White Paper

AMD MEMORY ENCRYPTION

April 21, 2016

David Kaplan, Jeremy Powell, Tom Woller

DISCLAIMER

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.

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

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

Introduction

The need for practical security in modern computing systems is greater than ever. The increase in system complexity, growth of the cloud, and advent of new technologies are all contributing to a computing environment that is difficult yet critical to protect. AMD recognizes these serious challenges and has developed new memory encryption technologies that are designed to address these needs across a variety of systems.

Secure Memory Encryption (SME) defines a simple and efficient architectural capability for main memory encryption. While memory encryption technologies have been used previously in various specialized products and industries, SME is a general purpose mechanism that is flexible, integrated into the CPU architecture, scalable from embedded to high-end server workloads, and requires no application software modifications.

Main memory encryption can be utilized to protect a system against a variety of attacks. While data is typically encrypted today when stored on disk, it is stored in DRAM in the clear. This can leave the data vulnerable to snooping by unauthorized administrators or software or by hardware probing. New non-volatile memory technology (NVDIMM) exacerbates this problem since an NVDIMM chip can be physically removed from a system with the data intact, similar to a hard drive. Without encryption any stored information such as sensitive data, passwords, or secret keys can be easily compromised.

Secure Encrypted Virtualization (SEV) integrates main memory encryption capabilities with the existing AMD-V virtualization architecture to support encrypted virtual machines. Encrypting virtual machines can help protect them not only from physical threats but also from other virtual machines or even the hypervisor itself. SEV thus represents a new virtualization security paradigm that is particularly applicable to cloud computing where virtual machines need not fully trust the hypervisor and administrator of their host system. As with SME, no application software modifications are required to support SEV.

This document presents a technical overview of the SME and SEV and describes how they can be utilized by operating system (OS), hypervisor (HV), and guest virtual machine (VM) software in a variety of different environments to protect data in DRAM.

SME Technical Overview

Main memory encryption is performed via dedicated hardware in the on-die memory controllers. Each controller includes a high performance Advanced Encryption Standard (AES) engine that encrypts data when it is written to DRAM, and decrypts it when read as shown in Figure 1: Memory Encryption Behavior. The encryption of data is done with a 128-bit key in a mode which utilizes an additional physical address-based tweak to protect against cipher-text block move attacks.

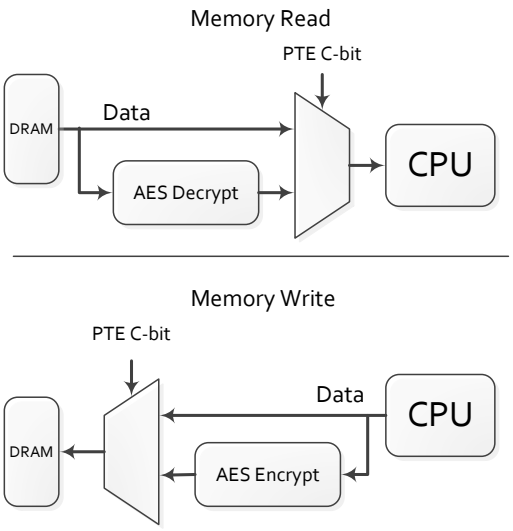

Figure 1: Memory Encryption Behavior

The encryption key used by the AES engine with SME is randomly generated on each system reset and is not visible to any software running on the CPU cores. This key is managed entirely by the AMD Secure Processor (AMD-SP), a 32-bit microcontroller (ARM® Cortex®-A5) that functions as a dedicated security subsystem integrated within the AMD SOC. The key is generated using the onboard NIST SP 800-90 compliant hardware random number generator and is stored in dedicated hardware registers where it is never exposed outside the SOC in the clear. Unlike the SEV mode described later, SME does not require the software running on the CPU cores to participate in key management.

