June 2022

Encrypted Memory Access by Peripheral Devices of a Confidential Virtual Machine

Marc S. Orr

Rachit Mathur

Erdem Aktas

Andrew Honig

Encrypted Memory Access by Peripheral Devices of a Confidential Virtual Machine

ABSTRACT

A confidential virtual machine (CVM) uses a hardware-rooted key to encrypt customer data written to memory. For enhanced security, the CVM makes the plaintext-to-ciphertext map depend on the physical address (PA) of the memory location. Peripheral devices operate not in PA space but in input-output address space. Currently, peripherals can only perform memory accesses without encryption, or need to use a two-pass, high-latency, power-intensive encryption procedure involving a transport key distinct from the hardware-rooted key.

This disclosure describes techniques to enable a peripheral device of a confidential virtual machine to access encrypted memory using a single encryption pass. The techniques enable secure, high-speed computing at low power consumption. Address translation between input-output and physical address spaces is accounted for such that peripherals continue to work in input-output address space while encryption continues to depend on the physical address of the data. The techniques obviate compute-intensive transport keys subsidiary to hardware-rooted virtual machine keys.

KEYWORDS

- Confidential virtual machine (CVM)
- Trusted computing
- Memory management unit (IOMMU)
- Direct memory access (DMA)
- Physical address
- Input-output address
- Encrypted memory
- Advanced encryption standard (AES)
- Address translation

BACKGROUND

Cloud computing services are provided by dividing compute resources into multiple virtual machines that are provided to individual customers. A confidential virtual machine (CVM) is a type of virtual machine that protects customer data by encrypting (decrypting) customer data before writing to or reading from memory. The encryption key is rooted to hardware, such that not even the cloud provider can extract customer data. Private data accessed by a CVM is stored in a region of memory referred to as confidential, encrypted, or private memory.

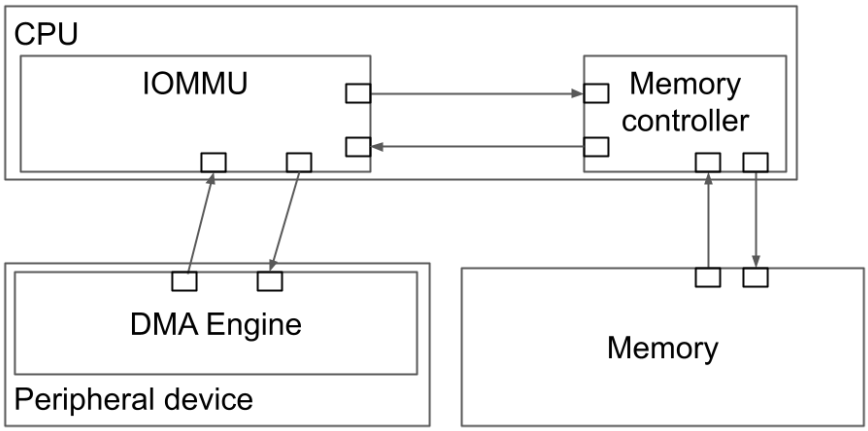

Fig. 1: Memory access by the CPU and by peripheral devices in CVMs

Fig. 1 illustrates memory access in CVMs by the CPU and by peripheral devices (also known as input-output devices) such as GPUs (graphical processing units), NICs (network interface cards), SSDs (solid-state drives), etc. The CPU, which in this context hosts the virtual machine, uses an IOMMU (input-output memory management unit) to specify to a memory controller the addresses from memory that the VM seeks. Data written to or read from the memory by the CPU is encrypted (decrypted) by an encryption engine typically located within the memory controller. When it comes to access of memory by a peripheral device, for various

reasons, e.g., relating to the width of the bus between a peripheral and the memory, etc., the physical address of a memory location can differ from the address used for that location by a peripheral device. For example, a peripheral may use the input-output address (IOA) 0x1000 to refer to the physical address (PA) location 0x5000. The IOMMU is a module that translates addresses between the PA space and the input-output address (IOA) space.

