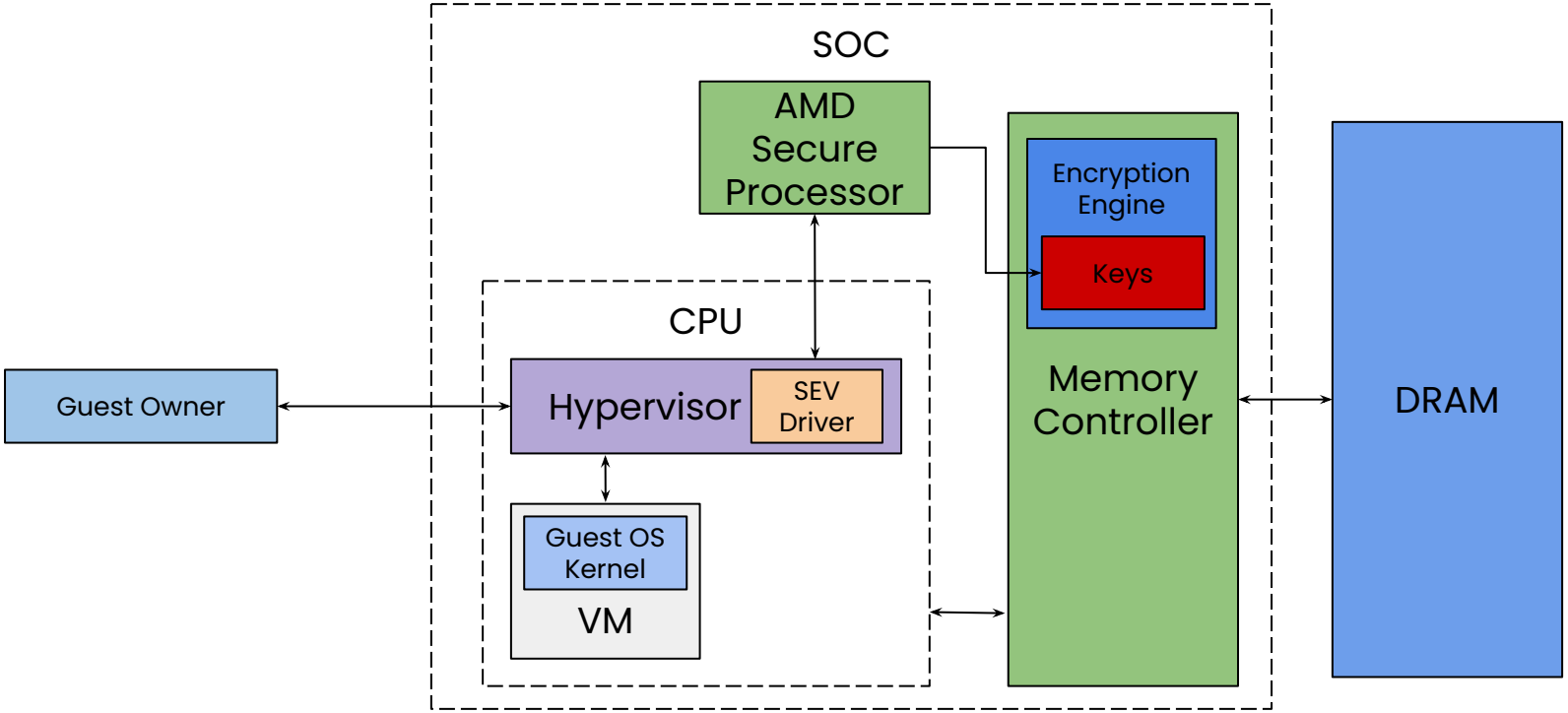
- Delivered with cloud providers or server manufacturers

AMD SEV-SNP

Key Features

- Memory Encryption
- Nested Paging
- Integrity Protection
- Key Management and Attestation

SEV Architecture



OVMF

What is OVMF?

- Open-source UEFI firmware for virtual machines.
- Part of Tianocore's EDK II project.

Key Features:

- UEFI-compliant boot environment.
- Works with QEMU/KVM.
- Simplifies UEFI app development.

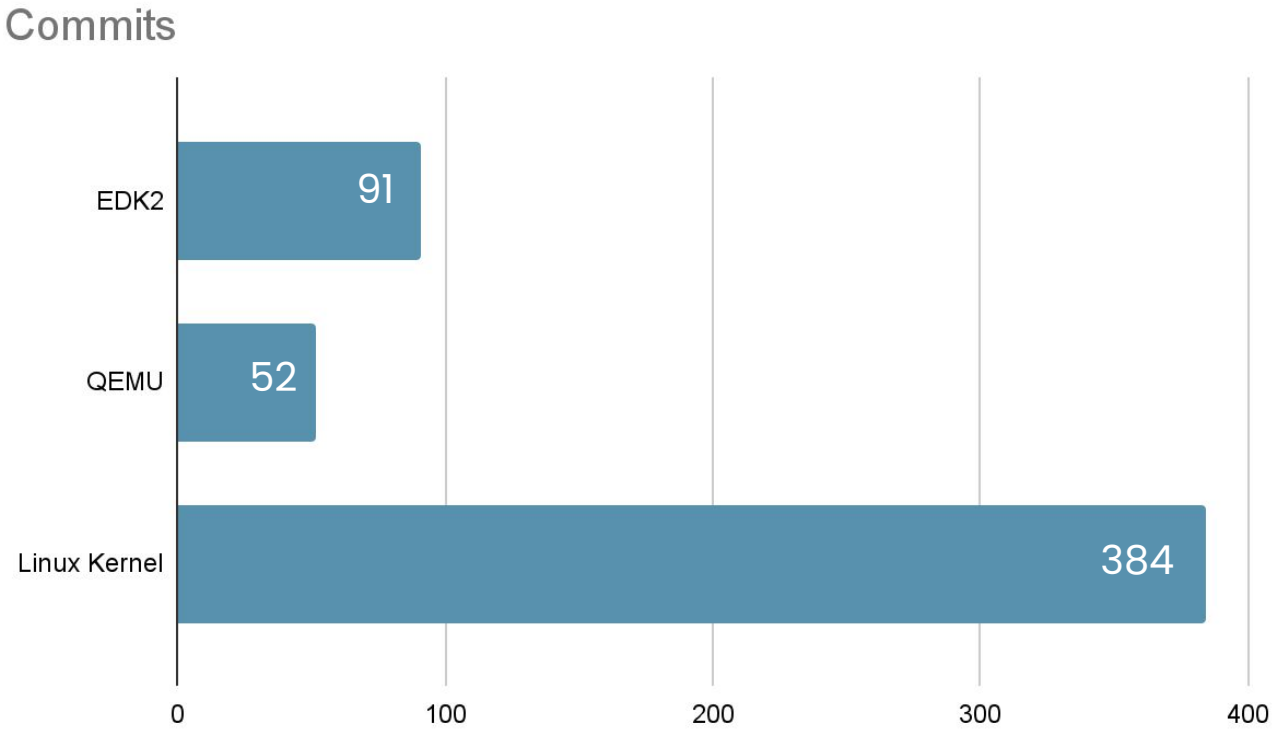
SUSE

Upstream Status

AMD SEV-SNP in Open Source

Technology	Features	EDK2	QEMU	Linux
SEV	Memory encryption	>= edk2-stable201808	>= 2.12	>= 4.15
SEV-ES	Memory encryption + CPU state encryption	>= edk2-stable202008	>= 6.0	>= 5.10
SEV-SNP	Memory encryption + CPU state encryption + Memory integrity protection	>= edk2-stable202405	>= 9.10	>= 6.11

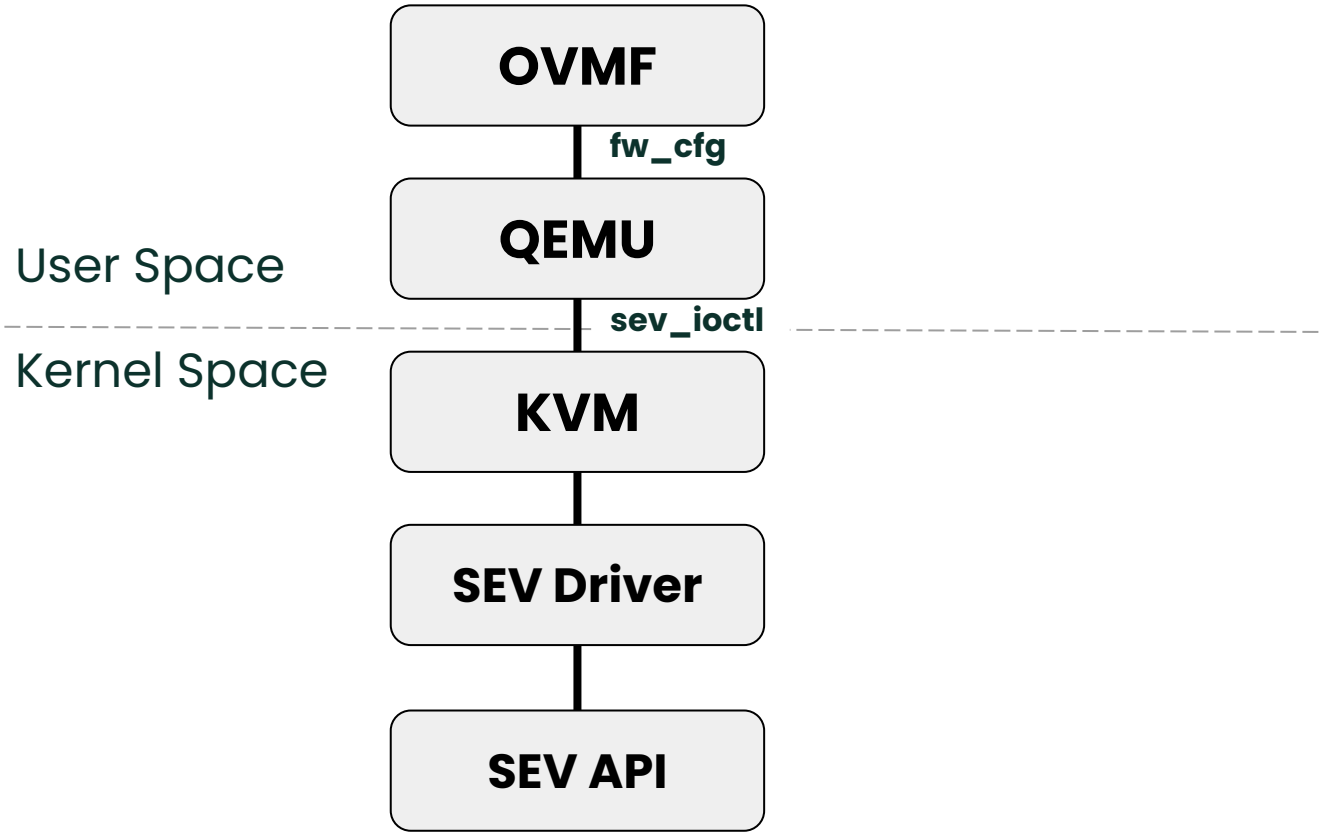
Commits



SUSE

Integration in Virtualization

Integration



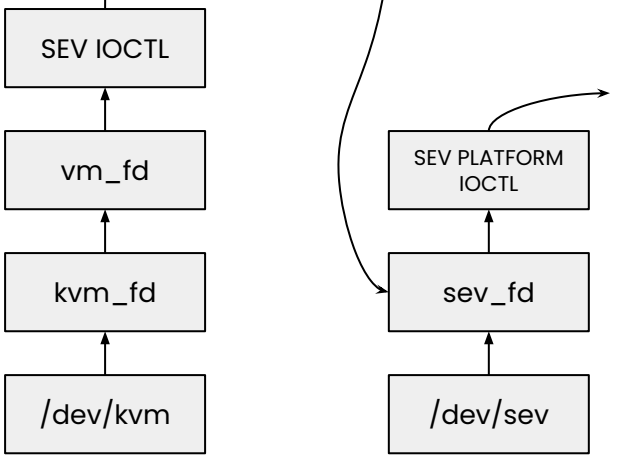
SEV Driver

```
struct kvm_sev_cmd {
        __u32 id;
        __u32 pad0;
        __u64 data;
        __u32 error;
        __u32 sev_fd;
};
```

```
/* Secure Encrypted Virtualization command */
enum sev_cmd_id {
        /* Guest initialization commands */
        KVM_SEV_INIT = 0,
        KVM_SEV_ES_INIT,
        /* Guest launch commands */
        KVM_SEV_LAUNCH_START,
        KVM_SEV_LAUNCH_UPDATE_DATA,
        KVM_SEV_LAUNCH_UPDATE_VMSA,
    …
        KVM_SEV_NR_MAX,
};
```

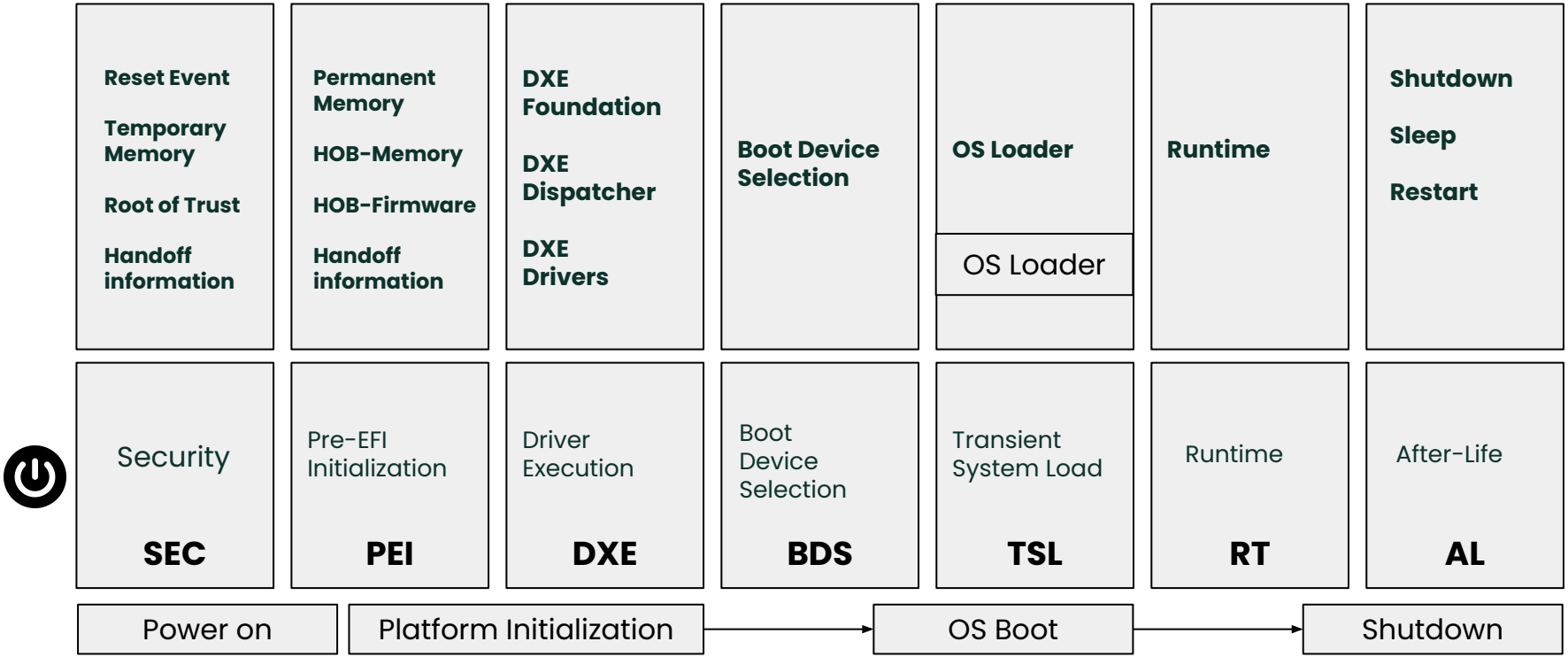
```
struct sev_issue_cmd {
        __u32 cmd;              /* In */
        __u64 data;             /* In */
        __u32 error;            /* Out */
} __packed;
```

```
/**
 * SEV platform commands
 */
enum {
        SEV_FACTORY_RESET = 0,
        SEV_PLATFORM_STATUS,
        SEV_PEK_GEN,
        SEV_PEK_CSR,
        SEV_PDH_GEN,
        SEV_PDH_CERT_EXPORT,
        SEV_PEK_CERT_IMPORT,
        SEV_GET_ID,     /* This command is
deprecated, use SEV_GET_ID2 */
        SEV_GET_ID2,
        SNP_PLATFORM_STATUS,
        SNP_COMMIT,
        SNP_SET_CONFIG,
        SNP_VLEK_LOAD,

        SEV_MAX,
};
```

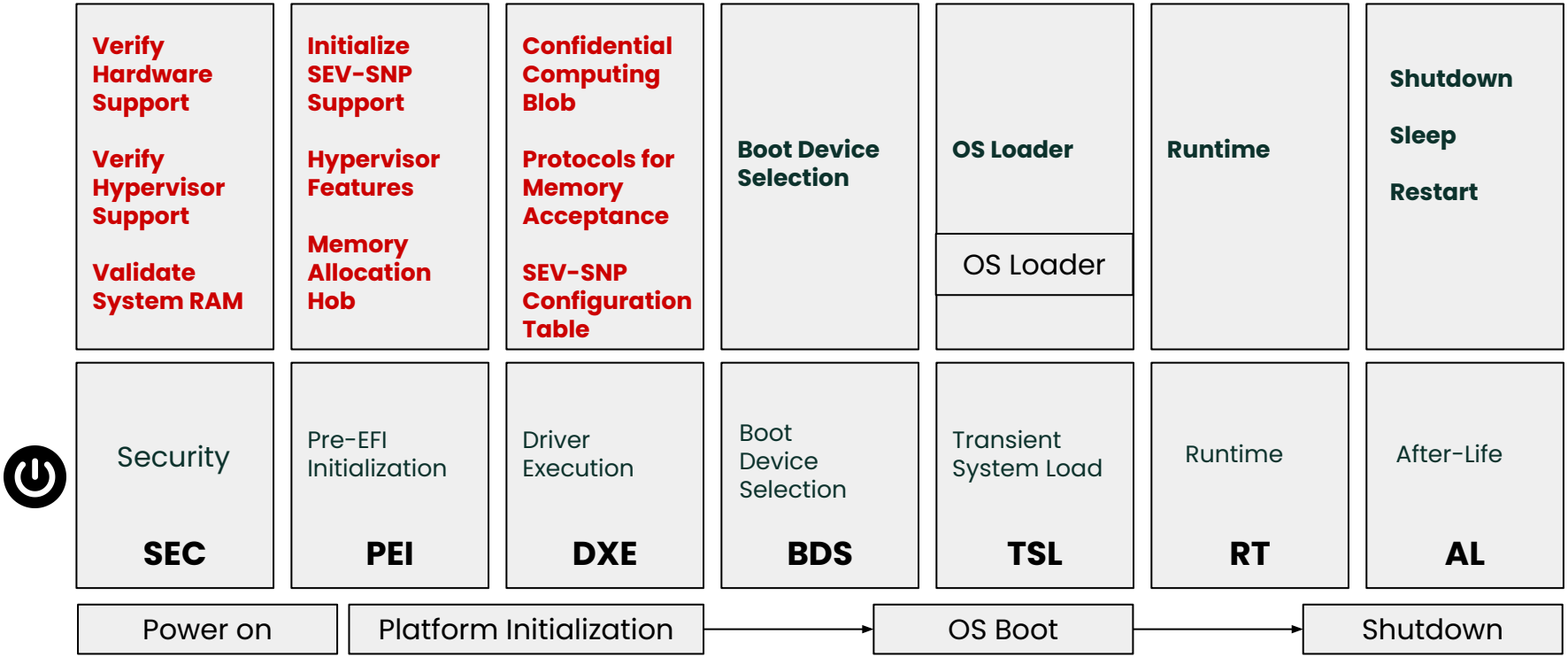


SUSE

PI Architecture Firmware Phases



PI Architecture Firmware Phases



SUSE

Thank You!