The control over which pages of memory are encrypted is controlled via the OS or HV in the software managed page tables. After memory encryption is enabled, physical address bit 47 (aka the C-bit, as in enCrypted) is utilized to mark if a memory page is protected. The OS or HV sets bit 47 of a physical address to 1 in the page table entry (PTE) to indicate the page should be encrypted, causing accesses to that memory to be automatically encrypted and decrypted by the AES engine in the memory controller.

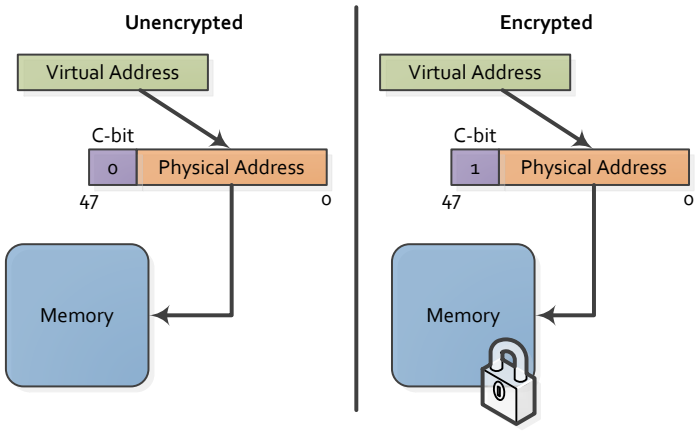

Figure 2: Address Mapping

The encryption and decryption of memory through the AES engine does incur a small amount of additional latency for DRAM memory accesses. The impact of this latency to software is very dependent on the system workload but is estimated to have a very small overall effect on system performance. If only a subset of memory is encrypted, the performance impact will be less as unencrypted accesses generally incur no additional latency.

SME Use Models

This section discusses two different example models by which software can utilize the SME feature. In the first model, all of DRAM is encrypted while in the second, only a select region (corresponding to guest VMs) is encrypted.

Full Memory Encryption

Full memory encryption is a simple yet compelling model for many computing systems, especially when physical attacks on the system are of concern, including government data centers and/or smaller or remote enterprise data centers that have more limited physical security. With full memory encryption, all DRAM contents are encrypted utilizing the random key which provides strong protection against cold boot, DRAM interface snooping, and similar types of attacks. For systems with NVDIMM, full memory encryption also provides protection against an attacker removing a memory module and attempting to extract its contents.

Supporting full memory encryption with SME is accomplished by either the OS or HV setting the C-bit in all DRAM physical addresses in all the page tables. This should include both instruction and data pages, as well as the pages corresponding to the page tables themselves. In essence, the OS or HV software may view the system as having DRAM that starts at address 0x8000_0000_0000. In the case of a HV setting the C-bit to encrypt all physical memory, all the VMs controlled by the HV are encrypted along with the HV and use the same encryption key.

Note that DMA to memory encrypted via SME is supported. From a device standpoint, an encrypted memory access is just a normal memory access with bit 47 set.

Partial Memory Encryption

The use of the C-bit in the page tables provides the flexibility for the OS or HV to selectively encrypt only a subset of memory if it desires. Doing so still provides physical protection of the encrypted memory, but can also be used to improve performance on non-sensitive data. In addition, the choice of encrypted vs. unencrypted memory can be used to provide a layer of isolation for critical workloads.