For robust encryption, the ciphertext corresponding to a given plaintext depends not only on the plaintext but also the physical address of the plaintext. A plaintext ‘ABC’ will map to one ciphertext if ‘ABC’ were stored starting at physical address 0x5000 and to another, distinct, ciphertext if ‘ABC’ were stored starting at a different physical address 0x5001. Thus, not only is the hardware-rooted key required to encrypt plaintext (to decrypt ciphertext), but also the physical address of the plaintext (ciphertext).

Since the confidential memory of the CVM uses the physical address of the data as a part of the encrypt/decrypt function, the peripheral device needs a way to ascertain the physical address for a piece of data being encrypted/decrypted. However, with the IOMMU, the peripheral device knows not the physical address of the data but only its IO address. One way to enable peripheral access to memory is to delineate a section of memory as unencrypted (also known as shared memory, as opposed to private or confidential memory), and stipulate that peripherals access only the unencrypted region of memory. Although such a scheme obviates encryption-decryption and requires the peripheral to only know the IO address (not physical address) of the data being accessed, the scheme leaves peripheral-memory communication open to attacks. Also, the scheme can suffer from bottlenecks in data flow due to the relatively small size of memory that can safely be left in plaintext.

Currently, there is no way to transfer encrypted data of a CVM to a peripheral device (e.g., GPU, NIC, SSD, etc.) without decrypting it with the hardware-rooted CVM private key, re-encrypting it with a transport key that is distinct from the hardware-rooted key used in the memory controller and sharing the transport key with the peripheral. Indeed, a PCI standard protocol can be defined to add a peripheral device (e.g., GPU, NIC, SSD, etc.) to the trusted compute base (TCB) of the CVM; establish a secure channel between the device and the CVM; establish a transport key between the device and the CVM; and enable the device to access encrypted CVM memory (e.g., via an enhanced IOMMU and DMA engine). In such a protocol, a transport key encrypts/decrypts over the secure channel while a distinct, hardware-rooted CVM private key encrypts/decrypts to the DRAM. However, to maintain confidentiality across the secure channel, the protocol would require two encryption/decryption passes when a device accesses CVM confidential memory, e.g., one pass to the CVM private key and a second pass to the transport key, resulting in a substantial reduction in computing speed.

DESCRIPTION

This disclosure describes techniques to enable a peripheral device (e.g., GPU, NIC, SSD, etc.) of a confidential virtual machine (CVM) to access the encrypted memory of the CVM securely and efficiently using just one encryption/decryption pass. The techniques enable secure, high-speed computing at low power consumption. The techniques account for address translation between input-output address space and physical address space, such that a peripheral can continue working in IOA space while encryption (decryption) can continue to depend on the physical address of the plaintext (ciphertext) to be encrypted (decrypted). The techniques obviate the use of compute-intensive keys such as transport keys subsidiary to the hardware-rooted keys

used for CVM-memory communication.

The techniques operate in two phases - CVM setup and CVM operation.

CVM setup

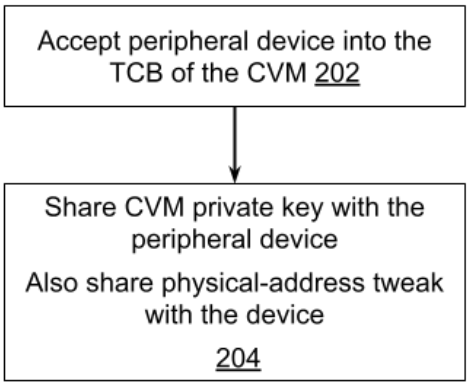

Fig. 2: CVM setup phase

During CVM setup, the hardware-rooted CVM private key is securely shared with the peripheral device. As illustrated in Fig. 2, this is done by first accepting the peripheral device into the trusted compute base (TCB) of the CVM (202), establishing a secure channel between the device and the CVM. Alongside the hardware-rooted CVM private key, the physical address tweak, e.g., the IOA-PA offset and the procedure to incorporate the offset into encryption, is also shared with the device (204). Since CVM encryption (decryption) depends on the physical address of the plaintext (ciphertext), sharing the physical address tweak enables the peripheral to access encrypted memory using DMA (direct memory access).