Q&A

Reference

[1] AMD64 Architecture Programmer's Manual Volume 2: System Programming
https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24593.pdf
[2] AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions
https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24594.pdf
[3] AMD-V™ Nested Paging
https://www.cse.iitd.ac.in/~sbansal/csl862-virt/2010/readings/NPT-WP-1%201-final-TM.pdf
[4] Accelerating Two-Dimensional Page Walks for Virtualized Systems
https://pages.cs.wisc.edu/~remzi/Classes/838/Spring2013/Papers/p26-bhargava.pdf
[5] Memory virtualization: shadow page & nest page
https://blog.csdn.net/hit_shaoqi/article/details/121887459
[6] AMD-SEV API
https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/programmer-references/55766_SEV-KM_API_Specification.pdf

Reference

[7] AMD Memory Encryption
https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf
[8] QEMU - AMD SEV
https://www.qemu.org/docs/master/system/i386/amd-memory-encryption.html
[9] Linux - KVM
https://www.kernel.org/doc/html/v5.7/virt/kvm/index.html
[10] AMD SEV-SNP White Paper
https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf
[11] AMD SEV in ThinkSystem
https://lenovopress.lenovo.com/lp1545-using-amd-secure-encrypted-virtualization-encrypted-state-sev-es
[12] AMD SEV-SNP Key Attestation
https://www.amd.com/content/dam/amd/en/documents/developer/lss-snp-attestation.pdf
[13] AMD Virtualization Memory Encryption Technology
https://www.linux-kvm.org/images/7/74/02x08A-Thomas_Lendacky-AMDs_Virtualizatoin_Memory_Encryption_Technology.pdf

SUSE

AMD SEV-SNP

Integrity Threats

[11]

THREAT	DESIRED SECURITY PROPERTY	SEV-SNP ENFORCEMENT MECHANISM
REPLAY PROTECTION	Only the owner of a memory page can write that page	Reverse Map Table (RMP)
DATA CORRUPTION	Only the owner of a memory page can write that page	Reverse Map Table (RMP)
MEMORY ALIASING	Every physical memory page can map only to a single guest page at one time	Reverse Map Table (RMP)
MEMORY RE-MAPPING	Every guest page can map only to a single physical memory page at one time	Page Validation

Reverse Map Table (RMP)

[11]

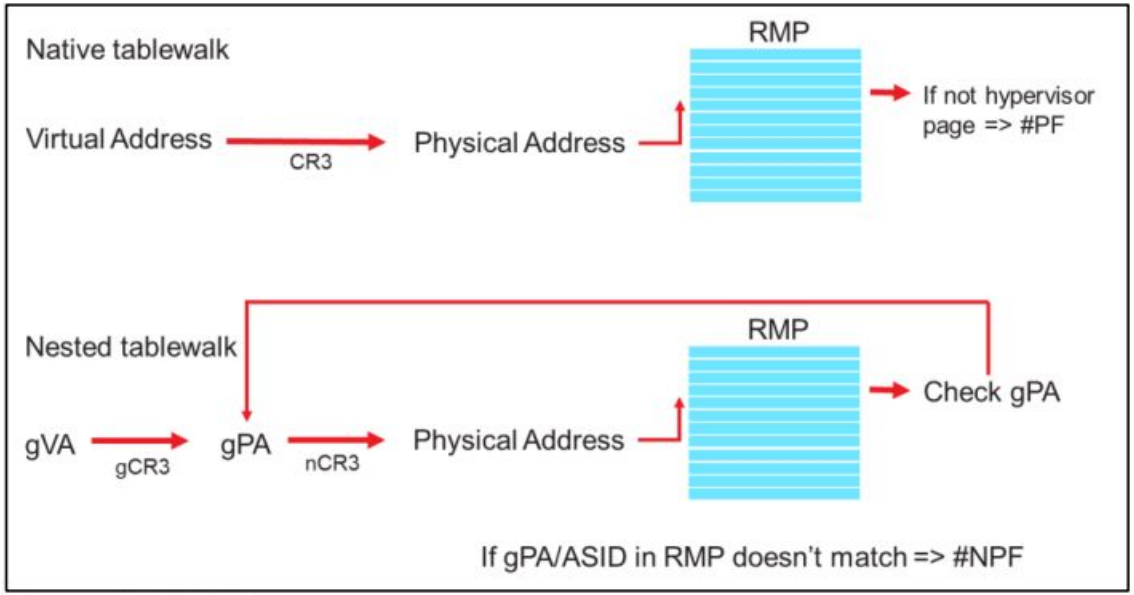


FIGURE 3: RMP CHECKS

Page Validation

[11]

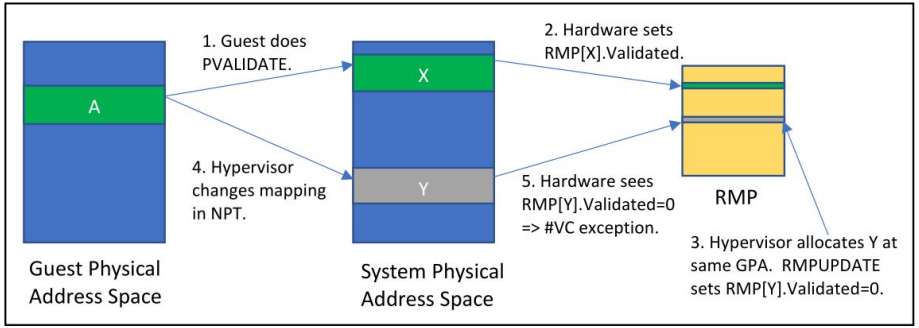


FIGURE 5: PAGE RE-MAPPING ATTACK

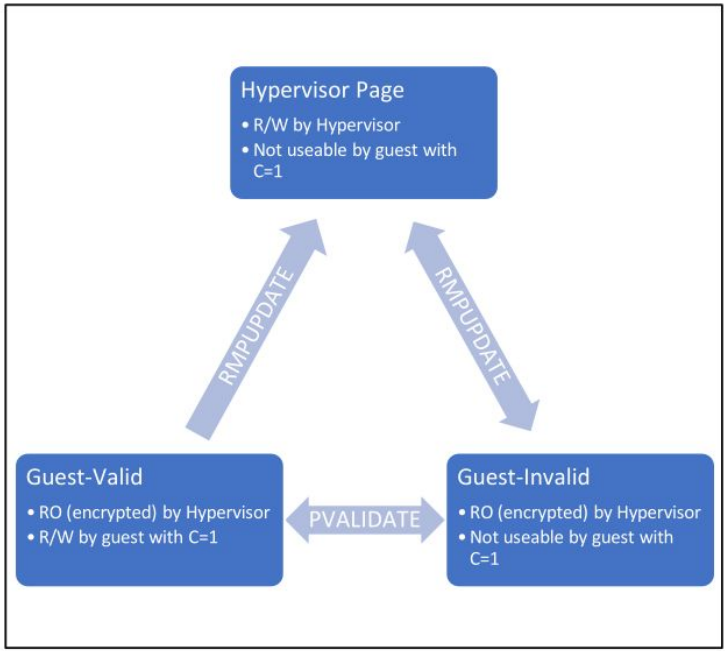


FIGURE 4: BASIC PAGE STATES

SEV-SNP Page States

[11]

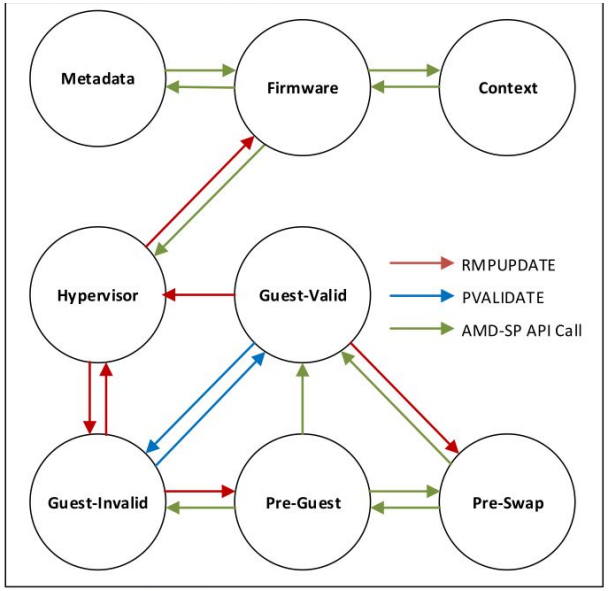


FIGURE 6: PAGE STATE TRANSITIONS

[1]

STATE	DESCRIPTION	NOTES
HYPERVISOR	Default state for otherwise unassigned memory	Used for hypervisor memory, non-SNP-VM memory, and shared (C=0) memory
GUEST-INVALID	Page is assigned to a guest but not ready to be used	Not useable by SEV-SNP VMs until validation has occurred
GUEST-VALID	Page is assigned to a guest and useable	Page may be used as private (C=1) memory by the assigned SEV-SNP VM
PRE-GUEST	Page is Immutable and not validated	Used when initially launching SEV-SNP VMs
PRE-SWAP	Page is Immutable and validated	Used when swapping guest pages to disk
FIRMWARE	Page is Immutable and reserved for AMD-SP use	Typically used as transitory state until AMD-SP has configured the page
METADATA	Page is Immutable and used for metadata	Metadata is used when swapping guest pages to disk
CONTEXT	Page is Immutable and used for context information	Context pages are used by the AMD-SP to identify individual VMs and hold per-VM

Virtual Machine Privilege Levels (VMPL)

- VMPL0 being the highest privilege level and VMPL3 the least privileged.

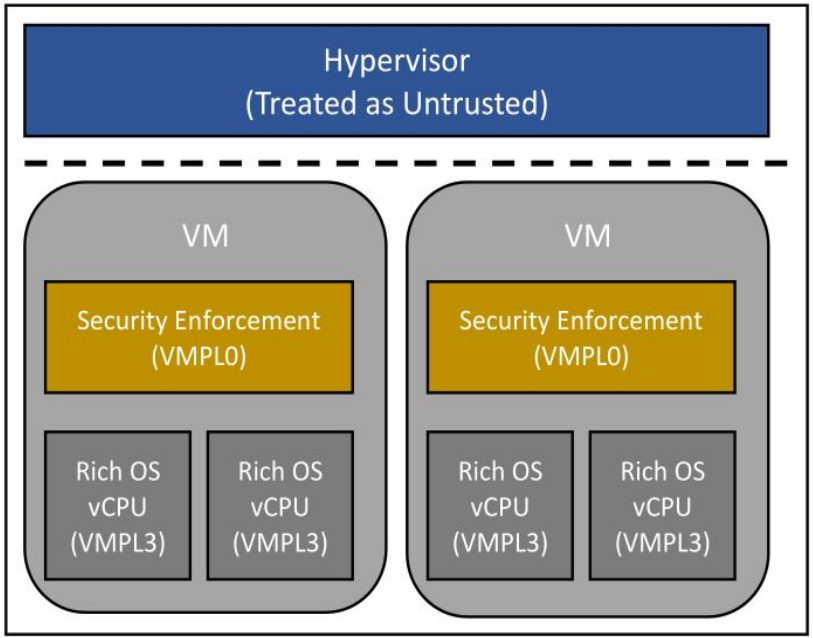


FIGURE 7: VMPLS

Interrupt/Exception Protection

Two optional modes:

- Restricted Injection
- Alternate Injection

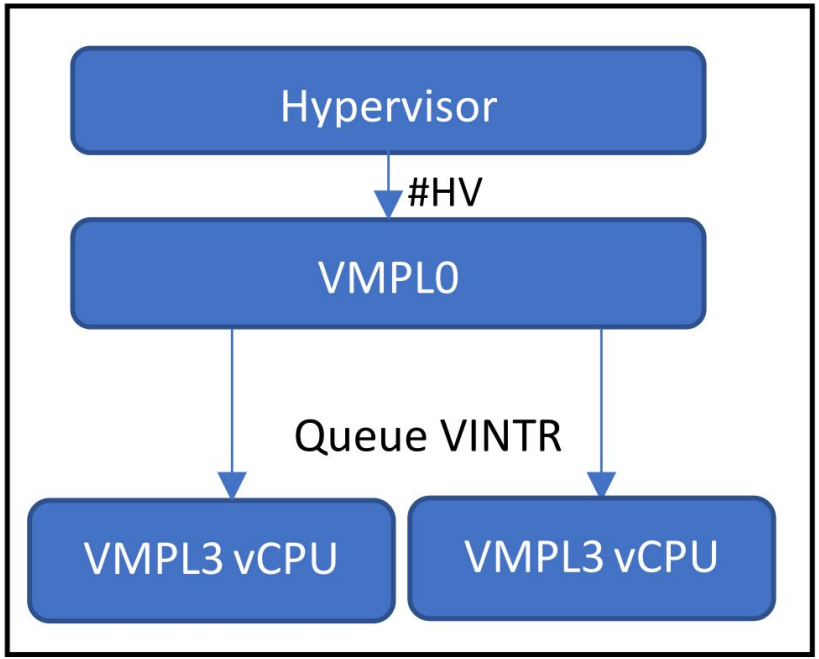


FIGURE 9: VMPL INTERRUPT HANDLING

VM Launch & Attestation

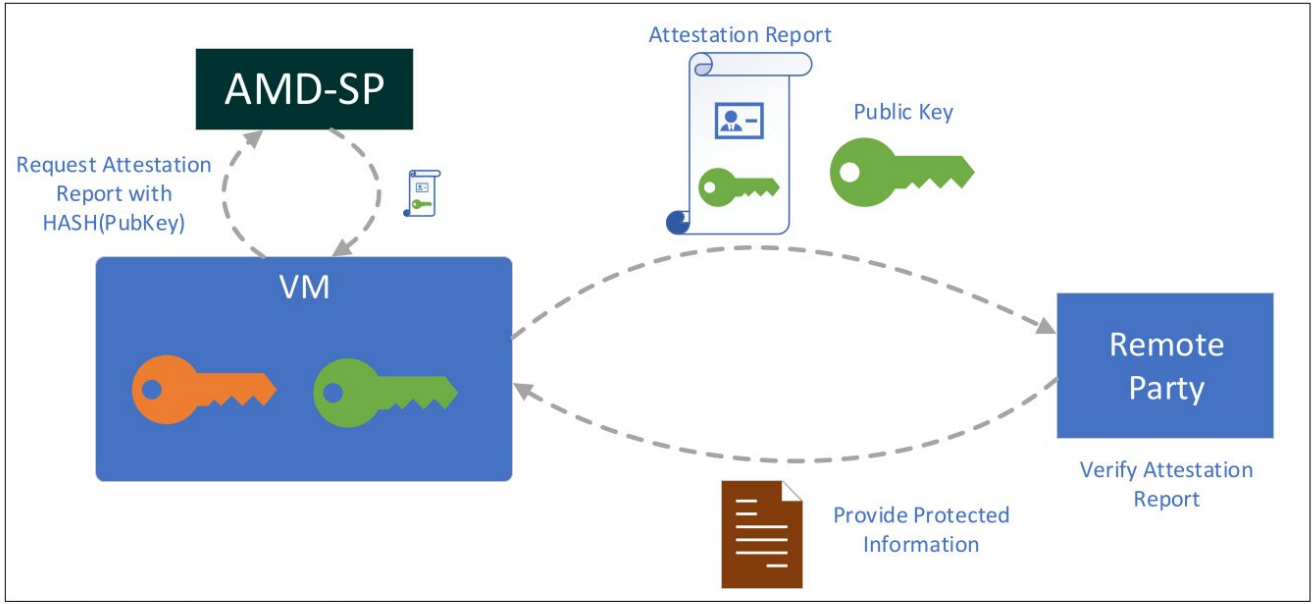
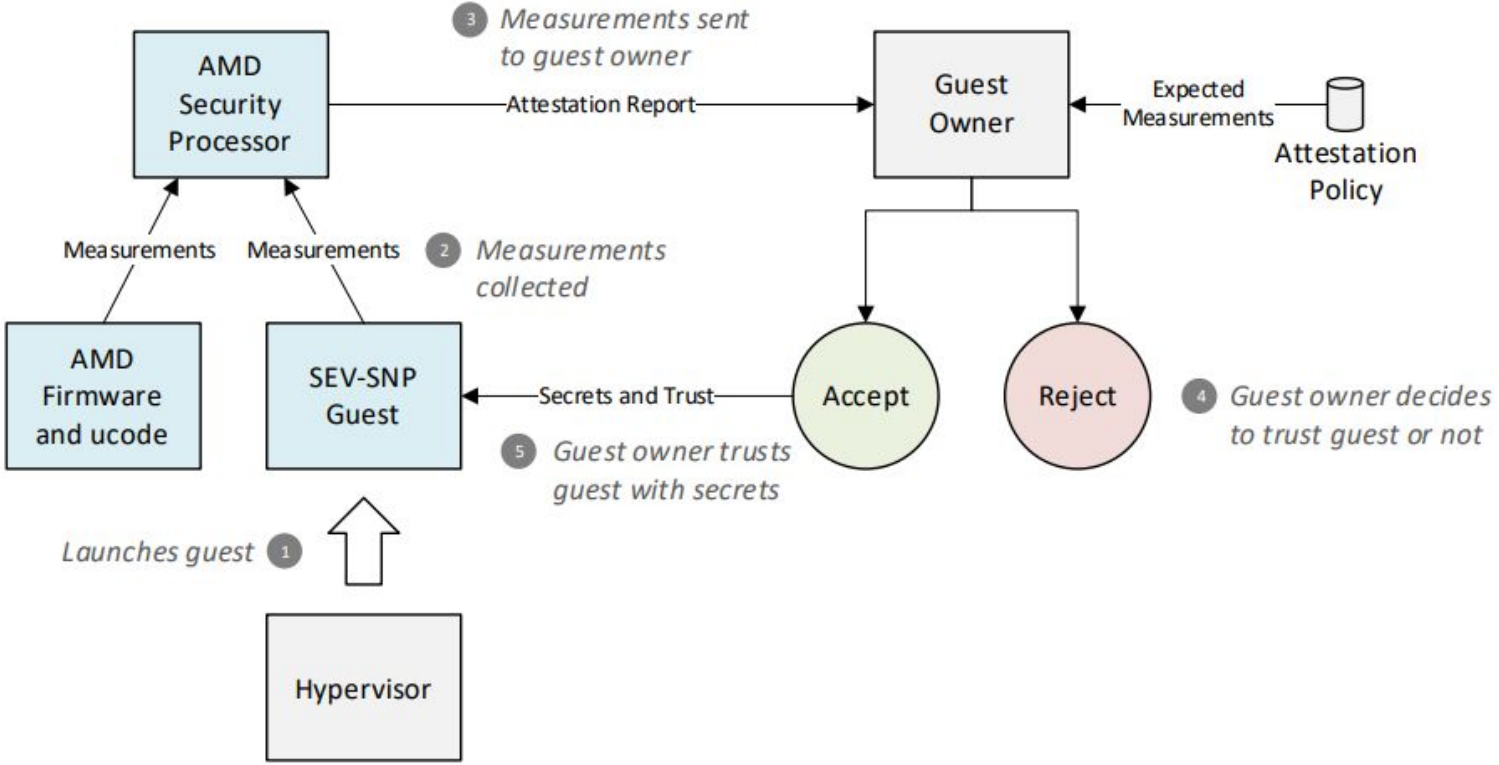


FIGURE 10: SEV-SNP ATTESTATION

Key Attestation



AMD Secure Processor

Key Management

MMIO Registers (Platform Management API, Guest Management API)