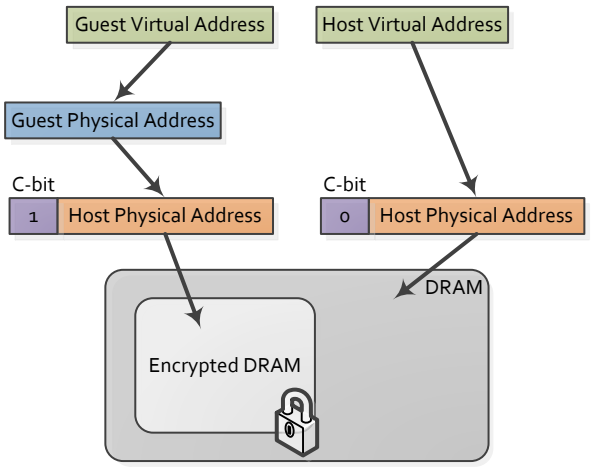

Figure 3: Encrypted VMs

One example of this isolation would be a system which only marks memory used by guest VMs as encrypted. This can be done without any modification to the guest VMs whatsoever. By setting the C-bit in all the nested page table entries corresponding to DRAM (as shown in Figure 3: Encrypted VMs), the hypervisor enables encryption for the VM memory only.

Such a scenario could be used to protect VMs against a rogue machine administrator. While the administrator would have access to the hardware and host system, they would not be able to inspect guest VM data even with memory scanning utilities.

SME Special Considerations

SME provides a straightforward method for software to encrypt some or all memory pages. There are some important special programming considerations however that must be taken into account when working with encrypted memory.

The primary consideration is that hardware does not manage coherency between encrypted and unencrypted copies the same page. Therefore, software must be careful when modifying the C-bit of pages and ensure that if the C-bit of a page needs to be modified, that page must be flushed from the CPU caches prior to the page table modification.

Worth noting is that it is allowed for devices to issue DMA to encrypted memory but like with CPU accesses, they must set bit 47 of the physical address which is not possible on some 32-bit legacy devices. To compensate, software can utilize the IOMMU to re-map device request addresses to addresses with the C-bit set.

Additional technical details of the SME feature can be found in the latest copy of the AMD Programmer's Manual (APM) volume 2.

Transparent SME

While SME provides a lot of flexibility for managing main memory encryption, it does require support in the OS/HV. For systems that desire only the physical protection of SME but run legacy OS or HV software, they may use a mode called Transparent SME (TSME). In TSME, all memory is encrypted regardless of the value of the C-bit on any particular page. This mode provides a simple method to enable encryption without requiring software modifications.

TSME can be enabled via a BIOS setting on supported platforms. When TSME is used, other memory encryption features (including SEV) are not available.

Intro to SEV: Complexity and Consolidation

With advances in both hardware and software, computing systems today are more complex than ever. The size and functionality of software, especially the most privileged kernel-level software responsible for security in the system, has increased dramatically. The Linux kernel for instance has grown from less than 200k lines of code to almost 22 million lines today.

This evolution has also led to more consolidation than ever before allowing systems to perform more tasks and do more work on the same hardware. This consolidation has led to many positive results such as the cloud, and virtual data centers on which anyone can buy affordable compute time.

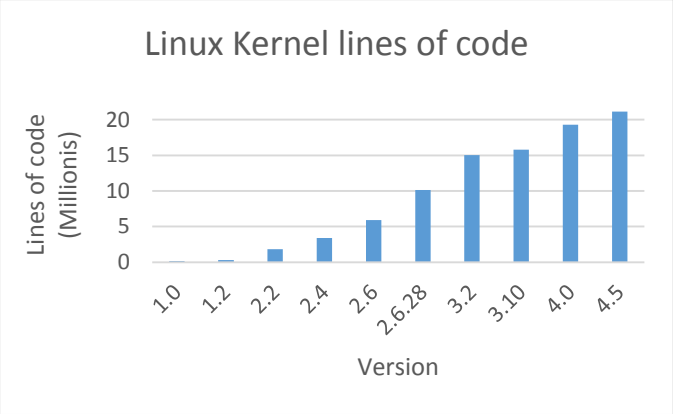

Figure 4: Kernel Code Size

Yet both complexity and consolidation have a generally negative effect on the security of a system. Complex software is harder to verify and is more likely to have bugs which hackers could exploit. A typical Linux system loads close to 5.5MB of kernel code into memory which must be bug-free to avoid compromising the system.