CVM operation at runtime

During CVM operation, the peripheral device accesses confidential memory using the CVM private key shared during the setup phase, and, if the IOMMU is on, with IOA-PA address

translation enabled by the IOMMU. CVM operation is described in greater detail below under four cases, e.g., reading/writing with IOMMU on/off.

DMA-read with IOMMU

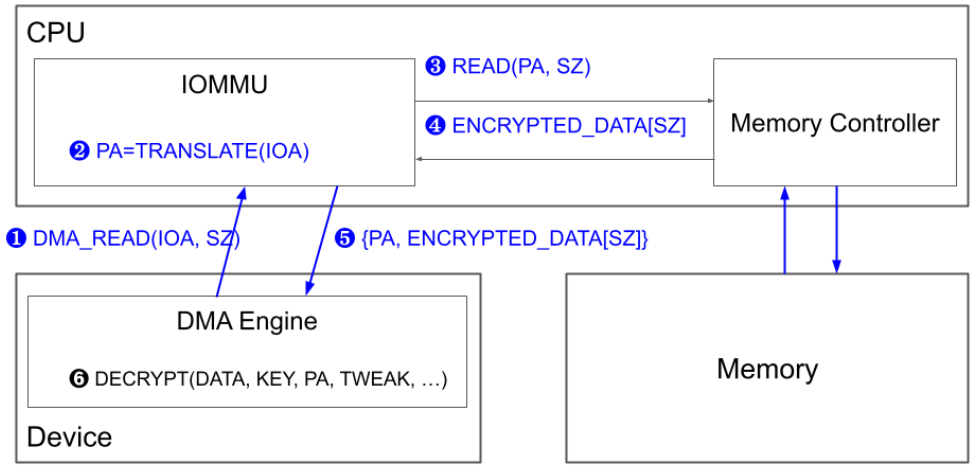

Fig. 3: A DMA-read by a peripheral device with the IOMMU on

As illustrated in Fig. 3, a DMA-read by a peripheral device with the IOMMU on takes place as follows:

- To read the private memory of the CVM via DMA, the device ❶ passes the starting IO address (IOA) and the size (SZ) of memory to be read as parameters into the DMA_READ API.
- The IOMMU ❷ translates the IO address to a physical address.
- The IOMMU ❸ signals the memory controller to read data from the CVM private memory.

- The memory controller ❹ returns the encrypted data in the memory (ENCRYPTED_DATA[SZ]) to the IOMMU.
- The IOMMU ❺ returns a two-element tuple comprising PA and ENCRYPTED_DATA[SZ].
- The device ❻ decrypts the data using the private key and PA tweak obtained during the setup phase and the physical address obtained from the IOMMU.