Hypervisor -> SEV Driver -> MMIO Registers

SUSE

AMD-SEV

VM Launch & Attestation

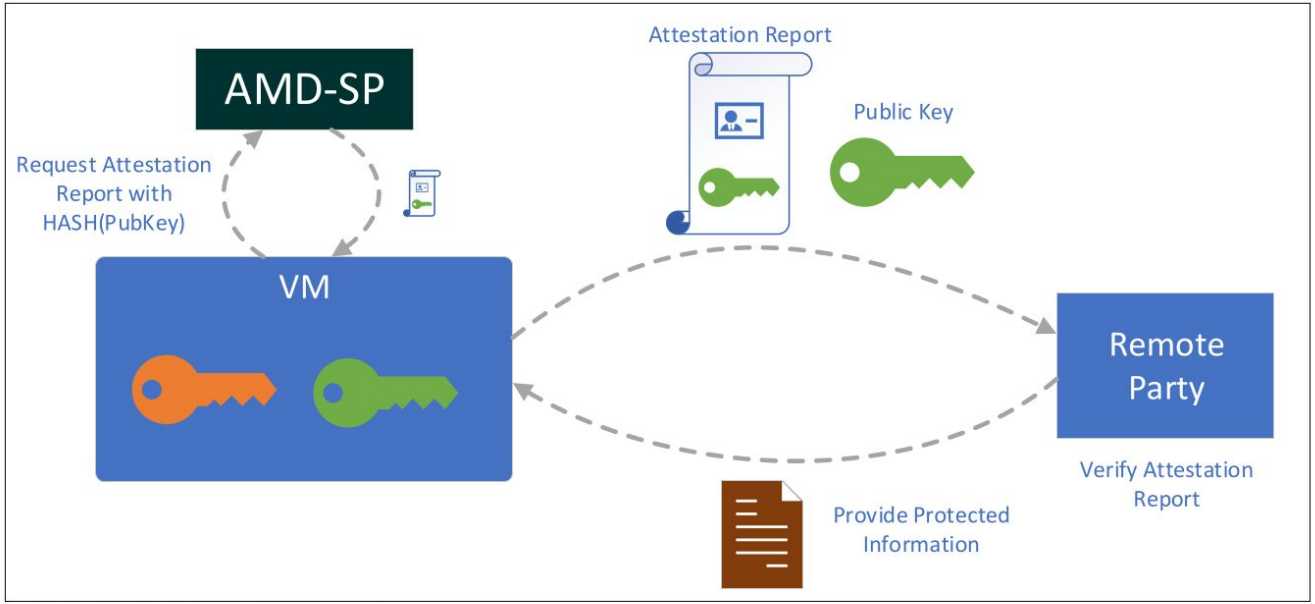
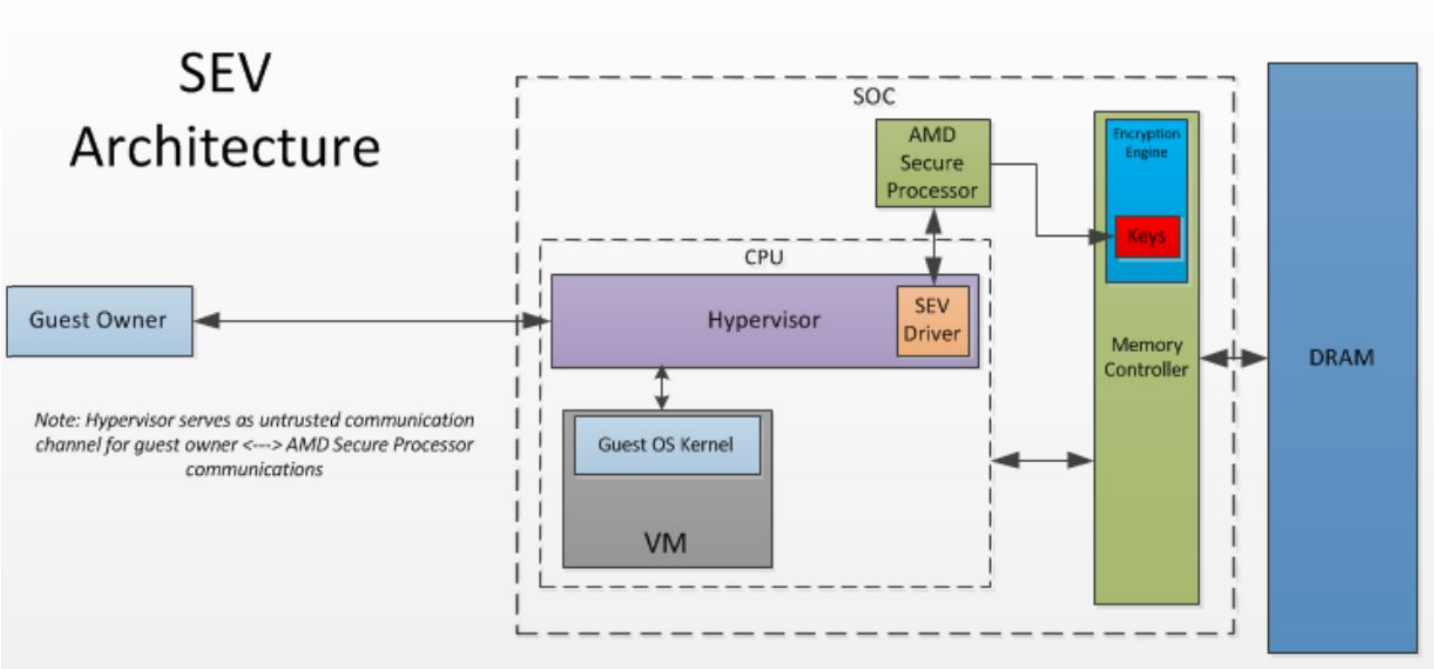


FIGURE 10: SEV-SNP ATTESTATION

Architecture



Key Management [6]

Table 1. Summary of Keys

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

Platform Authenticity (AMD, Cloud Provider)

Confidential Communication (Guest Owner <-> PSP)

Sevctl

```
PDH EP384 D256 f8f7389e5743fc00f9e88b219a7b9e80680148b2dd026e99237ca804b88bb763
 PEK EP384 E256 8af13aa247a4714433b3f5b223dbb5c2b3b7d6f2488a94f59343dc50fa597832
  OCA EP384 E256 d934d8fe6fd300d3d188822d781ea744b0f861a806f8580773e2e0893ad86c33
   CEK EP384 E256 1e999ed0c950b95df1cc738b52c8a2b54aa9c2f6ec7117df1303e98b975c2f19
    ASK R4096 R384 9fa6db577758411b576cfcdc3c2f0851e60f0c69ffdcb6301ad2f7864f91e829f62925888464d9d285612eb03b3fc63d
     ARK R4096 R384 b66e16ff0ebeed57f0c4dffa154872c4156f2e5b1c2f4e66e58fc37f1d2a60d31a342ecb6430a9b2510a970c75c37926

 = self signed,  = signs,  = invalid self sign,  = invalid signs
```

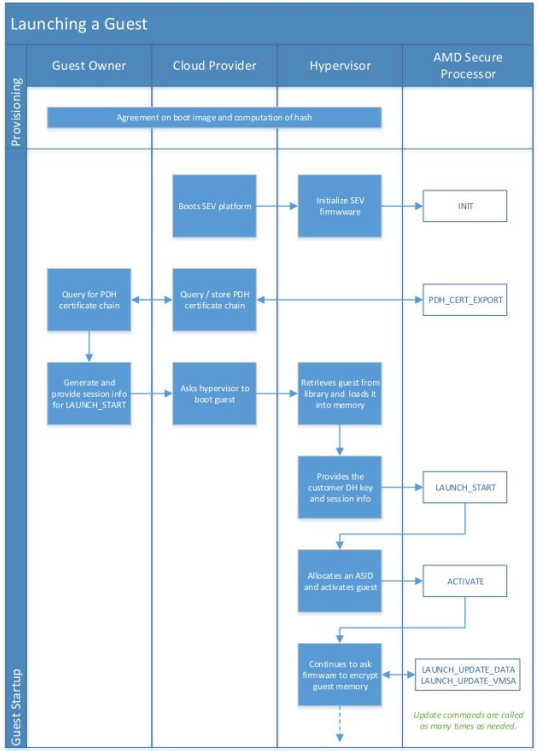
Certificate chain for device identity

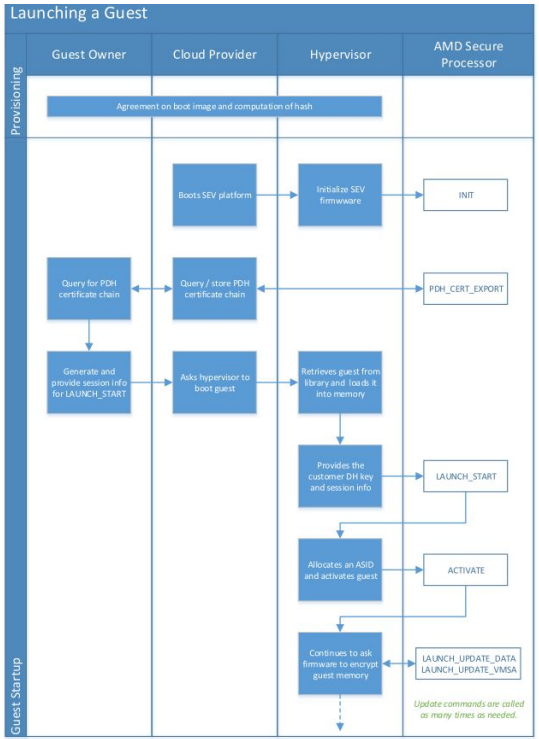
- **ARK -> ASK -> CEK -> PEK -> report (AMD)**

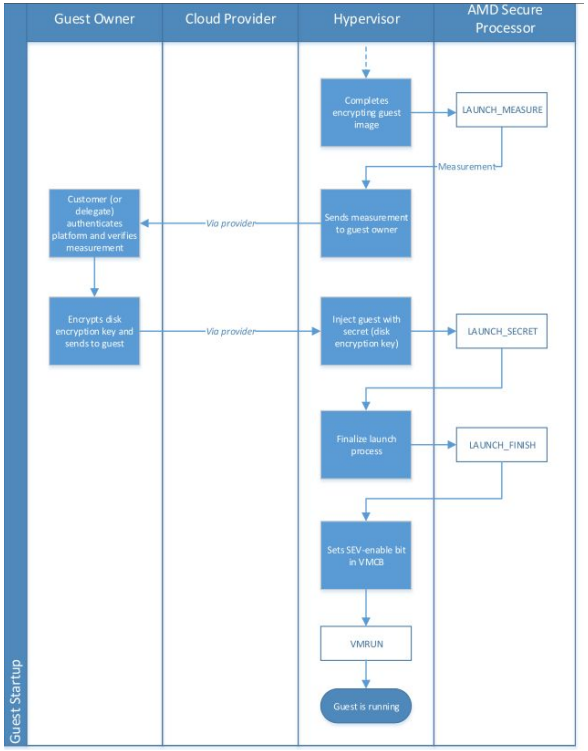
Certificate chain for platform owner identity:

- **OCA -> PEK -> report (Platform Owner)**

Launch a Guest







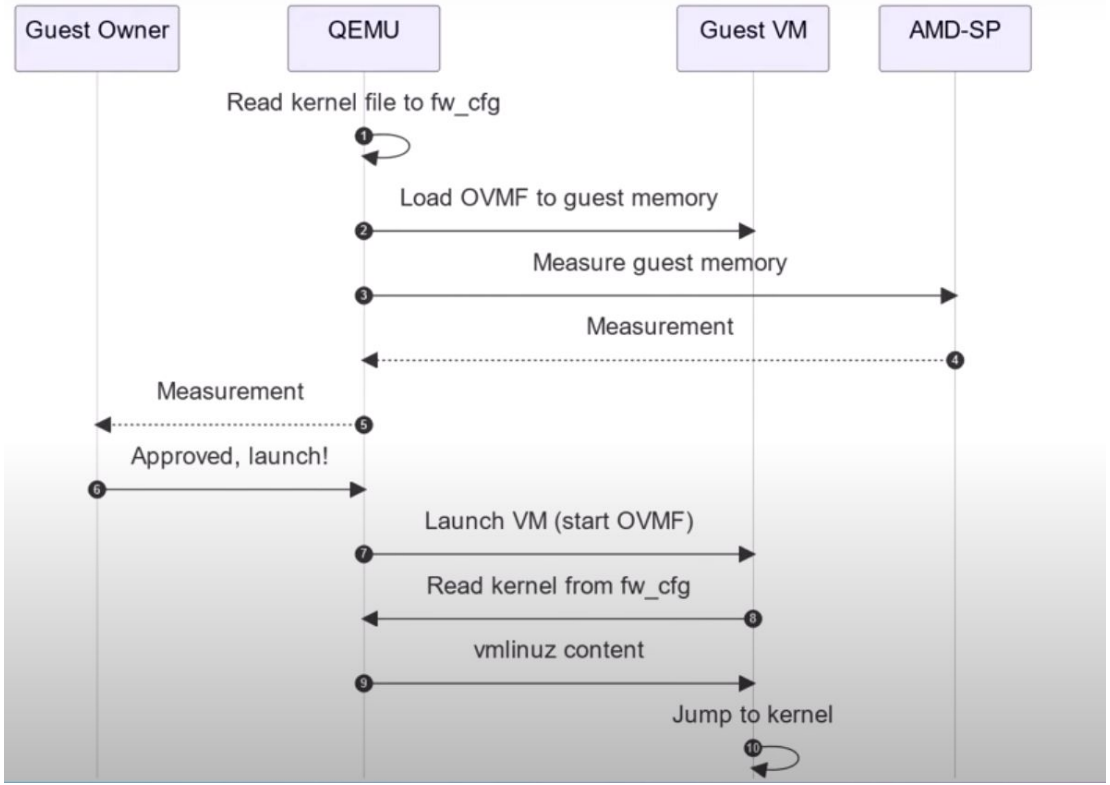
Launch a Guest

- PSP sends **PDH** and **PEK** to hypervisor, which sends chain to guest owner
- Guest owner validates **PDH** with chain of **PEK**, **CEK**, **ASK** and **ARK**, as well as **OCA**
- Guest owner generates ephermeral ECDH key (**GDH**)
- Guest owner derives shared Key Encryption Key (**KEK**) from private key of **GDH** and public key of **PDH**
- Guest owner generate ephemeral Transport Integrity Key (**TIK**) and Transport Encryption Key (**TEK**), encrypts both with **KEK**
- Guest owner sends public key of **GDH**, encrypted **TIK**/**TEK**, and launch policy to hypervisor
- Hypervisor calls PSP `LAUNCH_START` with these parameters (but can't decrypt the **TIK**/**TEK**)
- PSP derives **KEK** from its private key of **PDH** and the public key of **GDH**, uses **KEK** to decrypt **TIK**
- Hypervisor allocates ASID and calls PSP `ACTIVATE` to enable guest
- PSP generates memory encryption key (**VEK**) for this ASID (if the ASID is already in use, the PSP returns `ASID_OWNED` and won't activate the guest)
- Guest owner sends *clear text* kernel and initrd to hypervisor

Launch a Guest

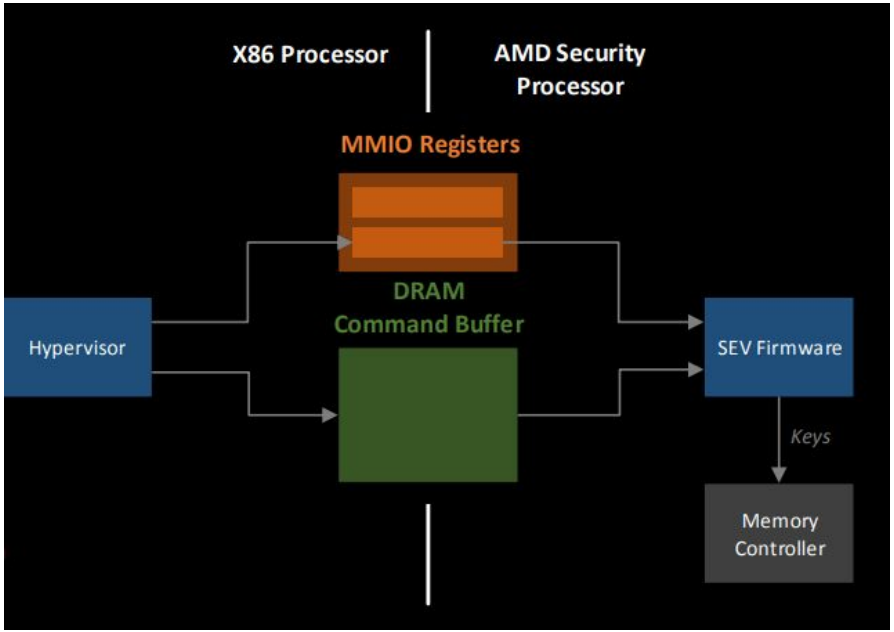
- Hypervisor calls PSP `LAUNCH_UPDATE_DATA` to add the kernel and initrd to the guest
- PSP hashes clear text of data and encrypts physical pages with **VEK** (*** IS THERE A TOCTOU? ***)
- Hypervisor calls PSP `LAUNCH_UPDATE_VMSA` to configure virtualization structures
- PSP hashes clear text VMSA structures and encrypts them with **VEK**
- Hypervisor calls PSP `LAUNCH_MEASURE`
- PSP generates liveliness nonce and computes HMAC of the nonce data, vmsa and policy using the **TIK** that only it and the guest owner know
- Hypervisor sends this measurement HMAC to the guest owner (it can't fake it since it doesn't know **TIK**)
- Guest owner validates HMAC, trusts that VM has been setup correctly
- (Hypervisor can't add new pages at this point since the `LAUNCH_MEASURE` moves the guest into `LSECRET` mode and disables the `LAUNCH_DATA` command)
- Guest owner sends up to 16 KB encrypted with **TEK** and HMAC'ed with **TIK** to the hypervisor. This can contain secrets like disk encryption keys so that the cloud provider can't see them (although the cloud provider was able to see the entire kernel and initrd, so they should not contain secrets).
- Hypervisor calls PSP `LAUNCH_SECRET` to add this data to the guest
- PSP validates HMAC with **TIK**, decrypts with **TEK** and re-encrypts with **VEK** into the guest's memory
- Hypervisor calls PSP `LAUNCH_FINISH`, which causes the PSP to forget all of the guest keys except for **VEK**
- Hypervisor invokes `VMRUN` to start the encrypted guest enclave

VM boot process with kernel



SEV Key Management

Communicates with x86 software
– Mailbox registers
– Shared memory buffer



SMD-SEV Support [1]

- Supported Operating Modes
  - Long mode
  - Legacy PAE mode

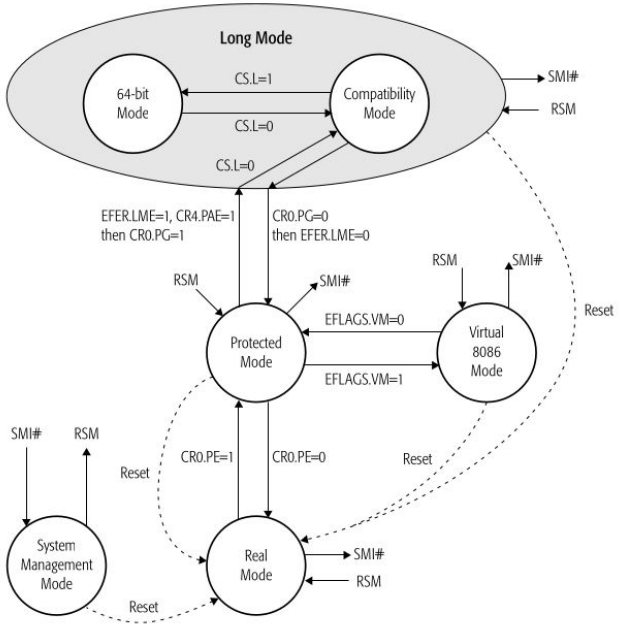


Figure 1-6. Operating Modes of the AMD64 Architecture

SUSE

AMD-SEV ES

VMM Communication Exception (#VC)

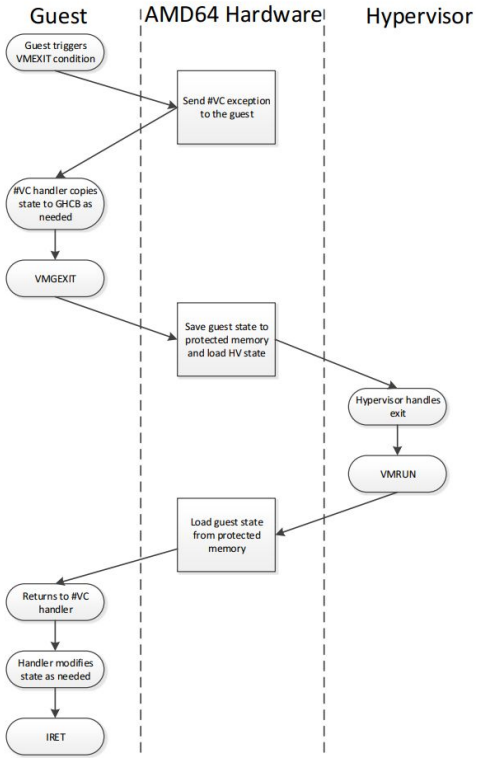


Figure 15-31. EXAMPLE #VC FLOW

VMCB

[12]