Consolidated systems can also negatively impact security by increasing the attack surface, allowing many different pieces of software to use the same hardware and share resources like memory. This combination yields a security model which relies on a large amount of privileged bug-free code, which coupled with a large attack surface provides an increased possibility for intrusions.

To counter these forces, Secure Encrypted Virtualization is a new feature being added to the AMD architecture which is designed to better deal with the complexity and isolation needs of modern systems. SEV enhances isolation through the use of cryptography, encrypting code and data and enables an entirely new security model where code can be cryptographically protected from higher privileged code such as a hypervisor.

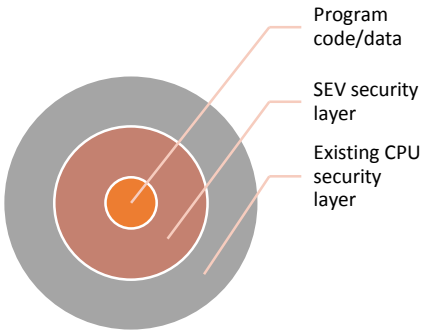

Figure 5: Security Layers

SEV Security Model

Traditional computing systems have operated using a ring-based security model. In this model, high privilege code has full access to the resources at its level and of all lower privileged levels.

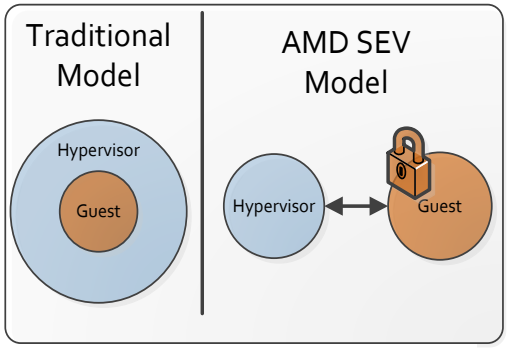

Figure 6: SEV Security Model

In the SEV model, code executing at different levels (namely hypervisor vs guest) is isolated so neither has access to the resources of the other. Even though the hypervisor level is traditionally "more privileged" than the guest level, SEV separates these levels through cryptographic isolation. This provides additional security for lower privileged code, without requiring trust in the high privileged code on which the less privileged code is dependent upon for startup and execution. Communication between hypervisor and guest is still possible, but those communication paths are tightly controlled.

Consequently, SEV technology is built around a threat model where an attacker is assumed to have access to not only execute user level privileged code on the target machine, but can potentially execute malware at the higher privileged hypervisor level as well. The attacker may also have physical access to the machine including to the DRAM chips themselves. In all these cases, SEV provides additional assurances to help protect the guest virtual machine code and data from the attacker. Note that SEV does not protect against denial-of-service attacks against the guest.

SEV Use Cases

Built around the concept of hardware VMs, SEV can be used to enhance security in a variety of use cases. Due to its use of main memory encryption, SEV provides the all of the same security benefits as SME for physical attack protection described earlier. In addition, SEV can be used to protect environments such as the following.

Cloud

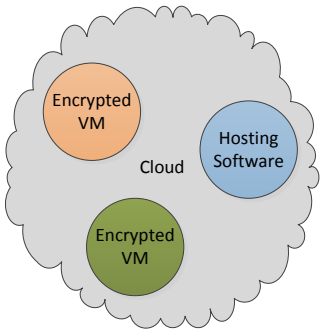

Figure 7: Encrypted VMs in the Cloud

The growth of the cloud and in particular Infrastructure as a Service (IaaS) data centers has made computational power both cheap and affordable. This growth does not come without security challenges however, as cloud infrastructure/personnel may not always be trustworthy. This can be particularly true when dealing with sensitive data, such as health records or trade secrets. Furthermore, the sharing of hardware amongst multiple customers can raise security concerns for owners of many types of workloads. Despite the best efforts of software designers, there have been many examples of cases where this isolation has failed, leading to the compromise of sensitive code or data.