DMA-write with IOMMU

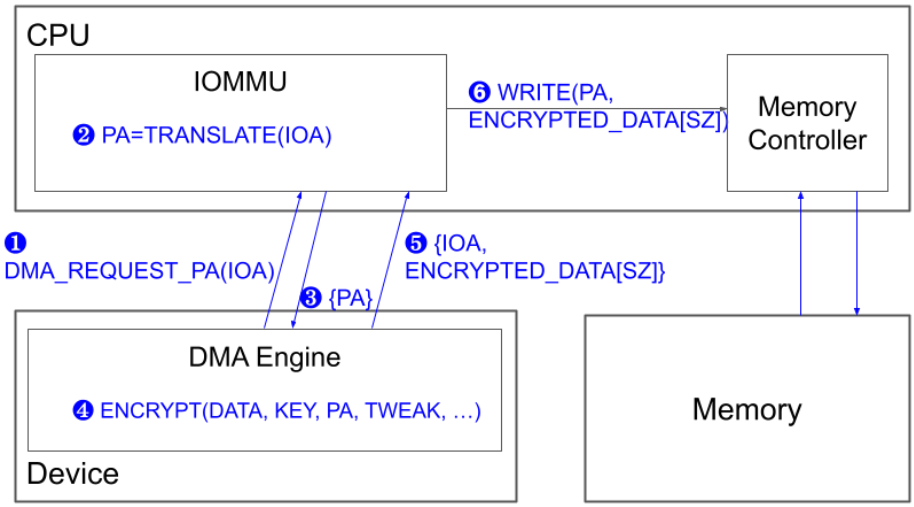

Fig. 4: A DMA-write by a peripheral device with the IOMMU on

As illustrated in Fig. 4, a DMA-write by a peripheral device with the IOMMU on takes place as follows:

- The device ❶ requests from the IOMMU the physical address for the memory region being written. This can be done via a zero-length read to the IO address being written.

- The IOMMU ❷ translates IO address (IOA) to a physical address.
- The IOMMU ❸ sends the PA to the device.
- The device ❹ encrypts the data (DATA) to be DMA-written using the private key and PA tweak obtained during the setup phase and the PA obtained from the IOMMU.
- The device ❺ sends the encrypted data (ENCRYPTED_DATA) of the required size (SZ) and starting IO address (IOA) to the IOMMU.
- The IOMMU ❷ translates the IO address to a physical address.
- The IOMMU ❻ writes the encrypted data to the CVM private memory.

DMA-read without IOMMU

If the IOMMU is not involved in memory access, then the device needs to know the physical address of the region of memory to be accessed. The device DMA-reads the CVM private memory as though it were written in plaintext (although it is not). The device decrypts the data using the physical address along with the private key and the PA tweak obtained during the setup phase.

DMA-write without IOMMU

If the IOMMU is not involved in memory access, then the device needs to know the physical address of the region of memory to be accessed. The device encrypts the data using the

physical address and the private key and PA tweak obtained during the setup phase. The encrypted data is DMA-written over the PCI bus into the private memory of the CVM.

To forestall the possibility of a device outside of the TCB of the CVM impersonating a device within the TCB, DMA-reads and the DMA-writes can be amended with cryptographic signatures that can be used by the IOMMU or memory controller to authenticate the device.

For every new IO address, the device asks for IOA-to-PA translation and stores it in its input/output translation lookaside buffer (IOTLB). Since translations are cached inside the device IOTLB, IOMMU accesses are removed from the critical path. The device accesses encrypted data directly from the memory controller.

Some CVM hardware uses error correcting code (ECC) bits (also known as integrity bits) in the private memory of the CVM to ensure that the memory has not been maliciously or accidentally overwritten. Peripheral devices typically don't have direct access to these integrity bits. Therefore, the memory controller can update the ECC bits during a DMA-write from a device within the TCB of a CVM.

In this manner, the described techniques enable encrypted access from a peripheral to confidential memory in a fast and power-efficient manner. The described techniques circumvent the complexity of maintaining in software a transport key that is distinct from the hardware-rooted private key of the CVM.

CONCLUSION

This disclosure describes techniques to enable a peripheral device of a confidential virtual machine to access encrypted memory using a single encryption pass. The techniques enable secure, high-speed computing at low power consumption. Address translation between input-output and physical address spaces is accounted for such that peripherals continue to work in

input-output address space while encryption continues to depend on the physical address of the data. The techniques obviate compute-intensive transport keys subsidiary to hardware-rooted virtual machine keys.

REFERENCES

[1] "An introduction to IOMMU infrastructure in the Linux kernel," https://lenovopress.lenovo.com/lp1467.pdf accessed May 14, 2022.

[2] Morgan, Benoît, Éric Alata, Vincent Nicomette, and Mohamed Kaâniche. "IOMMU protection against I/O attacks: a vulnerability and a proof of concept." *Journal of the Brazilian Computer Society* 24, no. 1 (2018): 1-11.