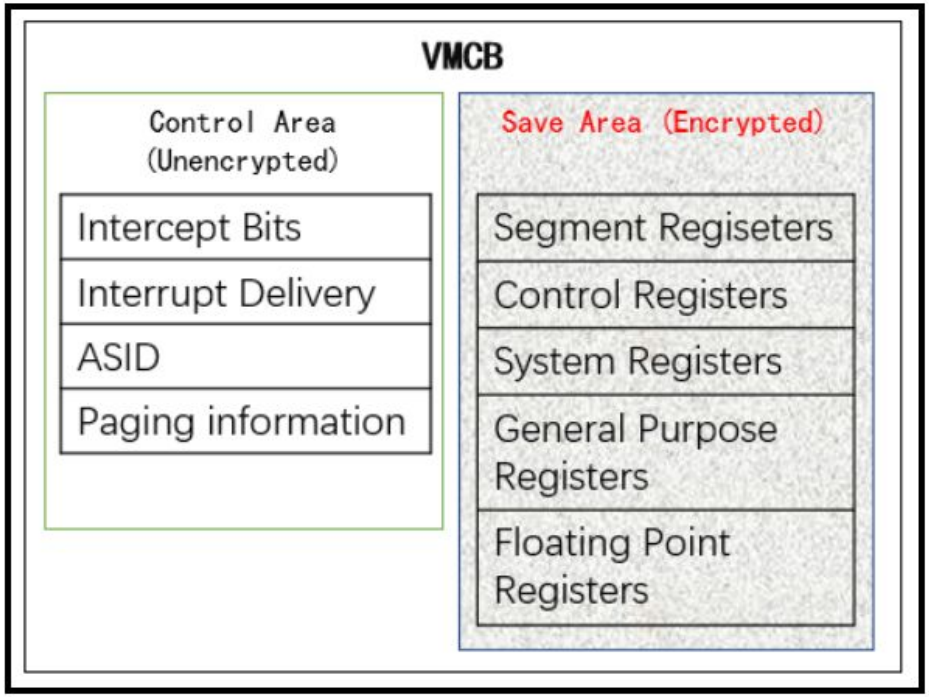


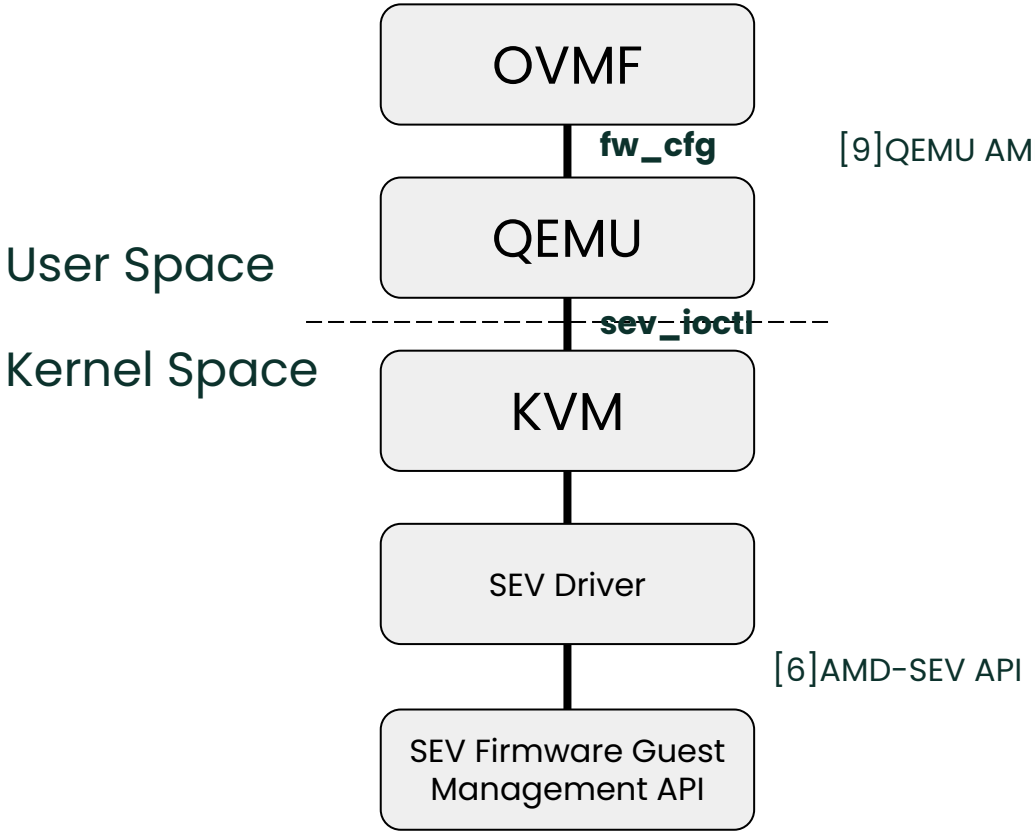
Figure 3. Virtual Machine Control Block (VMCB)

VMCB

SUSE

Implementation

AMD SEV



[9]QEMU AMD-SEV

[6]AMD-SEV API

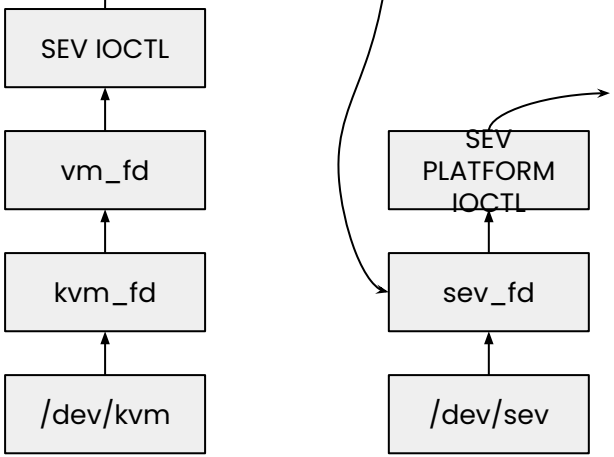
AMD SEV

```
struct kvm_sev_cmd {
        __u32 id;
        __u32 pad0;
        __u64 data;
        __u32 error;
        __u32 sev_fd;
};
```

```
/* Secure Encrypted Virtualization command */
enum sev_cmd_id {
        /* Guest initialization commands */
        KVM_SEV_INIT = 0,
        KVM_SEV_ES_INIT,
        /* Guest launch commands */
        KVM_SEV_LAUNCH_START,
        KVM_SEV_LAUNCH_UPDATE_DATA,
        KVM_SEV_LAUNCH_UPDATE_VMSA,
    …
        KVM_SEV_NR_MAX,
};
```

```
struct sev_issue_cmd {
        __u32 cmd;              /* In */
        __u64 data;             /* In */
        __u32 error;            /* Out */
} __packed;
```

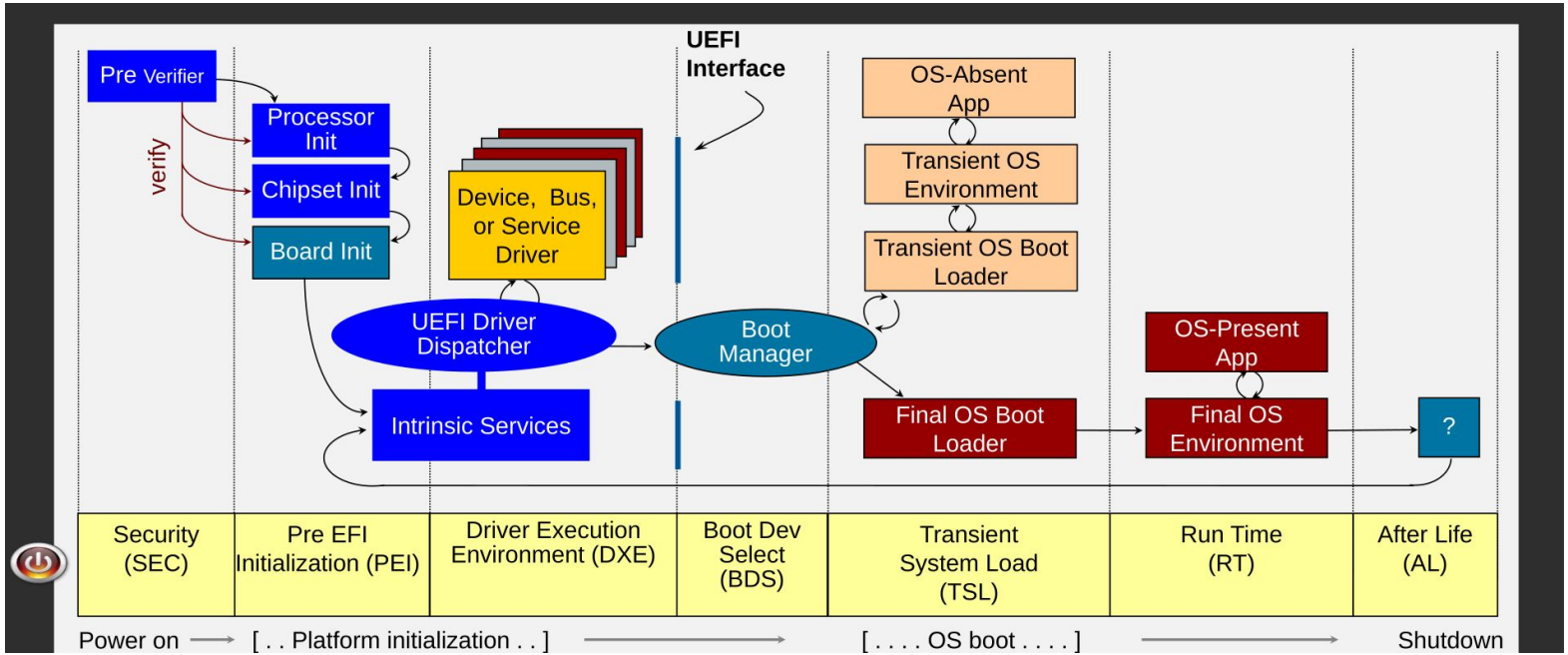
```
/**
 * SEV platform commands
 */
enum {
        SEV_FACTORY_RESET = 0,
        SEV_PLATFORM_STATUS,
        SEV_PEK_GEN,
        SEV_PEK_CSR,
        SEV_PDH_GEN,
        SEV_PDH_CERT_EXPORT,
        SEV_PEK_CERT_IMPORT,
        SEV_GET_ID,     /* This command is
deprecated, use SEV_GET_ID2 */
        SEV_GET_ID2,
        SNP_PLATFORM_STATUS,
        SNP_COMMIT,
        SNP_SET_CONFIG,
        SNP_VLEK_LOAD,

        SEV_MAX,
};
```



character device driver

SUSE

VM Launch & Attestation

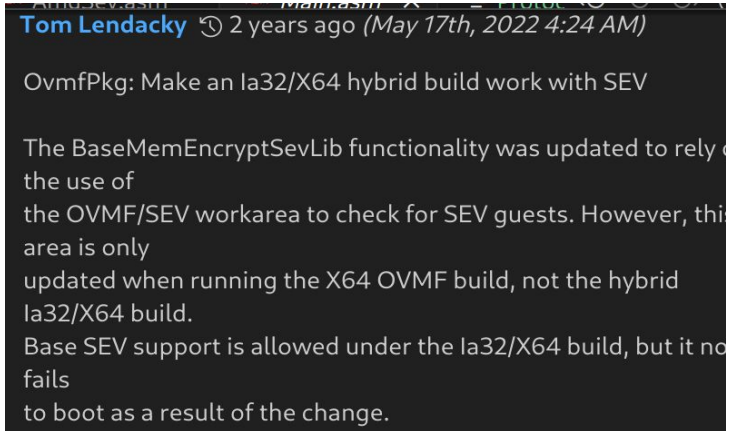


Ovmf ResetVector

/edk2/OvmfPkg/ResetVector/Main.asm

- Main routine of the pre-SEC code up through the jump into SEC

```
BITS    16
Main16:
   OneTimeCall EarlyInit16

   OneTimeCall TransitionFromReal16To32BitFlat
BITS    32
   mov     byte[WORK_AREA_GUEST_TYPE], 0

Main32:
OneTimeCall InitTdx
OneTimeCall Flat32SearchForBfvBase
OneTimeCall Flat32SearchForSecEntryPoint

%ifdef ARCH_IA32
   OneTimeCall CheckSevFeatures
   …
   jmp     rsi

%else
   OneTimeCall Transition32FlatTo64Flat

BITS    64
   …
jmp     rsi
```



Ovmf ResetVector

/edk2/OvmfPkg/ResetVector/Ia32/AmdSev.asm

- Provide the functions to check whether SEV and SEV-ES is enabled.

```
; Check if SEV memory encryption is enabled
;  MSR_0xC0010131 - Bit 0 (SEV enabled)
mov       ecx, SEV_STATUS_MSR
rdmsr
bt        eax, 0
jnc       NoSev
```

```
; Check if SEV-ES is enabled
;  MSR_0xC0010131 - Bit 1 (SEV-ES enabled)
mov       ecx, SEV_STATUS_MSR
rdmsr
bt        eax, 1
jnc       GetSevEncBit
```

/edk2/OvmfPkg/Sec/AmdSev.c

- File defines the Sec routines for the AMD SEV

```
//
// Check MSR_0xC0010131 Bit 2 (Sev-Snp Enabled)
//
if (Msr.Bits.SevSnpBit) {
  return TRUE;
}
```

Ovmf - SEC

/edk2/OvmfPkg/Library/QemuFwCfgLib/QemuFwCfgSec.c

- BOOLEAN EFIAPI QemuFwCfgIsAvailable (VOID)
  - QemuFwCfgSelectItem (QemuFwCfgItemSignature); - Signature = QemuFwCfgRead32 ();
  - QemuFwCfgSelectItem (QemuFwCfgItemInterfaceVersion) - Revision = QemuFwCfgRead32 ();

- BOOLEAN InternalQemuFwCfgIsAvailable (VOID)
- BOOLEAN sInternalQemuFwCfgDmaIsAvailable (VOID)
- BOOLEAN InternalQemuFwCfgDmaIsAvailable (VOID)
- VOID InternalQemuFwCfgDmaBytes

Ovmf - PEI

EDK2/OvmfPkg/Library/QemuFwCfgLib/QemuFwCfgPei.c

- STATIC BOOLEAN QemuFwCfgIsCcGuest (VOID)
- BOOLEAN EFIAPI QemuFwCfgIsAvailable
    - InternalQemuFwCfgIsAvailable ()
- STATIC VOID QemuFwCfgProbe
    - UINT32 Signature;
    - UINT32 Revision;
- STATIC EFI_HOB_PLATFORM_INFO
    *QemuFwCfgGetPlatformInfo
    - EFI_HOB_PLATFORM_INFO *PlatformInfoHob;
    - EFI_HOB_GUID_TYPE   *GuidHob;
- RETURN_STATUS EFIAPI QemuFwCfgInitialize
- BOOLEAN InternalQemuFwCfgIsAvailable
- BOOLEAN InternalQemuFwCfgDmaIsAvailable
- VOID InternalQemuFwCfgDmaBytes

Ovmf - DXE

EDK2/OvmfPkg/Library/QemuFwCfgLib/QemuFwCfgDxe.c

- UINTN EFIAPI QemuGetFwCfgSelectorAddress
- UINTN EFIAPI QemuGetFwCfgDataAddress
- UINTN EFIAPI QemuGetFwCfgDmaAddress
- RETURN_STATUS EFIAPI QemuFwCfgInitialize

```
- EFI_STATUS             Status;
- FDT_CLIENT_PROTOCOL    *FdtClient;
- CONST UINT64           *Reg;
- UINT32                 RegSize;
- UINTN                  AddressCells, SizeCell
- UINT64                 FwCfgSelectorAddress;
- UINT64                 FwCfgSelectorSize;
- UINT64                 FwCfgDataAddress;
- UINT64                 FwCfgDataSize;
- UINT64                 FwCfgDmaAddress;
- UINT64                 FwCfgDmaSize;
- QEMU_FW_CFG_RESOURCE   *FwCfgResource;
```

QEMU SEV, SEV-ES, SEV-SNP Enabled

/qemu/target/i386/sev.c

```
bool
sev_enabled(void)
{
    ConfidentialGuestSupport *cgs = MACHINE(qdev_get_machine())->cgs;

    return !!object_dynamic_cast(OBJECT(cgs), TYPE_SEV_COMMON);
}
```

```
bool
sev_es_enabled(void)
{
    ConfidentialGuestSupport *cgs = MACHINE(qdev_get_machine())->cgs;

    return sev_snp_enabled() ||
           (sev_enabled() && SEV_GUEST(cgs)->policy & SEV_POLICY_ES);
}
```

```
bool
sev_snp_enabled(void)
{
    ConfidentialGuestSupport *cgs = MACHINE(qdev_get_machine())->cgs;

    return !!object_dynamic_cast(OBJECT(cgs), TYPE_SEV_SNP_GUEST);
}
```

QEMU SEV I/O Control

/qemu/target/i386/sev.c

```c
static int
sev_ioctl(int fd, int cmd, void *data, int *error)
{
    int r;
    struct kvm_sev_cmd input;

    memset(&input, 0x0, sizeof(input));

    input.id = cmd;
    input.sev_fd = fd;
    input.data = (uintptr_t)data;

    r = kvm_vm_ioctl(kvm_state, KVM_MEMORY_ENCRYPT_OP, &input);

    if (error) {
        *error = input.error;
    }

    return r;
}
```

/qemu/linux-headers/asm-x86/kvm.h

```c
struct kvm_sev_cmd {
    __u32 id;
    __u32 pad0;
    __u64 data;
    __u32 error;
    __u32 sev_fd;
};
```

/qemu/accel/kvm/kvm-all.c

```c
int kvm_vm_ioctl(KVMState *s, int type, ...)
{
    int ret;
    void *arg;
    va_list ap;

    va_start(ap, type);
    arg = va_arg(ap, void *);
    va_end(ap);

    trace_kvm_vm_ioctl(type, arg);
    accel_ioctl_begin();
    ret = ioctl(s->vmfd, type, arg);
    accel_ioctl_end();
    if (ret == -1) {
        ret = -errno;
    }
    return ret;
}
```

QEMU SEV Platform I/O Control

/qemu/target/i386/sev.c

```c
static int
sev_platform_ioctl(int fd, int cmd, void *data, int *error)
{
    int r;
    struct sev_issue_cmd arg;

    arg.cmd = cmd;
    arg.data = (unsigned long)data;
    r = ioctl(fd, SEV_ISSUE_CMD, &arg);
    if (error) {
        *error = arg.error;
    }

    return r;
}
```

/qemu/linux-headers/linux/psp-sev.h

```c
/**
 * struct sev_issue_cmd - SEV ioctl parameters
 *
 * @cmd: SEV commands to execute
 * @opaque: pointer to the command structure
 * @error: SEV FW return code on failure
 */
struct sev_issue_cmd {
    __u32 cmd;              /* In */
    __u64 data;             /* In */
    __u32 error;                /* Out */
} __attribute__((packed));
```

/usr/include/sys/ioctl.h

```c
/* Perform the I/O control operation specified by REQUEST on FD.
   One argument may follow; its presence and type depend on REQUEST.
   Return value depends on REQUEST.  Usually -1 indicates error.  */
#ifndef __USE_TIME_BITS64
extern int ioctl (int __fd, unsigned long int __request, ...) __THROW;
#else
# ifdef __REDIRECT
extern int __REDIRECT_NTH (ioctl, (int __fd, unsigned long int __request, ...),
                           __ioctl_time64);
# else
extern int __ioctl_time64 (int __fd, unsigned long int __request, ...) __THROW;
#  define ioctl __ioctl_time64
# endif
#endif
```

SEV-SNP Implementation in OVMF

KVM

- 3d4aeaad8bb8 KVM: SVM: Report Nested Paging support to userspace
- 5bd2edc341d1 KVM: SVM: Implement MMU helper functions for Nested Nested Paging
- 709ddebf81cb KVM: SVM: add support for Nested Paging
- 6c7dac72d5c7 KVM: SVM: add module parameter to disable Nested Paging
- e3da3acdb32c KVM: SVM: add detection of Nested Paging feature
- c63cf135cc99 KVM: SEV: Add support to handle RMP nested page faults
- e3cdaab5ff02 KVM: x86: SVM: fix nested PAUSE filtering when L0 intercepts PAUSE
- 28f091bc2f8c KVM: MMU: shadow nested paging does not have PKU
- 6fec21449a62 KVM: x86: use correct page table format to check nested page table reserved bits
- 54987b7afa90 KVM: x86: propagate exception from permission checks on the nested page fault
- e2358851efbc KVM: SVM: comment nested paging and virtualization module parameters
- 3d4aeaad8bb8 KVM: SVM: Report Nested Paging support to userspace
- 5bd2edc341d1 KVM: SVM: Implement MMU helper functions for Nested Nested Paging

```
tools/testing/selftests/kvm/lib/x86_64/svm.c:#define SEV_DEV_PATH "/dev/sev"
tools/testing/selftests/kvm/lib/x86_64/svm.c: *   The opened file descriptor of /dev/sev.
```