SEV can be used to increase the level of security in these IaaS clouds by providing better security isolation, rooted in the hardware itself. While existing security technologies like Microsoft's BitLocker® and LUKS can protect data-at-rest in hard drives, SEV protects data-in-use enabling customer workloads to be protected cryptographically from each other as well as protected from the hosting software. Even an

administrator with malicious intentions at a cloud data center would not be able to access the data in a hosted VM.

Sandboxing

Through the use of the hardware VM constructs, SEV is built around an idea of secure sandbox environments where software can execute and is protected from all other software on the system. These sandboxes can be as big as a full VM with its own disk and OS, but they can also be small and used for more fine-grained isolation. For example, SEV hardware could be used to cryptographically isolate Docker containers from the host system to better protect confidential data.

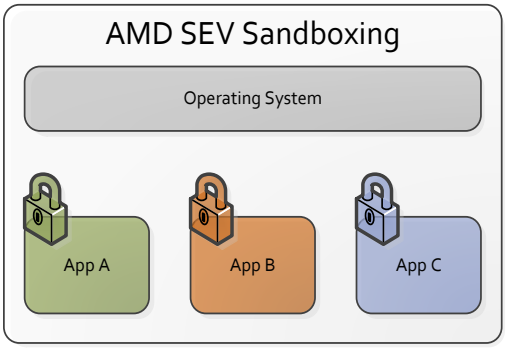

Figure 8: Sandboxing

SEV Architecture

Technical Overview

SEV is an extension to the AMD-V architecture which supports running multiple VMs under the control of a hypervisor. When enabled, SEV hardware tags all code and data with its VM ASID which indicates which VM the data originated from or is intended for. This tag is kept with the data at all times when inside the SOC, and prevents that data from being used by anyone other than the owner.

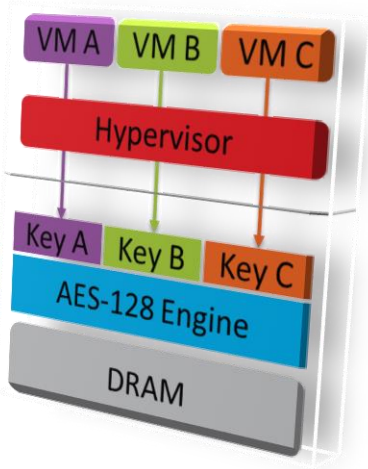

Figure 9: SEV Architecture

While the tag protects VM data inside the SOC, AES with 128 bit encryption protects data outside the SOC. When data leaves or enters the SOC, it is encrypted/decrypted respectively by hardware with a key based on the associated tag.

Each VM as well as the hypervisor is associated with a tag, and consequently an associated encryption key. Because of the tag and memory encryption, data is restricted to only the VM using that tag. If that data is accessed by anyone else, including the hypervisor, they will only be able to see the data in its encrypted form. This provides strong cryptographic isolation between the VMs, as well as between the VMs and the hypervisor.

Encrypted Memory

SEV uses the same high performance memory encryption engine as the SME feature described earlier. It also uses the same C-bit in the page tables to mark pages as encrypted, although with some additional restrictions.

One of the key features of SEV is that guest VMs are able to choose which data memory pages they would like to be private. This choice is done using the standard CPU page tables, and is fully controlled by the guest. Private memory is encrypted with the guest-specific key, while shared memory may be encrypted with the hypervisor key. This feature allows VMs to mark selected pages of memory data they want to keep

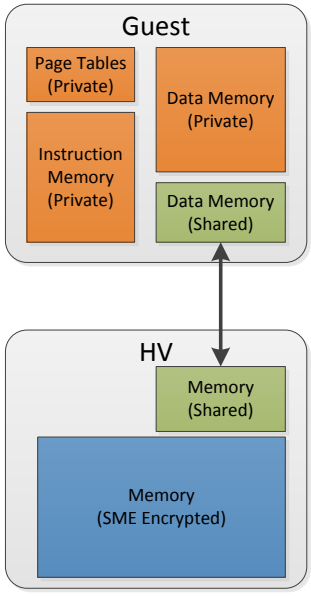

Figure 10: Guest/HV Communication Example

confidential (private), and others to be used for communication with other VMs or the hypervisor. In a typical arrangement, the guest would map all of its code and data as private, except for specific shared pages that it chooses to expose. For security, SEV hardware does require certain types of memory (including instruction pages and page tables) to always be private to protect the VM.

An example communication configuration is shown in Figure 10. In this example, the SEV-enabled guest and hypervisor communicate through memory that both entities mark as shared. All other guest memory is encrypted with the guest's key (which the HV cannot use). Any memory the HV does not use for direct guest communication is encrypted using the SME feature described earlier.

Key Management

The security of SEV is highly dependent on the security of the memory encryption keys. Exposure to malicious entities, such as a malicious or buggy hypervisor, can endanger the SEV protected guest. While the hypervisor must manage the guest and its resources, the hypervisor must never gain knowledge of the memory encryption keys themselves. The SEV firmware that runs within the AMD-SP provides a secure key management interface to accomplish this. The hypervisor uses this interface to enable SEV for secure guests and perform common hypervisor activities such as launching, running, snapshotting, migrating, and debugging a guest.

To protect SEV enabled guests, the firmware assists in the enforcement of three main security properties: authenticity of the platform, attestation of a launched guest, and the confidentiality of the guest's data.

Authenticating the platform prevents malicious software or a rogue device from masquerading as a legitimate platform. The authenticity of the platform is proven with its identity key. This key is signed by AMD to demonstrate that the platform is an authentic AMD platform with SEV capabilities. It is also signed by the owner of the platform to show who administers and owns the machine to the owners of guests or to other instances of the firmware on remote platforms.

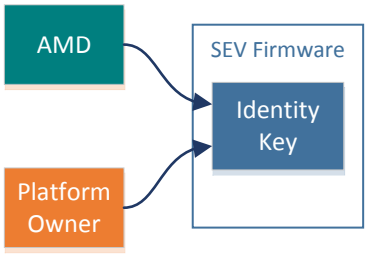

Figure 11: Authenticating the Firmware

Attestation of the guest launch proves to guest owners that their guests securely launched with SEV enabled. A signature of various components of the SEV related guest state—including initial contents of memory—is provided by the firmware to the guest owner to verify that the guest is in the expected state. With this attestation, a guest owner can ensure that the hypervisor did not interfere with the initialization of SEV before transmitting confidential information to the guest.

An example of this process is shown in Figure 12. Initially, the guest owner provides the guest image to the cloud system. The SEV firmware assists in launching the guest and provides a measurement back to

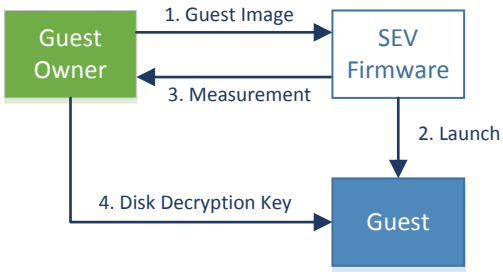

Figure 12: Guest Attestation Example

the guest owner. If the guest owner deems this measurement correct, they in turn provide additional secrets (such as a disk decryption key) to the running guest allowing it to proceed with start-up.

Confidentiality of the guest is accomplished by encrypting memory with a memory encryption key that only the SEV firmware knows. The SEV management interface does not allow the memory encryption key—or any other secret SEV state—to be exported outside of the firmware without properly authenticating the recipient. This prevents the hypervisor from gaining access to the keys and consequently the guest's data.