KVM

d8aa7eea78a1 x86/mm: Add Secure Encrypted Virtualization (SEV) support

```
commit d8aa7eea78a1401cce39b3bb61ead0150044a3df
Author: Tom Lendacky <thomas.lendacky@amd.com>
Date:   Fri Oct 20 09:30:44 2017 -0500

    x86/mm: Add Secure Encrypted Virtualization (SEV) support

    Provide support for Secure Encrypted Virtualization (SEV). This initial
    support defines a flag that is used by the kernel to determine if it is
    running with SEV active.
```

```
+/*
+ * SME and SEV are very similar but they are not the same, so there are
+ * times that the kernel will need to distinguish between SME and SEV. The
+ * sme_active() and sev_active() functions are used for this.  When a
+ * distinction isn't needed, the mem_encrypt_active() function can be used.
+ *
+ * The trampoline code is a good example for this requirement.  Before
+ * paging is activated, SME will access all memory as decrypted, but SEV
+ * will access all memory as encrypted.  So, when APs are being brought
+ * up under SME the trampoline area cannot be encrypted, whereas under SEV
+ * the trampoline area must be encrypted.
+ */
+bool sme_active(void)
+{
+       return sme_me_mask && !sev_enabled;
+}
+EXPORT_SYMBOL_GPL(sme_active);
+
+bool sev_active(void)
+{
+       return sme_me_mask && sev_enabled;
+}
+EXPORT_SYMBOL_GPL(sev_active);
```

KVM

916391a2d1dc KVM: SVM: Add support for SEV-ES capability in KVM

```
commit 916391a2d1dc225bfb68624352b1495ec529444e
Author: Tom Lendacky <thomas.lendacky@amd.com>
Date:   Thu Dec 10 11:09:38 2020 -0600

    KVM: SVM: Add support for SEV-ES capability in KVM

    Add support to KVM for determining if a system is capable of supporting
    SEV-ES as well as determining if a guest is an SEV-ES guest.

    Signed-off-by: Tom Lendacky <thomas.lendacky@amd.com>
    Message-Id: <e66792323982c822350e40c7a1cf67ea2978a70b.1607620209.git.thom
    Signed-off-by: Paolo Bonzini <pbonzini@redhat.com>
```

KVM

cbd3d4f7c4e5 x86/sev: Check SEV-SNP features support

```
commit cbd3d4f7c4e5a93edae68e5142a269368fde77d6
Author: Brijesh Singh <brijesh.singh@amd.com>
Date:   Wed Feb 9 12:10:06 2022 -0600

    x86/sev: Check SEV-SNP features support

    Version 2 of the GHCB specification added the advertisement of features
    that are supported by the hypervisor. If the hypervisor supports SEV-SNP
    then it must set the SEV-SNP features bit to indicate that the base
    functionality is supported.

    Check that feature bit while establishing the GHCB; if failed, terminate
    the guest.
```

KVM

200664d5237f crypto: ccp: Add Secure Encrypted Virtualization (SEV) command support

- Support for SEV API Spec [link]

```
commit 200664d5237f3f8cd2a2f9f5c5dea08502336bd1
Author: Brijesh Singh <brijesh.singh@amd.com>
Date:   Mon Dec 4 10:57:28 2017 -0600

    crypto: ccp: Add Secure Encrypted Virtualization (SEV) command support

    AMD's new Secure Encrypted Virtualization (SEV) feature allows the
    memory contents of virtual machines to be transparently encrypted with a
    key unique to the VM. The programming and management of the encryption
    keys are handled by the AMD Secure Processor (AMD-SP) which exposes the
    commands for these tasks. The complete spec is available at:

    http://support.amd.com/TechDocs/55766_SEV-KM%20API_Specification.pdf

    Extend the AMD-SP driver to provide the following support:

     - an in-kernel API to communicate with the SEV firmware. The API can be
       used by the hypervisor to create encryption context for a SEV guest.

     - a userspace IOCTL to manage the platform certificates.
```

```
+static int sev_cmd_buffer_len(int cmd)
+{
+       switch (cmd) {
+       case SEV_CMD_INIT:                      return sizeof(struct sev_data_init);
+       case SEV_CMD_PLATFORM_STATUS:           return sizeof(struct sev_user_data_status);
+       case SEV_CMD_PEK_CSR:                   return sizeof(struct sev_data_pek_csr);
+       case SEV_CMD_PEK_CERT_IMPORT:           return sizeof(struct sev_data_pek_cert_import);
+       case SEV_CMD_PDH_CERT_EXPORT:           return sizeof(struct sev_data_pdh_cert_export);
+       case SEV_CMD_LAUNCH_START:              return sizeof(struct sev_data_launch_start);
+       case SEV_CMD_LAUNCH_UPDATE_DATA:        return sizeof(struct sev_data_launch_update_data);
+       case SEV_CMD_LAUNCH_UPDATE_VMSA:        return sizeof(struct sev_data_launch_update_vmsa);
+       case SEV_CMD_LAUNCH_FINISH:             return sizeof(struct sev_data_launch_finish);
+       case SEV_CMD_LAUNCH_MEASURE:            return sizeof(struct sev_data_launch_measure);
+       case SEV_CMD_ACTIVATE:                  return sizeof(struct sev_data_activate);
+       case SEV_CMD_DEACTIVATE:                return sizeof(struct sev_data_deactivate);
+       case SEV_CMD_DECOMMISSION:              return sizeof(struct sev_data_decommission);
+       case SEV_CMD_GUEST_STATUS:              return sizeof(struct sev_data_guest_status);
+       case SEV_CMD_DBG_DECRYPT:               return sizeof(struct sev_data_dbg);
+       case SEV_CMD_DBG_ENCRYPT:               return sizeof(struct sev_data_dbg);
+       case SEV_CMD_SEND_START:                return sizeof(struct sev_data_send_start);
+       case SEV_CMD_SEND_UPDATE_DATA:          return sizeof(struct sev_data_send_update_data);
+       case SEV_CMD_SEND_UPDATE_VMSA:          return sizeof(struct sev_data_send_update_vmsa);
+       case SEV_CMD_SEND_FINISH:               return sizeof(struct sev_data_send_finish);
+       case SEV_CMD_RECEIVE_START:             return sizeof(struct sev_data_receive_start);
+       case SEV_CMD_RECEIVE_FINISH:            return sizeof(struct sev_data_receive_finish);
+       case SEV_CMD_RECEIVE_UPDATE_DATA:       return sizeof(struct sev_data_receive_update_data);
+       case SEV_CMD_RECEIVE_UPDATE_VMSA:       return sizeof(struct sev_data_receive_update_vmsa);
+       case SEV_CMD_LAUNCH_UPDATE_SECRET:      return sizeof(struct sev_data_launch_secret);
+       default:                                return 0;
```

KVM

Implementation SEV API

```
9f5b5b950aa9 KVM: SVM: Add support for SEV LAUNCH_SECRET command
7d1594f5d94b KVM: SVM: Add support for SEV DEBUG_ENCRYPT command
24f41fb23a39 KVM: SVM: Add support for SEV DEBUG_DECRYPT command
255d9e75e254 KVM: SVM: Add support for SEV GUEST_STATUS command
5bdb0e2fa45e KVM: SVM: Add support for SEV LAUNCH_FINISH command
0d0736f76347 KVM: SVM: Add support for KVM_SEV_LAUNCH_MEASURE command
89c505809052 KVM: SVM: Add support for KVM_SEV_LAUNCH_UPDATE_DATA command
59414c989220 KVM: SVM: Add support for KVM_SEV_LAUNCH_START command
70cd94e60c73 KVM: SVM: VMRUN should use associated ASID when SEV is enabled
1654efcbc431 KVM: SVM: Add KVM_SEV_INIT command
dc48bae01e5a KVM: Define SEV key management command id
ed3cd233f807 KVM: SVM: Reserve ASID range for SEV guest
5dd0a57cf38e KVM: X86: Add CONFIG_KVM_AMD_SEV
```

KVM Nested Paging

KVM 6.11 Support sev-snp

ab978c62e72d Merge branch 'kvm-6.11-sev-snp' into HEAD

```
commit ab978c62e72d6b2d41842210e0cc435d9ed0dadb
Merge: f9d1b541d057 b2ec042347fd
Author: Paolo Bonzini <pbonzini@redhat.com>
Date:   Mon Jun 3 13:19:46 2024 -0400

    Merge branch 'kvm-6.11-sev-snp' into HEAD

    Pull base x86 KVM support for running SEV-SNP guests from Michael Roth:

    * add some basic infrastructure and introduces a new KVM_X86_SNP_VM
      vm_type to handle differences versus the existing KVM_X86_SEV_VM and
      KVM_X86_SEV_ES_VM types.

    * implement the KVM API to handle the creation of a cryptographic
      launch context, encrypt/measure the initial image into guest memory,
      and finalize it before launching it.

    * implement handling for various guest-generated events such as page
      state changes, onlining of additional vCPUs, etc.

    * implement the gmem/mmu hooks needed to prepare gmem-allocated pages
      before mapping them into guest private memory ranges as well as
      cleaning them up prior to returning them to the host for use as
      normal memory. Because those cleanup hooks supplant certain
      activities like issuing WBINVDs during KVM MMU invalidations, avoid
      duplicating that work to avoid unecessary overhead.

    This merge leaves out support support for attestation guest requests
    and for loading the signing keys to be used for attestation requests.
```

Linux Kernel SEV, SEV-ES, SEV-SNP Enabled

/linux/arch/x86/kvm/svm/sev.c

```
/* enable/disable SEV support */
static bool sev_enabled = true;
module_param_named(sev, sev_enabled, bool, 0444);

/* enable/disable SEV-ES support */
static bool sev_es_enabled = true;
module_param_named(sev_es, sev_es_enabled, bool, 0444);

/* enable/disable SEV-SNP support */
static bool sev_snp_enabled = true;
module_param_named(sev_snp, sev_snp_enabled, bool, 0444);
```

Reference

[1]AMD64 Architecture Programmer's Manual Volume 2: System Programming
https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24593.pdf
[2] AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions
https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24594.pdf
[3] AMD-V™ Nested Paging
https://www.cse.iitd.ac.in/~sbansal/csl862-virt/2010/readings/NPT-WP-1%201-final-TM.pdf
[4]Accelerating Two-Dimensional Page Walks for Virtualized Systems
https://pages.cs.wisc.edu/~remzi/Classes/838/Spring2013/Papers/p26-bhargava.pdf
[5]Memory virtualization: shadow page & nest page
https://blog.csdn.net/hit_shaoqi/article/details/121887459
[6]AMD-SEV API
https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/programmer-references/55766_SEV-KM_API_Specification.pdf

Reference

[7]AMD Memory Encryption
https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf
[8]AMD SEV基本原理
https://blog.csdn.net/huang987246510/article/details/135487665
[9]QEMU - AMD SEV
https://www.qemu.org/docs/master/system/i386/amd-memory-encryption.html
[10]Linux - KVM
https://www.kernel.org/doc/html/v5.7/virt/kvm/index.html
[11] AMD SEV-SNP White Paper
https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf
[12] AMD SEV in ThinkSystem
https://lenovopress.lenovo.com/lp1545-using-amd-secure-encrypted-virtualization-encrypted-state-sev-es
[13] AMD SEV-SNP Key Attestation
https://www.amd.com/content/dam/amd/en/documents/developer/lss-snp-attestation.pdf

[14]AMD Virtualization Memory Encryption Technology
https://www.linux-kvm.org/images/7/74/02x08A-Thomas_Lendacky-AMDs_Virtualizatoin_Memory_Encryption_Technology.pdf

SUSE

Thank You

SUSE

AMD SEV-SNP

Integrity Threats

[11]

THREAT	DESIRED SECURITY PROPERTY	SEV-SNP ENFORCEMENT MECHANISM
REPLAY PROTECTION	Only the owner of a memory page can write that page	Reverse Map Table (RMP)
DATA CORRUPTION	Only the owner of a memory page can write that page	Reverse Map Table (RMP)
MEMORY ALIASING	Every physical memory page can map only to a single guest page at one time	Reverse Map Table (RMP)
MEMORY RE-MAPPING	Every guest page can map only to a single physical memory page at one time	Page Validation

Reverse Map Table (RMP)

[11]

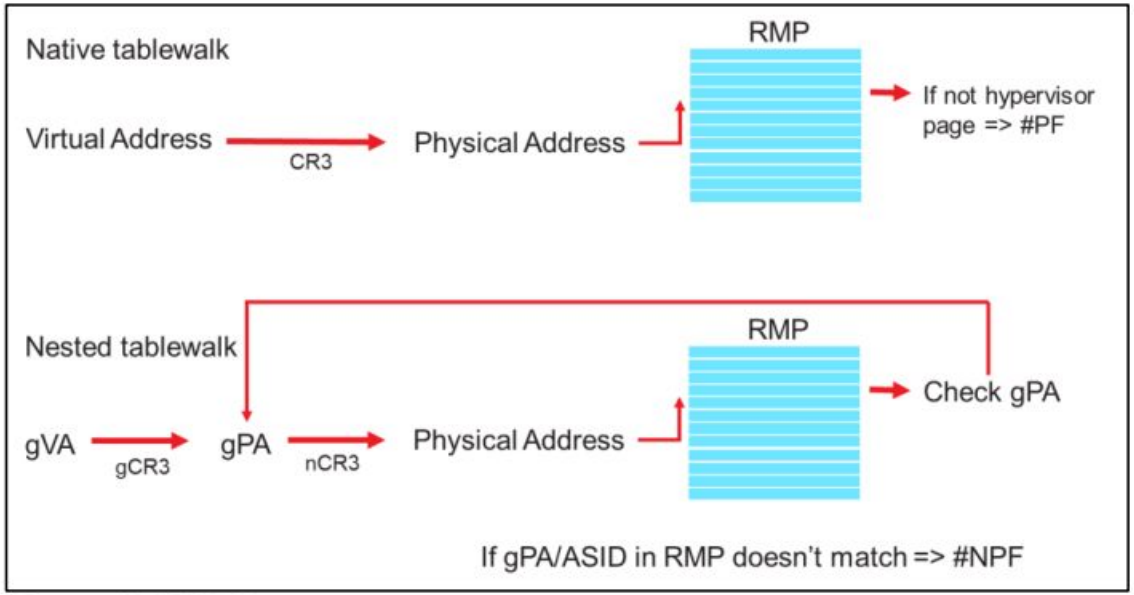


FIGURE 3: RMP CHECKS

Page Validation

[11]

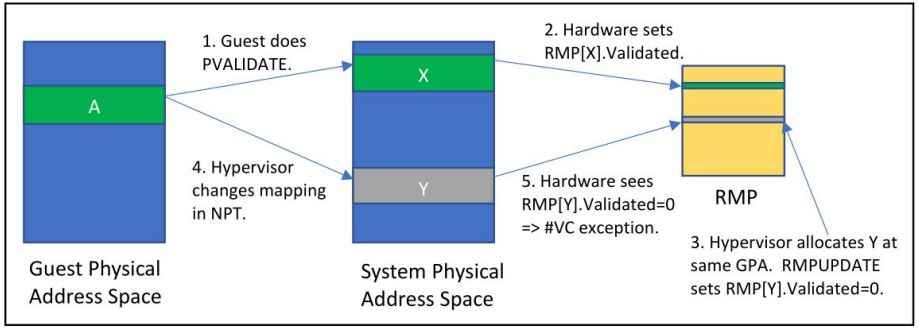


FIGURE 5: PAGE RE-MAPPING ATTACK

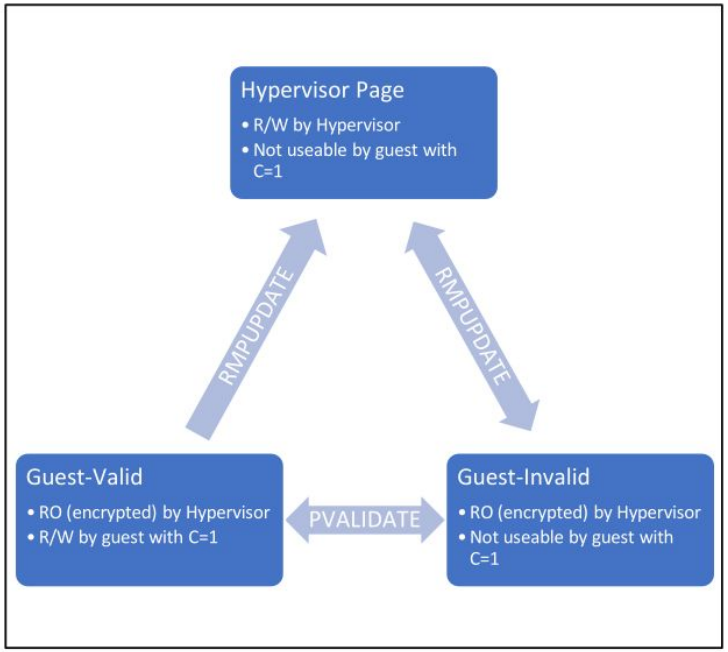


FIGURE 4: BASIC PAGE STATES

SEV-SNP Page States

[11]

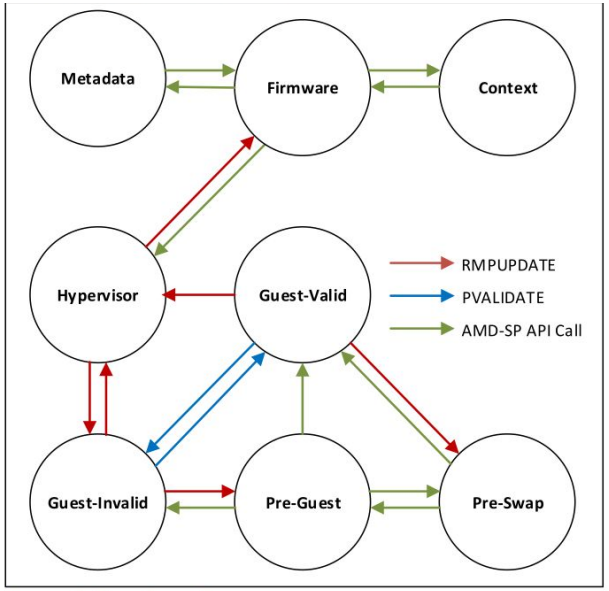


FIGURE 6: PAGE STATE TRANSITIONS

[1]

STATE	DESCRIPTION	NOTES
HYPERVISOR	Default state for otherwise unassigned memory	Used for hypervisor memory, non-SNP-VM memory, and shared (C=0) memory
GUEST-INVALID	Page is assigned to a guest but not ready to be used	Not useable by SEV-SNP VMs until validation has occurred
GUEST-VALID	Page is assigned to a guest and useable	Page may be used as private (C=1) memory by the assigned SEV-SNP VM
PRE-GUEST	Page is Immutable and not validated	Used when initially launching SEV-SNP VMs
PRE-SWAP	Page is Immutable and validated	Used when swapping guest pages to disk
FIRMWARE	Page is Immutable and reserved for AMD-SP use	Typically used as transitory state until AMD-SP has configured the page
METADATA	Page is Immutable and used for metadata	Metadata is used when swapping guest pages to disk
CONTEXT	Page is Immutable and used for context information	Context pages are used by the AMD-SP to identify individual VMs and hold per-VM

Virtual Machine Privilege Levels (VMPL)

- VMPL0 being the highest privilege level and VMPL3 the least privileged.

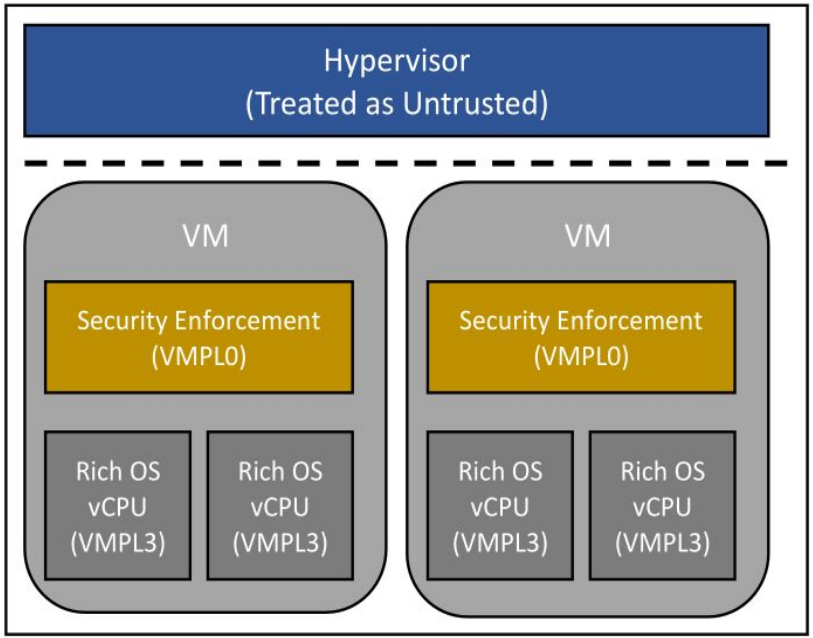


FIGURE 7: VMPLS

Interrupt/Exception Protection

Two optional modes:

- Restricted Injection
- Alternate Injection

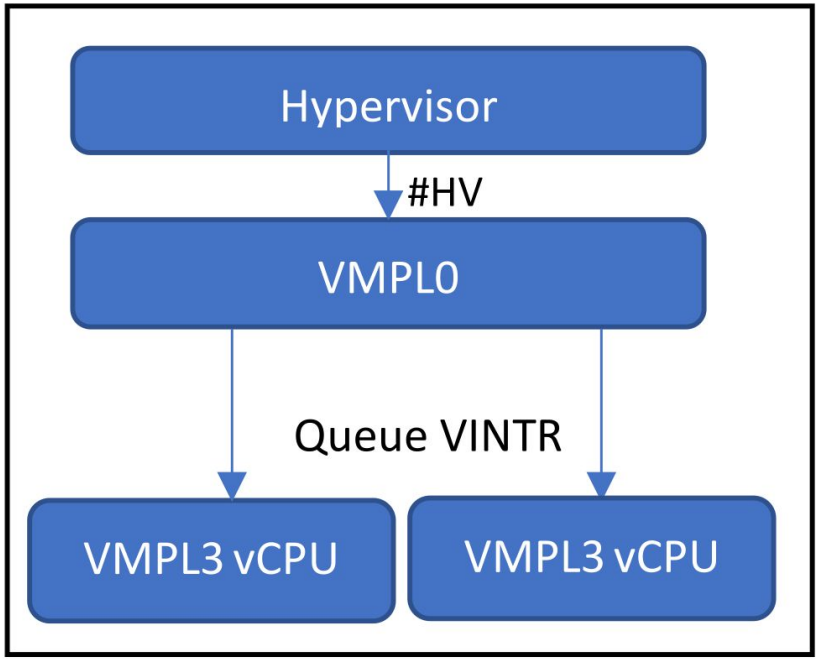


FIGURE 9: VMPL INTERRUPT HANDLING

VM Launch & Attestation

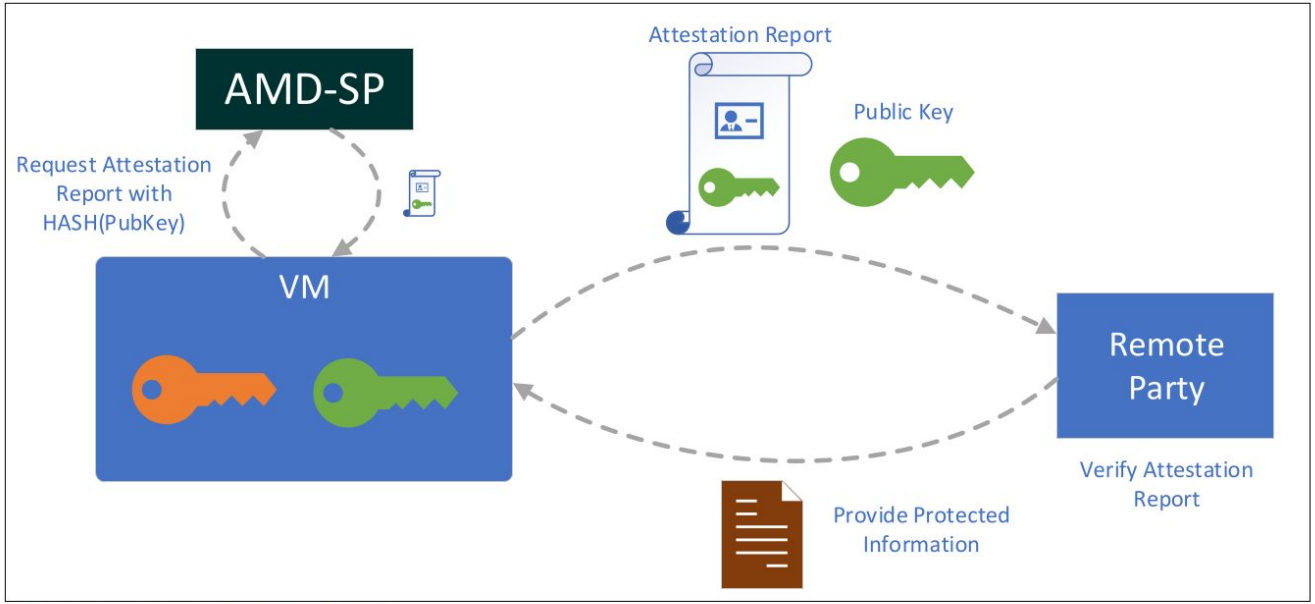
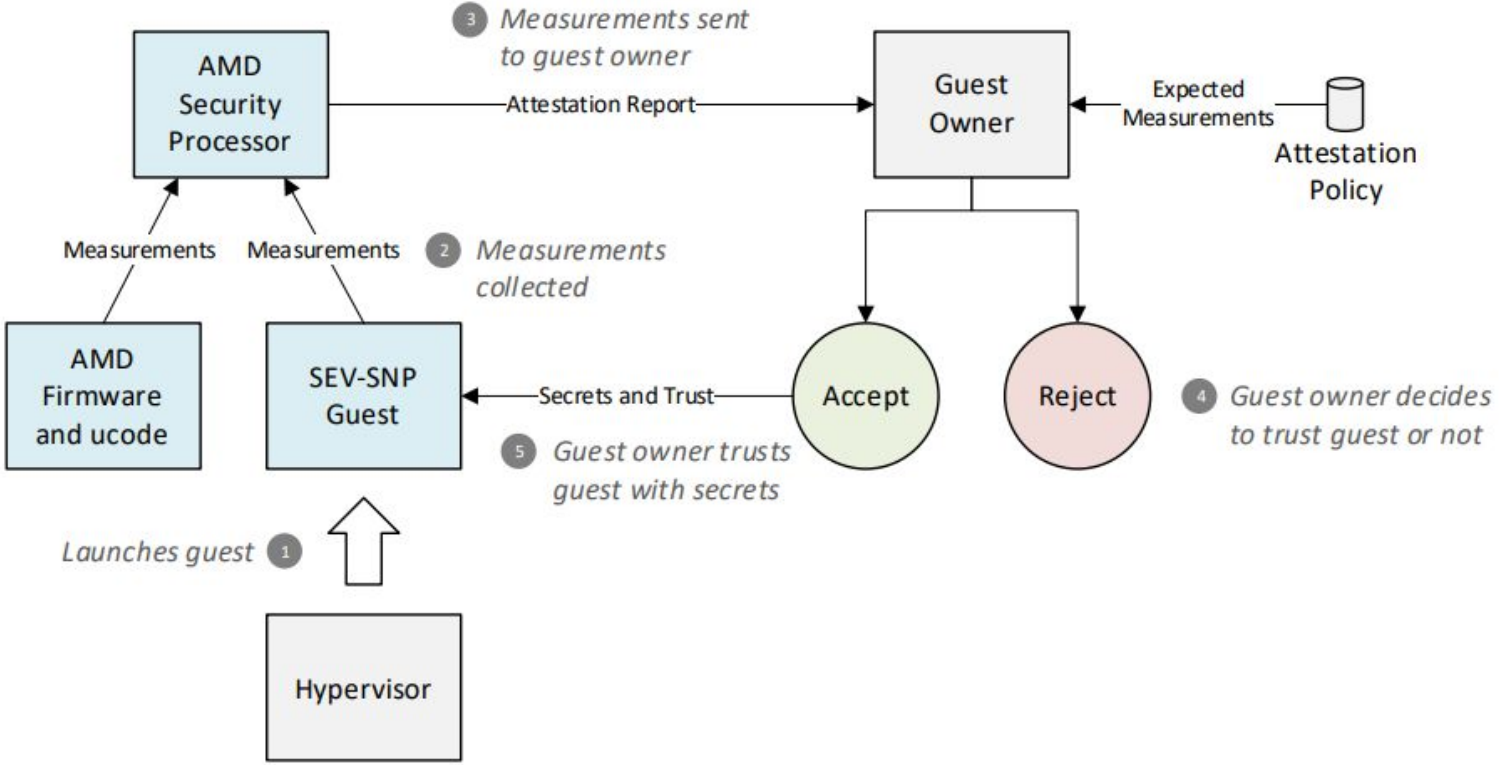


FIGURE 10: SEV-SNP ATTESTATION

Key Attestation



AMD Secure Processor

Key Management

MMIO Registers (Platform Management API, Guest Management API)

Hypervisor -> SEV Driver -> MMIO Registers

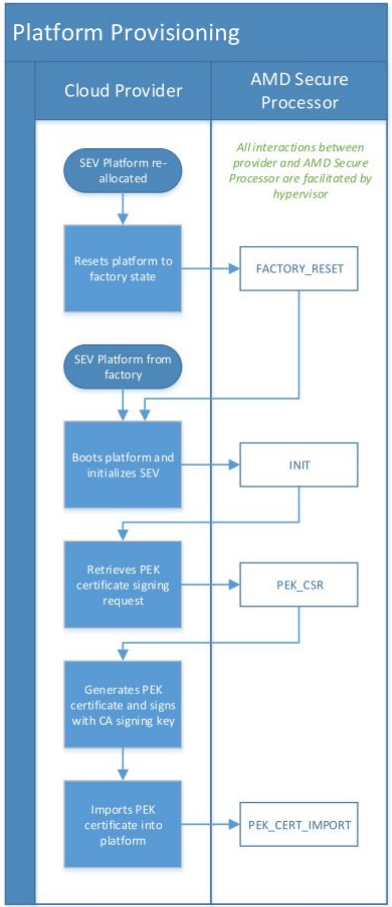
SUSE

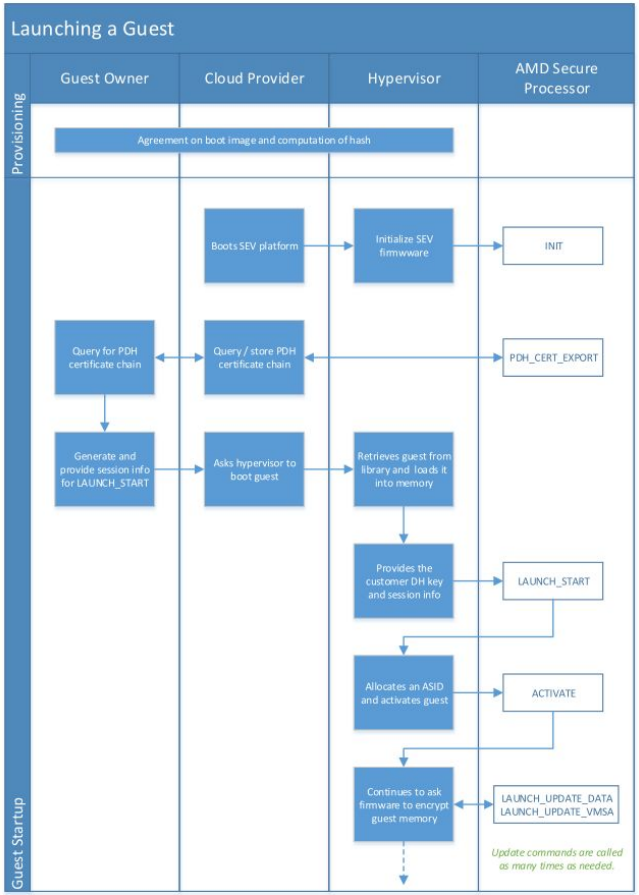
Appendix

SUSE

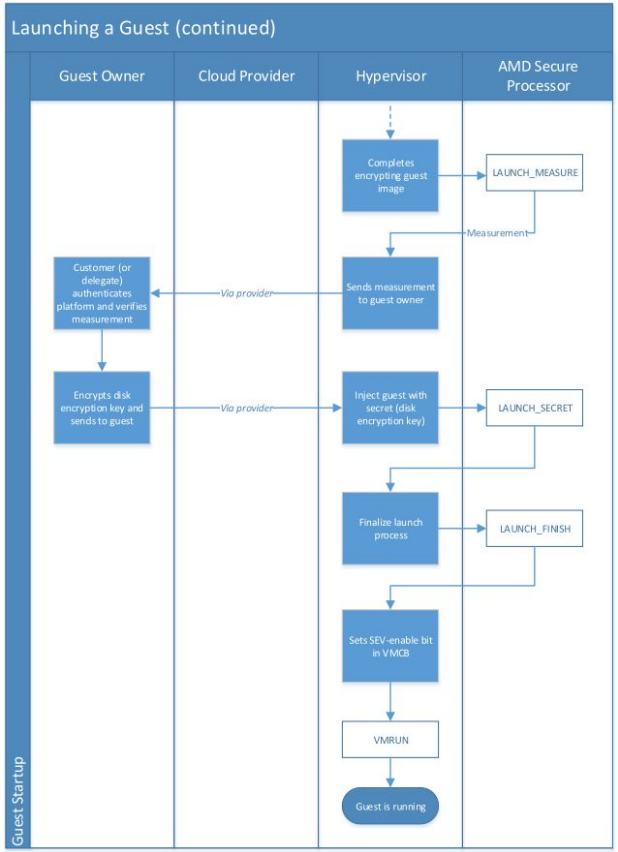
SEV API

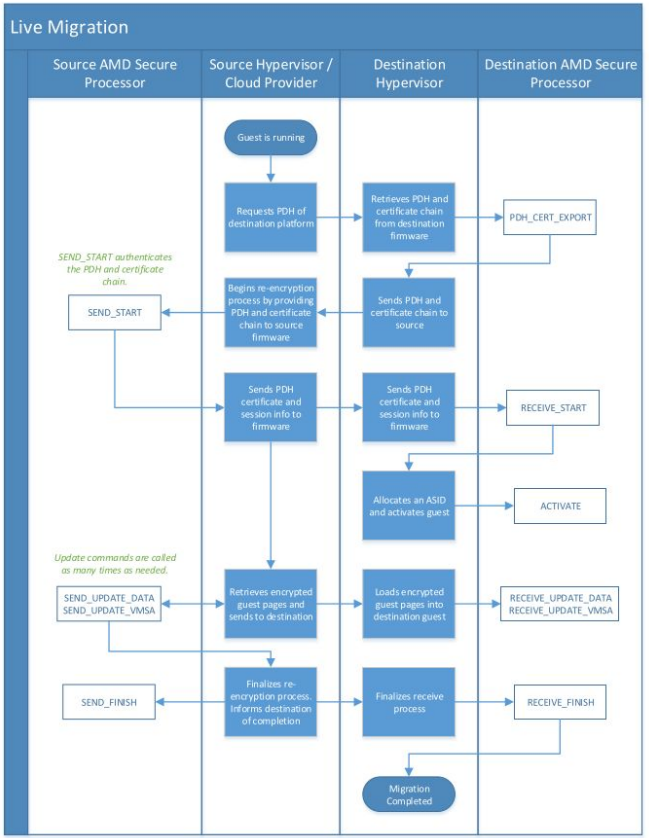
Usage Flows


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
Agreement on boot image and computation of hash
Guest Startup
Boots SEV platform
Initialize SEV firmware
INIT
Query for PDH certificate chain
Query / store PDH certificate chain
PDH_CERT_EXPORT
Generate and provide session info for LAUNCH_START
Asks hypervisor to boot guest
Retrieves guest from library and loads it into memory
Provides the customer DH key and session info
LAUNCH_START
Allocates an ASID and activates guest
ACTIVATE
Continues to ask firmware to encrypt guest memory
LAUNCH_UPDATE_DATA
LAUNCH_UPDATE_VMSA
Update commands are called as many times as needed.

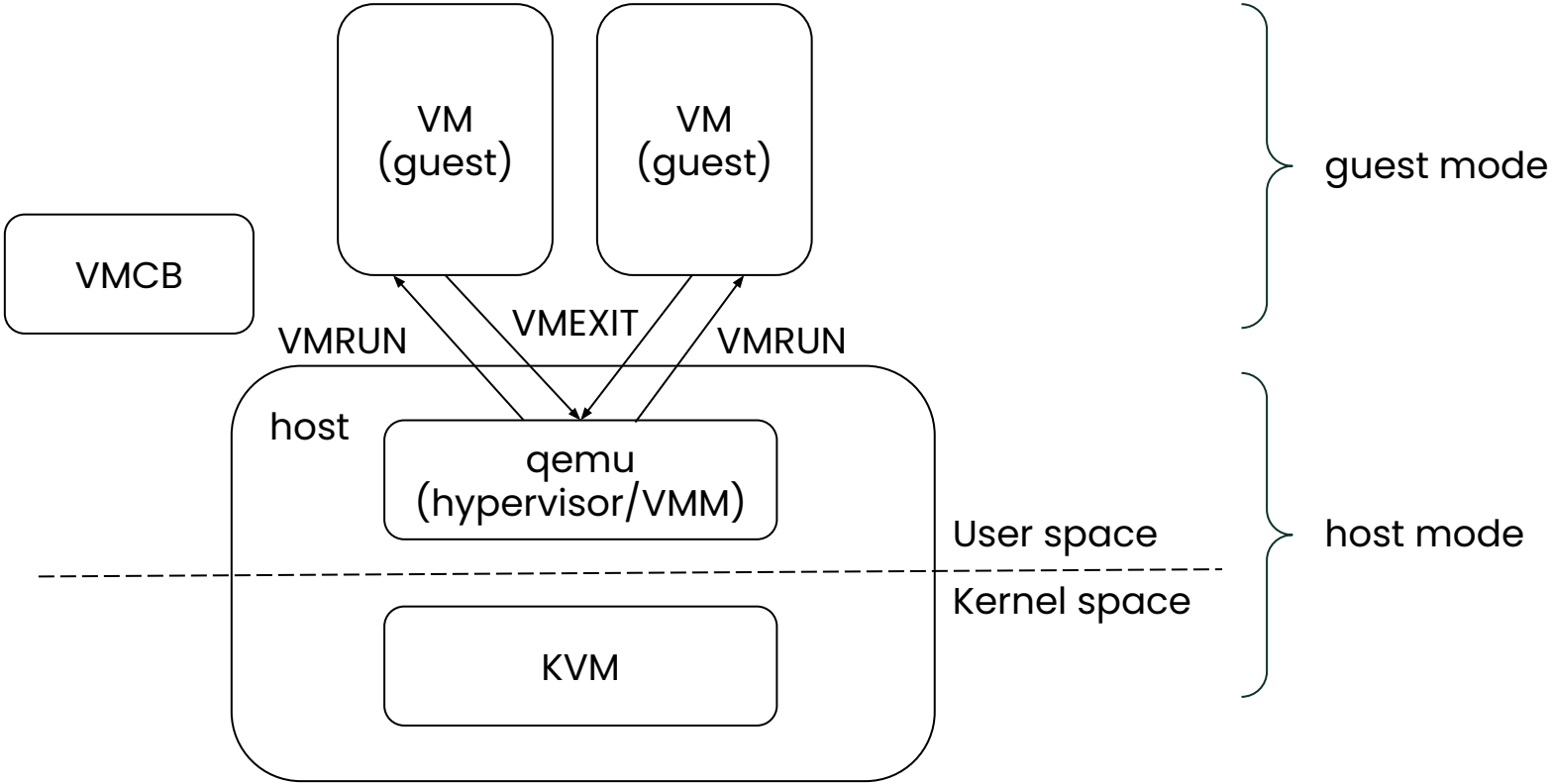
Usage Flows


Launching a Guest (continued)
Guest Owner
Cloud Provider
Hypervisor
AMD Secure Processor
Guest Startup
Completes encrypting guest image
LAUNCH_MEASURE
Measurement
Customer (or delegate) authenticates platform and verifies measurement
Via provider
Sends measurement to guest owner
Encrypts disk encryption key and sends to guest
Via provider
Inject guest with secret (disk encryption key)
LAUNCH_SECRET
Finalize launch process
LAUNCH_FINISH
Sets SEV-enable bit in VMCB
VMRUN
Guest is running