The interface also provides a mechanism to migrate the guest data to another SEV capable platform. During this operation, the guest's memory contents remains encrypted during transmission. Once the remote platform is authenticated, the SEV firmware sends the guest's memory encryption keys securely so the remote platform can run the guest itself. This transport mechanism allows a hypervisor to implement migration and snapshot functions securely with SEV enabled.

SEV Software Implications

Hypervisor

As with traditional virtualization, SEV continues to rely on the hypervisor for many VM functions such as device emulation and scheduling, but reduces the reliance on the hypervisor for security. A guest running with SEV enabled still uses the hypervisor as needed, but is able to protect itself by marking memory pages as private when they are not intended to be shared outside the VM.

During runtime the hypervisor communicates with the AMD-SP in order to coordinate the management of memory encryption keys. This communication is done via the AMD-SP driver and involves tasks such as informing the AMD-SP when a VM is about to be run, thus allowing the AMD-SP to load the appropriate encryption key into the AES-128 encryption engine. The hypervisor also communicates with the AMD-SP to establish a secure mechanism to perform guest attestation, perform migration, etc.

Although the hypervisor has control over the ASID used to run a VM and select the encryption key, this is not considered a security concern since a loaded encryption key is meaningless unless the guest was already encrypted with that key. If the incorrect key is ever loaded or the wrong ASID is used for a guest, the first instruction fetch of that guest will fail as memory will be decrypted with the wrong key, causing junk data to be executed (and very likely causing a fault).

Guest

The OS in an SEV-enabled guest must be aware of this new hardware feature and configure its page tables appropriately. This may be done in a similar method as the SME full memory encryption mode where most DRAM addresses are configured to have the C-bit (bit 47) set to 1.

One important consideration for an SEV-enabled guest is that DMA into guest encrypted memory is not allowed by the SEV hardware for security reasons. All DMA, whether from a real hardware or a HV emulated device, must occur to shared guest memory. The guest OS can therefore choose to allocate memory pages for DMA as shared (C-bit clear), or may copy data to/from a special buffer (aka "bounce buffer") for DMA purposes. Some operating systems have existing support for bounce buffers which may be used for this purpose, such as the swiotlb Linux functionality.

Finally, it should be noted that multi-core guests are supported with SEV and data can be shared amongst those cores with no additional performance penalty. The hypervisor must simply use the same ASID for all virtual CPU instances for a particular guest.

SEV Special Considerations

Many of the special considerations listed earlier regarding SME functionality also apply to SEV. In particular, as with SME, a page must be flushed from the cache prior to accessing it with a different C-bit. Also, prior to replacing a hardware memory encryption key, the hypervisor software must perform a full cache flush to ensure any modified data using that key has been written back to DRAM.

Finally, since the C-bit is only controllable by the guest OS when operating in 64-bit or 32-bit PAE mode, in all other modes the SEV hardware forces the C-bit to a 1. This allows a guest OS to start running encrypted code immediately and then transition into its final mode securely.

Conclusion

The two memory encryption features presented in this document represent major steps forward in general purpose computer security that may be utilized in a variety of environments.

The Secure Memory Encryption technology is a flexible yet powerful architectural feature which allows for main memory encryption for an operating system or hypervisor. This document has explained a few specific use models for memory encryption, including full memory encryption, selective encryption, and transparent encryption. All models provide new protections against physical hardware attacks, and can in some cases be used to help protect systems from rogue administrators. Other use models are likely possible as well and can provide additional options and performance/security tradeoffs. No application software changes are required for SME, and with the appropriate operating system or hypervisor modifications, all applications in a system can be protected.

Additionally, VM security can be enhanced with Secure Encrypted Virtualization, which supports running encrypted guests that a hypervisor cannot directly access. This provides new protections not only for cloud users but also for cloud hosters who wish to limit their visibility into customer data. Based on AMD-V technology, SEV can be used in both cloud and Docker-type models and provides a new security model for virtualized environments. Like the SME technology, no application software changes are required to guest VMs and VMs encryption is performed quickly and transparently with dedicated hardware engines.