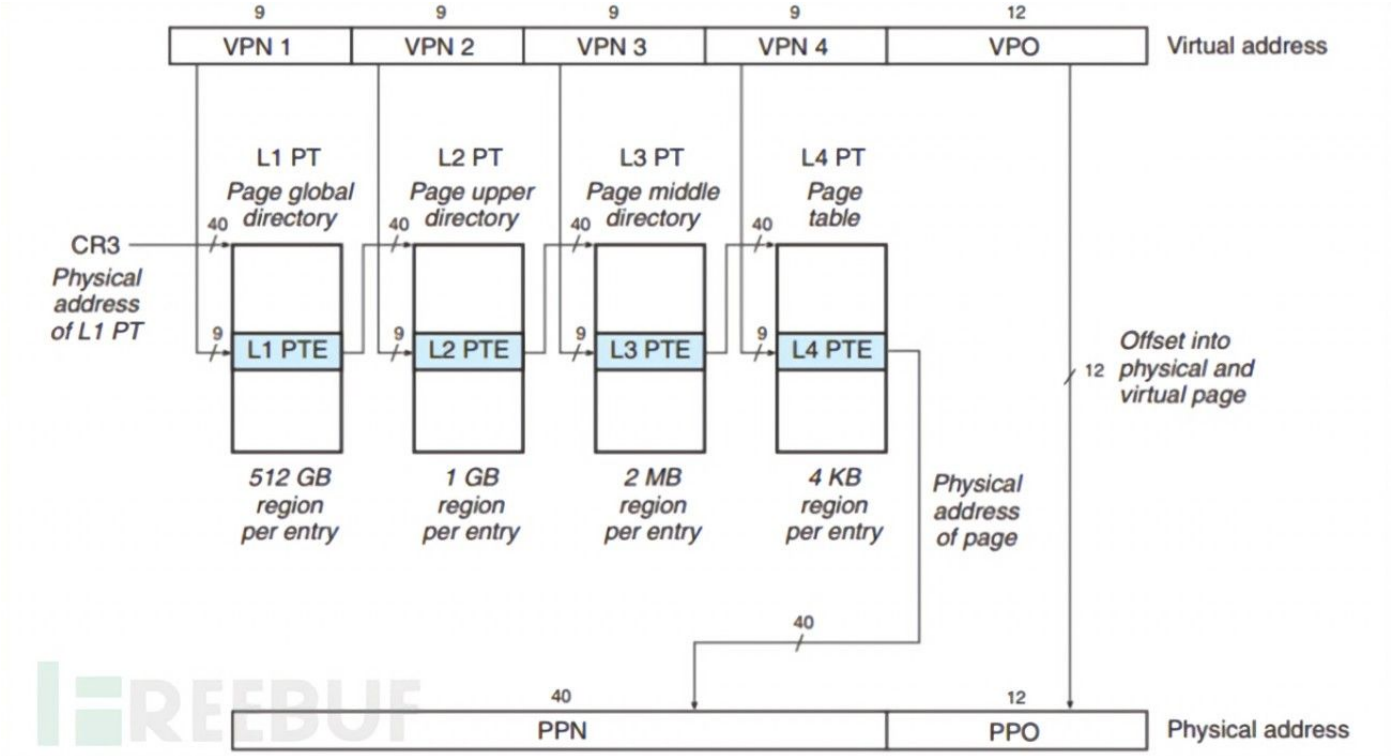

Live Migration
Source AMD Secure Processor
Source Hypervisor / Cloud Provider
Destination Hypervisor
Destination AMD Secure Processor
Guest is running
Requests PDH of destination platform
Retrieves PDH and certificate chain from destination firmware
PDH_CERT_EXPORT
SEND_START authenticates the PDH and certificate chain.
SEND_START
Begins re-encryption process by providing PDH and certificate chain to source firmware
Sends PDH and certificate chain to source
Sends PDH certificate and session info to firmware
Sends PDH certificate and session info to firmware
RECEIVE_START
Allocates an ASID and activates guest
ACTIVATE
Update commands are called as many times as needed.
SEND_UPDATE_DATA
SEND_UPDATE_VMSA
Retrieves encrypted guest pages and sends to destination
Loads encrypted guest pages into destination guest
RECEIVE_UPDATE_DATA
RECEIVE_UPDATE_VMSA
SEND_FINISH
Finalizes re-encryption process. Informs destination of completion
Finalizes receive process
RECEIVE_FINISH
Migration Completed

SUSE

Nested Paging


VM
(guest)
VM
(guest)
guest mode
VMCB
VMRUN
VMEXIT
VMRUN
host
qemu
(hypervisor/VMM)
User space
Kernel space
host mode
KVM

Page Walk


9
9
9
9
12
VPN 1
VPN 2
VPN 3
VPN 4
VPO
Virtual address
L1 PT
Page global directory
L2 PT
Page upper directory
L3 PT
Page middle directory
L4 PT
Page table
CR3
Physical address of L1 PT
40
9
L1 PTE
L2 PTE
L3 PTE
L4 PTE
512 GB region per entry
1 GB region per entry
2 MB region per entry
4 KB region per entry
Offset into physical and virtual page
Physical address of page
PPN
PPO
Physical address

Traditional Paging

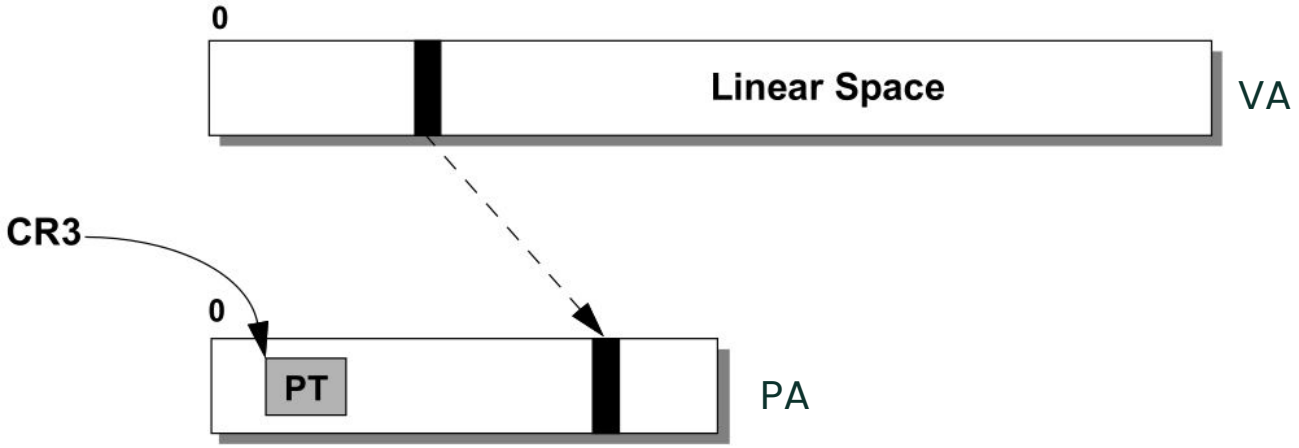


Figure 15-12. Address Translation with Traditional Paging

Nested Page Walk

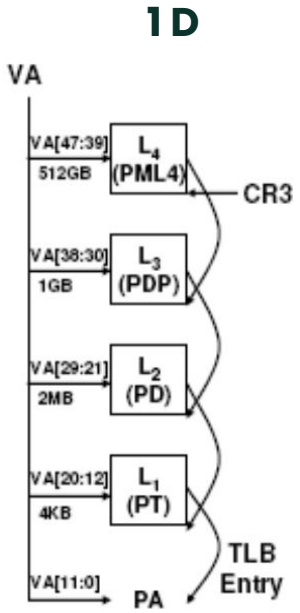


Figure 2: Linear/Virtual to physical address translation algorithm

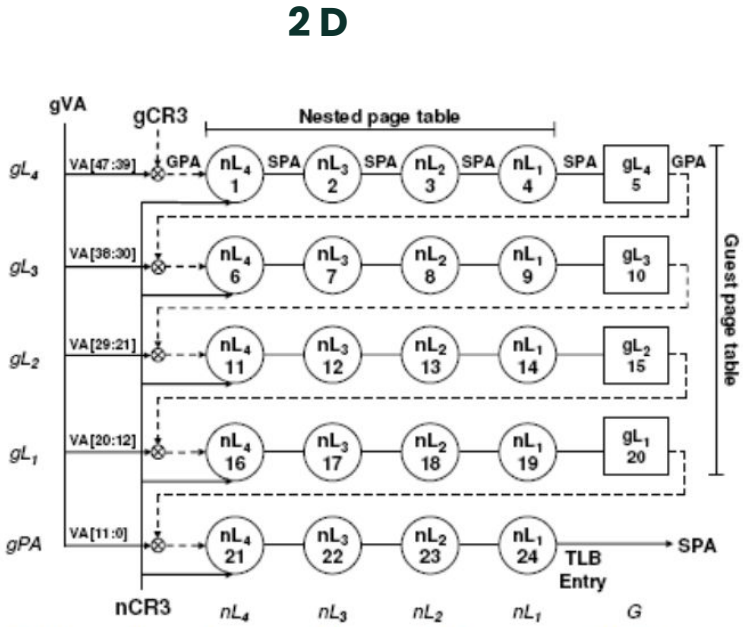
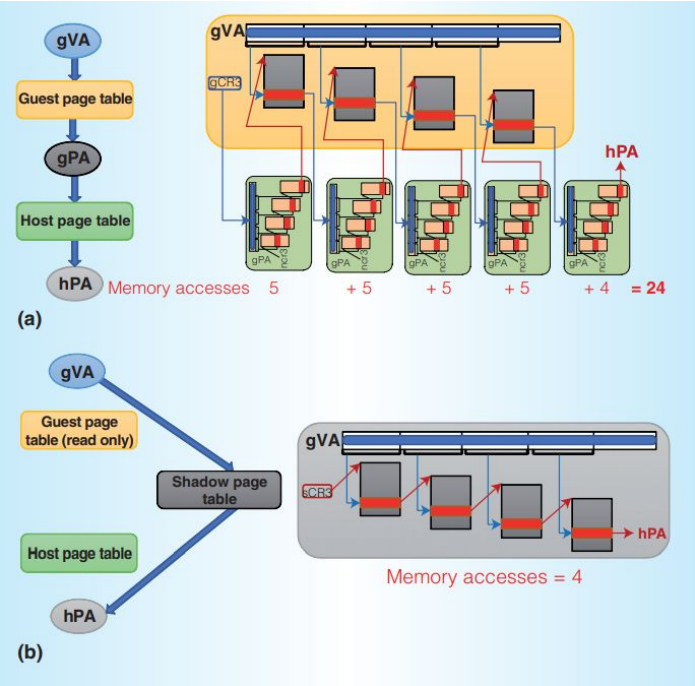


Figure 5: Address translation with nested paging. GPA is guest physical address; SPA is system physical address; nL is nested level; gL is guest level

Page Table Virtualization



(a) Hardware page table virtualization

- Intel - Extended Page Tables (EPT)
- AMD - Nested Page Tables (NPT)

(b) Software page table virtualization

- VMM - Shadow Page(gVA→hPA)
- Managed by VMM (Hypervisor)

Nested Page vs. Shadow Page

Compare	Nested Page	Shadow Page
Page Table Updates	Fast: Direct (hardware-managed)	Slow: VMM (software-managed)
Performance Overhead	Less	Higher
Page Table Walk (gVA -> hPA)	24	4
Hardware Support	Native Walk + Nested Page	Native Page

Nested Paging

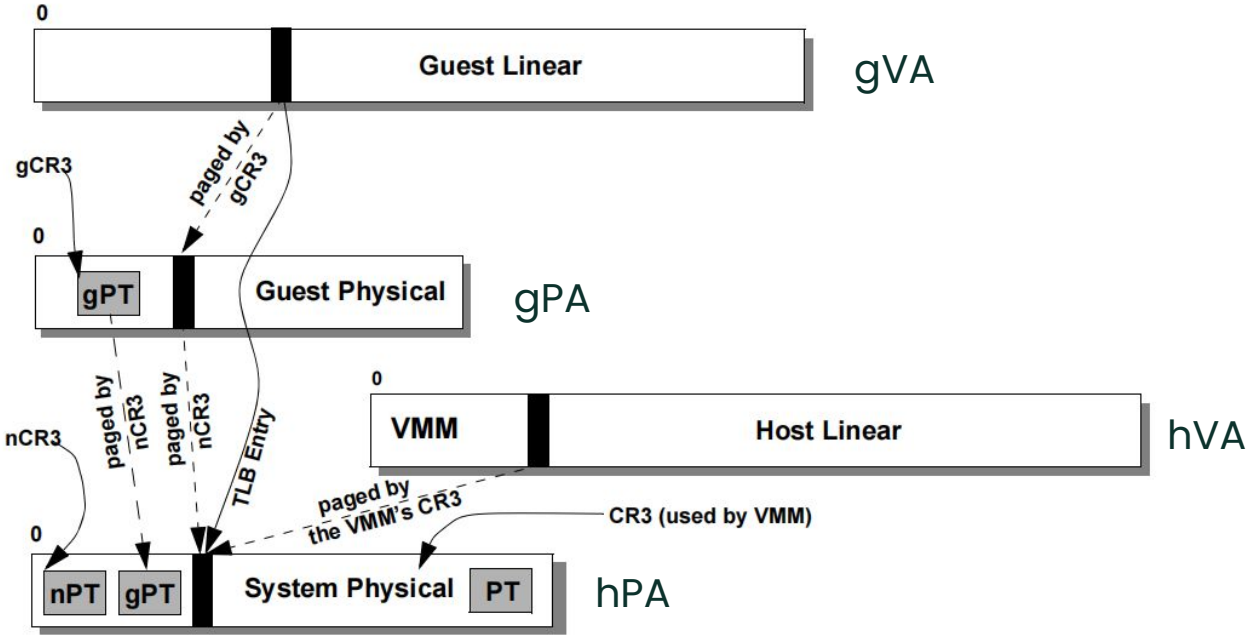


Figure 15-13. Address Translation with Nested Paging

SUSE

AMD-SME

CR3 register lifecycle

- AMD-V
  - AMD-V 提供了 host mode 和 guest mode，分别用来运行 host 和 guest，并且在 guest 运行了特权指令时 VMEXIT 到 host，由 host 进行处理，host 处理完后 VMRUN 进入到 guest 继续运行。
- KVM
  - KVM 使用 AMD-V，封装了 ioctl 供用户程序 (QEMU) 使用．
  - ioctl (…) 设置 guest(VM) 的内存，运行起始位置等
  - ioctl(KVM_RUN)，QEMU 调用 ioctl(KVM_RUN)，使得从 host mode VMRUN 进入到 guest mode，开始运行 guest 。在 guest 运行了 特权指令时 VMEXIT 到 host，即 ioctl (VM_RUN) 函数返回到 QEMU， QEMU 进行处理，处理完成后， 调用 ioctl (KVM_RUN) 进入 guest mode，继续运行 guest，循环这个过程。
  - VMRUN/VMEXIT 时，相关 host/guest 状态保存在 VMCB 中，每个 vCPU 对应一个 VMCB(一个 guest 可能使用多个 vCPU) 。
- QEMU
  - 使用一个进程代表一个 VM(guest)
  - 使用一个线程代表一个 vCPU 。
  - 使用 KVM 提供的 i octl 管理 / 运行 VM 。
  - 实例代码（参考下面的代 码）
- GUEST
  - guest 可以是不做任何修改的 kernel(全虚拟化 ?), 也可以是针对虚拟化修改过的 kernel(para- virtulization ?)
- VMCB
  - VMCB 保存了 相关 host/guest 状态 保存，在进行 host/guest mode 切换时会用到。
-

AMD-SME

- Key
  - **Same encryption key**: All memory encrypted with SME uses the same AES encryption key.
  - **Random key generation**: The AES encryption key is randomly generated each time the system boots.
- **Software cannot read or modify the key**: The encryption key cannot be read or modified by software.
- Determining Support
  - CPUID Fn8000_001F[EAX]. Bit 0 indicates support for Secure Memory Encryption.
- Enabling Memory Encryption Extensions
  - enabled by setting SYSCFG MSR bit 23 (MemEncryptionModEn) to 1
  - software must ensure it is executing from addresses where these upper physical address bits are 0 prior to setting SYSCFG[MemEncryptionModEn]
- Supported Operating Modes
  - Long Mode
  - Legacy PAE-Protected Mode
- I/O Accesses
  - Physical Address Reduction:
  - C-bit
  - MMIO Pages

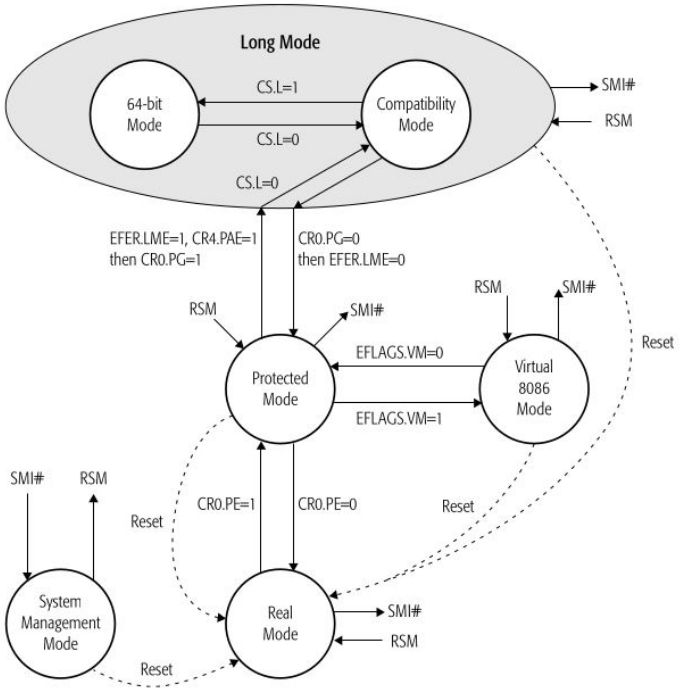


Figure 1-6. Operating Modes of the AMD64 Architecture

Encrypted Memory Access

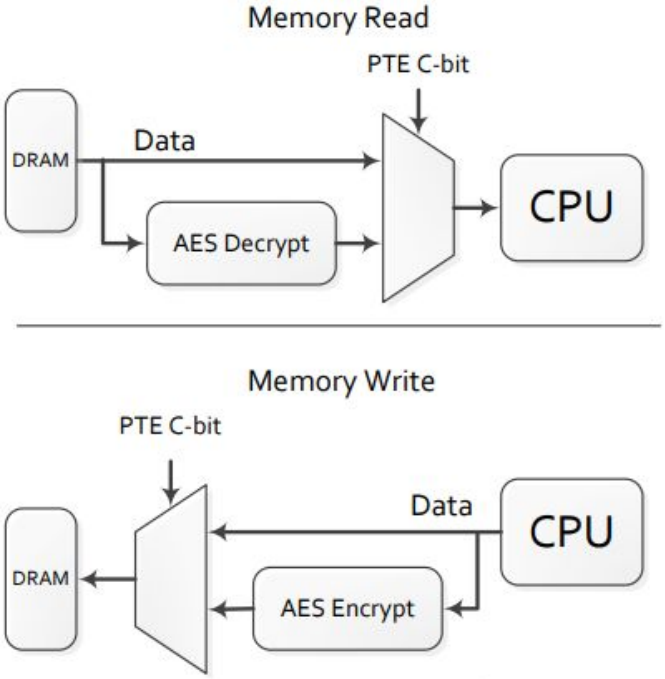


Figure 1: Memory Encryption Behavior

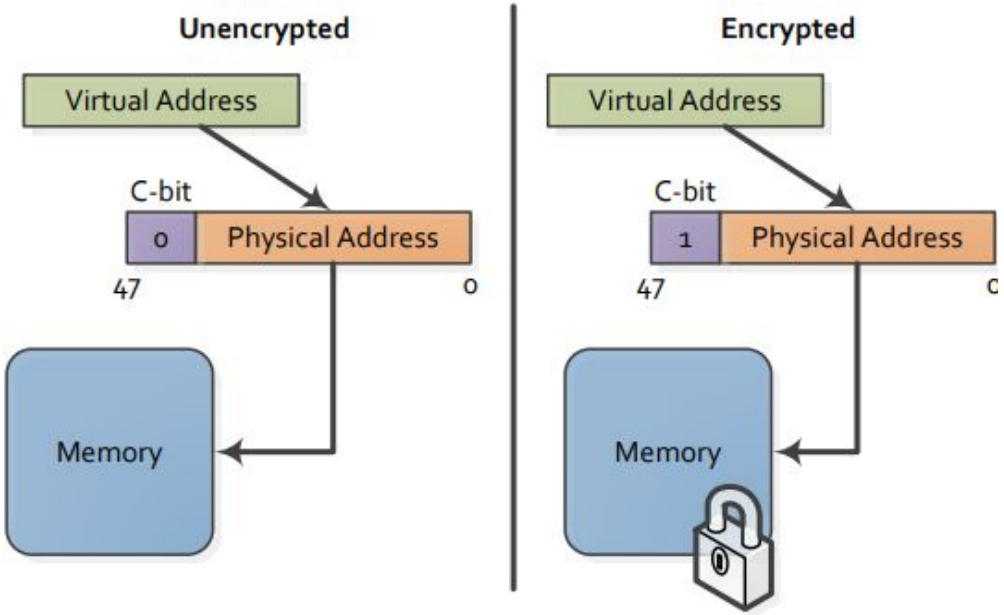


Figure 2: Address Mapping

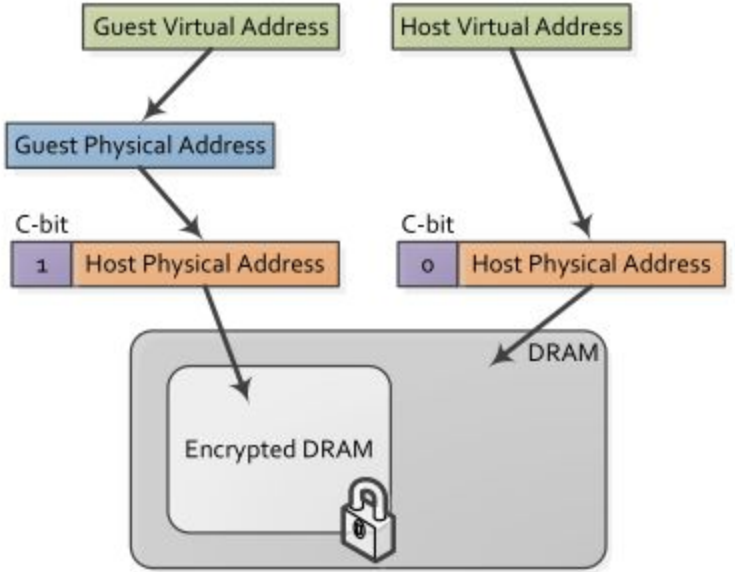


Figure 3: Encrypted VMs

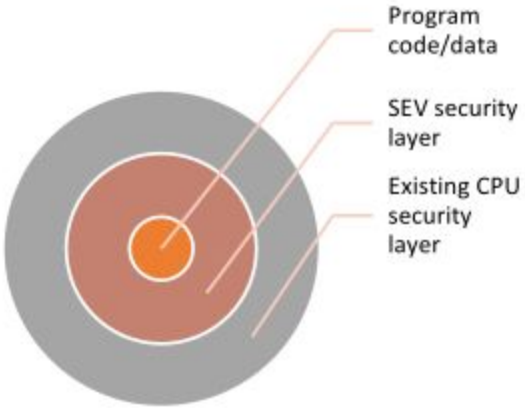


Figure 5: Security Layers

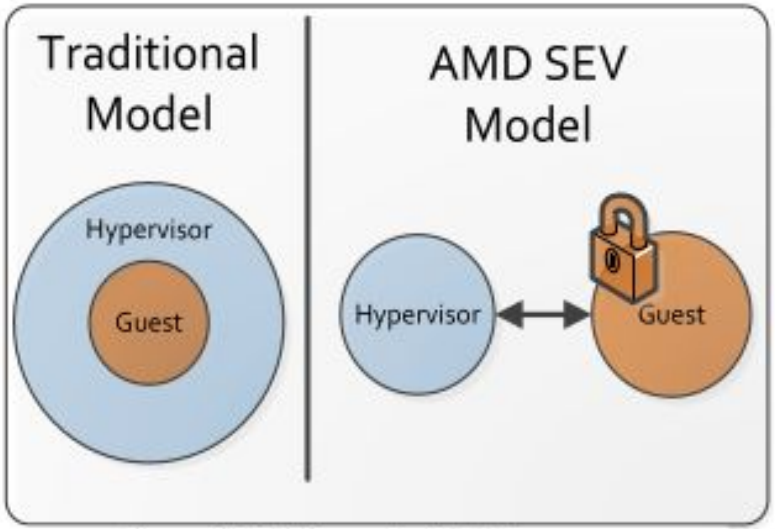


Figure 6: SEV Security Model

SEV Use Cases

Cloud

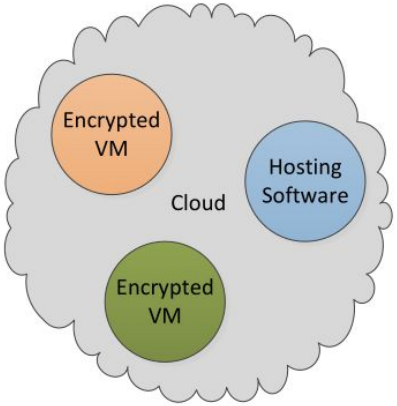


Figure 7: Encrypted VMs in the Cloud

Sandboxing

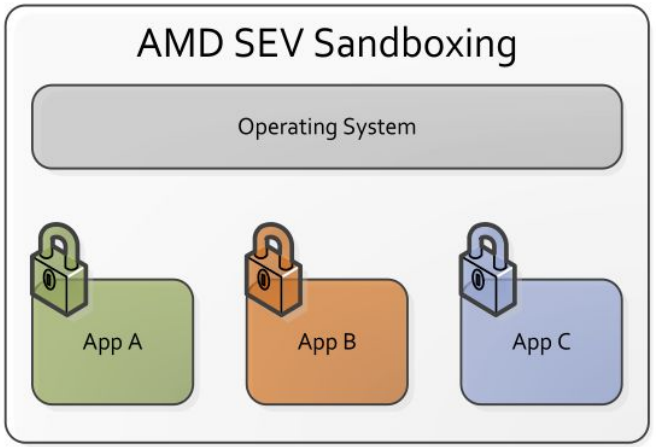


Figure 8: Sandboxing

SEV Architecture

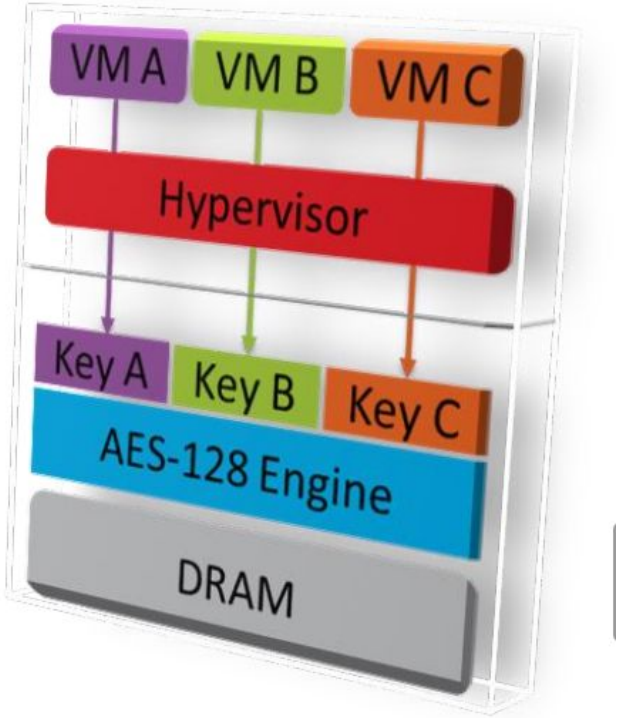


Figure 9: SEV Architecture

SEV Software Implications

- Hypervisor
- Guest

SEV

/qemu/target/i386/sev.c

```
/*
 * QEMU SEV support
 *
 * Copyright Advanced Micro Devices 2016-2018
 *
 * Author:
 *      Brijesh Singh <brijesh.singh@amd.com>
 *
 * This work is licensed under the terms of the GNU GPL, version 2 or later.
 * See the COPYING file in the top-level directory.
 *
 */
```

```
/**
 * SevGuestState:
 *
 * The SevGuestState object is used for creating and managing a SEV
 * guest.
 *
 * # $QEMU \
 *         -object sev-guest,id=sev0 \
 *         -machine ...,memory-encryption=sev0
 */
struct SevGuestState {
    SevCommonState parent_obj;
    gchar *measurement;      Michael Roth, 3 months ago • i386/sev: Introdu

    /* configuration parameters */
    uint32_t handle;
    uint32_t policy;
    char *dh_cert_file;
    char *session_file;
    OnOffAuto legacy_vm_type;
};
```

C-bit (Crypted bit)

- **Functional**
  - Encryption Control
- **Usage**
  - Software uses the **C-bit** to control memory encryption.
- **Location**
  - CPUID Fn8000_001F[EBX] [5:0]

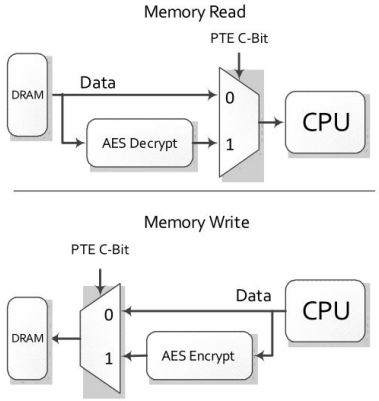


Figure 7-19. Encrypted Memory Accesses

CPUID Fn8000_001F_EBX Secure Encryption

Bits	Field Name	Description
31:16	—	Reserved
15:12	NumVMPL	Number of VM Permission Levels supported.
11:6	PhysAddrReduction	Physical Address bit reduction.
5:0	CbitPosition	C-bit location in page table entry.

CR3 Register

- **Hardware Register**
  - Point to the base address of the currently active page table

- **In VM, CR3 (gCR3)**
  - point to guest page table

- **In host, CR3(nCR3)**
  - points to the nested page table or the host's page table

CPUID Fn8000001F[EAX]

CPUID Fn8000_001F_EAX Secure Encryption

Bits	Field Name	Description
31	IbpbOnEntry	IBPB on Entry supported.
30	HvInUseWrAllowed	Writes to Hypervisor-Owned pages are allowed when marked in-use.
29	NestedVirtSnpMsr	VIRT_RMPUPDATE MSR (C001_F001h) and VIRT_PSMASH MSR (C001_F002h) supported.
28	SvsmCommPageMSR	SVSM Communication Page MSR (C001_F000h) is supported.
27	AllowedSevFeatures	Allowed SEV Features supported.
26	SecureAvic	Secure AVIC supported.
25	SmtProtection	SMT Protection supported.
24	VmsaRegProt	VMSA Register Protection supported.
23:22	—	Reserved
21	RMPREAD	RMPREAD Instruction supported.
20	PmcVirtGuestCtl	PMC Virtualization supported for SEV-ES and SEV-SNP guests.
19	IbsVirtGuestCtl	IBS Virtualization supported for SEV-ES and SEV-SNP guests.
18	VirtualTomMsr	Virtual TOM MSR supported.
17	VmgexitParameter	VMGEXIT Parameter supported.
16	VTE	Virtual Transparent Encryption supported.
15	PreventHostIbs	Disallowing IBS use by the host supported.
14	DebugVirt	Full debug state virtualization supported for SEV-ES and SEV-SNP guests.
13	AlternateInjection	Alternate Injection supported.
12	RestrictedInjection	Restricted Injection supported.
11	64BitHost	SEV guest execution only allowed from a 64-bit host.
10	HwEnfCacheCoh	Hardware cache coherency across encryption domains enforced.
9	TscAuxVirtualization	TSC AUX Virtualization supported.
8	SecureTsc	Secure TSC supported.
7	VmplSSS	VMPL Supervisor Shadow Stack supported.
6	RMPQUERY	RMPQUERY Instruction supported
5	VMPL	VM Permission Levels supported.
4	SEV-SNP	SEV Secure Nested Paging supported.
3	SEV-ES	SEV Encrypted State supported.
2	PageFlushMsr	Page Flush MSR available.
1	SEV	Secure Encrypted Virtualization supported.
0	SME	Secure Memory Encryption supported.

CPUID Fn8000001f[EBX]

CPUID Fn8000_001F_EBX Secure Encryption

Bits	Field Name	Description
31:16	—	Reserved
15:12	NumVMPL	Number of VM Permission Levels supported.
11:6	PhysAddrReduction	Physical Address bit reduction.
5:0	CbitPosition	C-bit location in page table entry.

CPUID Fn8000001f[EBX]

CPUID Fn8000_001F_EBX Secure Encryption

Bits	Field Name	Description
31:16	—	Reserved
15:12	NumVMPL	Number of VM Permission Levels supported.
11:6	PhysAddrReduction	Physical Address bit reduction.
5:0	CbitPosition	C-bit location in page table entry.

CPUID Fn8000001f[ECX]

CPUID Fn8000_001F_ECX Secure Encryption

Bits	Field Name	Description
31:0	NumEncryptedGuests	Number of encrypted guests supported simultaneously.

CPUID Fn8000001f[EDX]

CPUID Fn8000_001F_EDX Minimum ASID

Bits	Field Name	Description
31:0	MinSevNoEsAsid	Minimum ASID value for an SEV enabled, SEV-ES disabled guest.

QEMU SEV Enabled

/qemu/target/i386/sev.c

```
bool
sev_enabled(void)
{
    ConfidentialGuestSupport *cgs = MACHINE(qdev_get_machine())->cgs;

    return !!object_dynamic_cast(OBJECT(cgs), TYPE_SEV_COMMON);
}
```

/qemu/hw/core/qdev.c

```
Object *qdev_get_machine(void)
{
    static Object *dev;

    if (dev == NULL) {
        dev = container_get(object_get_root(), "/machine");
    }

    return dev;
}
```

/qemu/qom/object.c

```
Object *object_dynamic_cast(Object *obj, const char *typename)
{
    if (obj && object_class_dynamic_cast(object_get_class(obj), typename)) {
        return obj;
    }

    return NULL;
}
```

/qemu/tests/qtest/arm-cpu-features.c

```
#define MACHINE     "-machine virt,gic-version=max -accel tcg "
```

/qemu/qom/container.c

```
Object *container_get(Object *root, const char *path)
{
    Object *obj, *child;
    char **parts;
    int i;

    parts = g_strsplit(path, "/", 0);
    assert(parts != NULL && parts[0] != NULL && !parts[0][0]);
    obj = root;

    for (i = 1; parts[i] != NULL; i++, obj = child) {
        child = object_resolve_path_component(obj, parts[i]);
        if (!child) {
            child = object_new("container");
            object_property_add_child(obj, parts[i], child);
            object_unref(child);
        }
    }

    g_strfreev(parts);

    return obj;
}
```

/qemu/qom/object.c

```
Object *object_get_root(void)
{
    static Object *root;

    if (!root) {
        root = object_new("container");
    }

    return root;
}
```

/qemu/qom/object.c

```
Object *object_resolve_path_component(Object *pare
{
    ObjectProperty *prop = object_property_find(pa
    if (prop == NULL) {
        return NULL;
    }

    if (prop->resolve) {
        return prop->resolve(parent, prop->opaque
    } else {
        return NULL;
    }
}
```

```
Object *object_new(const char *typ
{
    TypeImpl *ti = type_get_by_nam

    return object_new_with_type(ti
}
```

```
ObjectProperty *
object_property_add_child(Object *obj, const cha
                          Object *child)
{
    return object_property_try_add_child(obj, na
}
```

Xen vs KVM vs QEMU

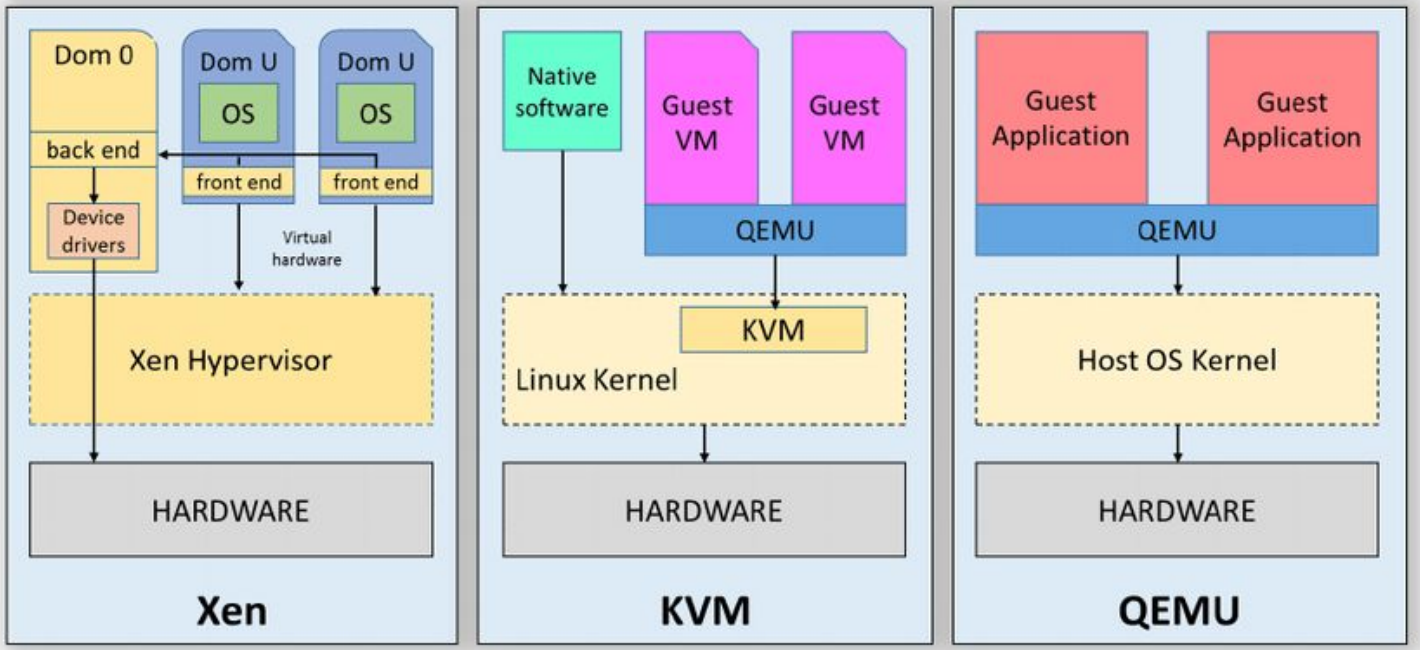


Table 15-36. Fields of an RMP Entry

Name	Notes
Assigned	Flag indicating that the system physical page is assigned to a guest or to the AMD-SP. 0: Owned by the hypervisor 1: Owned by a guest or the AMD-SP
Page_Size	Encoding of the page size. 0: 4KB page 1: 2MB page
Immutable	Flag indicating that software can alter the entry via x86 RMP manipulation instructions. 0: RMP entry can be altered by software 1: RMP entry cannot be altered by software
Guest_Physical_Address	Guest physical address associated with the page
ASID	ASID of guest to which page is assigned
VMSA	Flag indicating that the page is a VMSA page. 0: Non-VMSA page 1: VMSA page
Validated	Flag indicating that the guest has validated the page. See Section 15.36.6 for details. 0: The guest has not yet validated the page 1: The guest validated the page with PVALIDATE
Permissions[0]	VMPL permission masks for the page. See section 15.36.7. for details.
...	
Permissions[n-1]	

RMP Initialization

- **Initialization**:
  - Correctly set `RMP_BASE` and `RMP_END`, align them properly, and zero out the memory in the range before enabling SEV-SNP.

- **RMP Size and Coverage**
  - Understand the RMP structure, calculate its size, and how it maps to physical memory.

- **Role of AMD-SP**
  - The AMD-SP is crucial in finalizing RMP initialization and ensuring secure management of the RMP memory.

RMP Initialization

- **MSRs for RMP Initialization**
  - **MSR C001_0132 (RMP_BASE)**: Defines the starting physical address of the RMP.
  - **MSR C001_0133 (RMP_END)**: Defines the ending physical address of the RMP.
  - **Consistency**: Both `RMP_BASE` and `RMP_END` must be set identically across all cores in the system before globally enabling SEV-SNP.
- **Alignment Requirements**
  - **Alignment**: `RMP_BASE` and `(RMP_END + 1)` must be aligned to 8KB boundaries. Additional alignment requirements may be specified by the AMD Secure Processor (AMD-SP), so it's important to check the latest AMD-SP specifications for any further alignment details.
- **RMP Structure and Size**
  - **Memory Layout**: The memory region between `RMP_BASE` and `RMP_END` is organized as follows:
    - **16KB for Processor Bookkeeping**: This space is reserved for internal processor data.
    - **RMP Entries**: Following the bookkeeping area, the RMP entries are each 16 bytes in size.
  - **Coverage Calculation**: The size of the RMP determines the range of physical memory that the hypervisor can assign to SNP-active VMs. The RMP covers the physical address space from 0x0 to an address calculated by the formula:

    ```
    ((RMP_END + 1 − RMP_BASE − 16KB) / 16B) x 4KB
    ```

    Example: If `RMP_BASE` is set to `0x100000`, to cover the first 4GB of physical memory, `RMP_END` should be set to `0x1103FFF`, resulting in an RMP size just over 16MB.

- **Initialization**
  a. **Process****Pre-Initialization**: Before enabling SEV-SNP globally, the memory range from `RMP_BASE` to `RMP_END` should be zeroed out to ensure it starts in a known state.
  b. **SecureNestedPagingEn Bit**: Set this bit in the SYSCFG MSR to enable SEV-SNP.
  c. **AMD-SP Role**: The hypervisor requests the AMD-SP to finalize the initialization of the RMP. The AMD-SP initializes the RMP and prevents direct software modifications to this memory range.
  d. **Subsequent Modifications**: All further changes to RMP entries must be done using x86 RMP manipulation instructions or through interactions with the AMD-SP, ensuring controlled and secure management of the RMP.


SUSE